TARIYKDD: UNA HERRAMIENTA GENÉRICA DE DESCUBRIMIENTO DE CONOCIMIENTO EN BASES DE DATOS DÉBILMENTE ACOPLADA CON EL SGBD POSTGRESQL

ANDRÉS OSWALDO CALDERÓN ROMERO
IVÁN DARIO RAMIREZ FREYRE
JUAN CARLOS ALVARADO
ÁLVARO FERNANDO GUEVARA UNIGARRO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
PASTO
2007

TARIYKDD: UNA HERRAMIENTA GENÉRICA DE DESCUBRIMIENTO DE CONOCIMIENTO EN BASES DE DATOS DÉBILMENTE ACOPLADA CON EL SGBD POSTGRESQL

ANDRÉS OSWALDO CALDERÓN ROMERO
IVÁN DARIO RAMIREZ FREYRE
JUAN CARLOS ALVARADO PEREZ
ÁLVARO FERNANDO GUEVARA UNIGARRO

Trabajo de grado presentado como requesito parcial para optar el título de ingeniero de sistemas

RICARDO TIMARÁN PEREIRA, Ph.D
Director

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
PASTO
2007

Nota de aceptación:

_____
_____
_____
_____
_____
_____

_____
Firma del jurado

_____
Firma del jurado

_____
Firma del director de proyecto

Pasto, 23 de Febrero de 2007

“Las ideas y conclusiones aportadas en la tesis de grado son responsibilidad exclusiva de los autores”

Artículo 1. del acuerdo No. 324 del 11 de Octubre de 1966, emanado por el Honorable Consejo Directivo de la Universidad de Nariño.

A mi familia por su amor y confianza en mi,
aún en los momentos en los que yo me daba por vencido.
A mis compañeros de tesis y de semestre por acompañarme
y apoyarme durante mi carrera.
A mis amigos del Aula de Informática, ParqueSoft,
GRiAS y GUG+L por enseñarme a trabajar en equipo.
Al Profesor Ricardo Timarán por enseñarnos que
las fronteras no existen.
A Nancy por todo su amor y paciencia.
A todos aquellos que creen en el Software Libre
y liberan conocimiento.

Andrés O. Calderón Romero.

A papá y mamá, por ser mi ejemplo y por darme
la mano desde mi primer paso y por enseñarme a caminar por la vida.
A mis hermanos por hacerme reir y recordarme que la vida es alegria.
A mis amigos compañeros de trabajo por ayudarme a crecer.
A mi bicicleta compañera fiel en la ruta.
Al profesor Ricardo Tmarán Pereira por sus enseñanzas.

Alvaro Fernando Guevara Unigarro

Es mi deseo dedicarle este trabajo a Dios, que me permitió culminar este proyecto. A quienes me han amado, enseñado, apoyado, ayudado, y rezado por mí, los cuales se han convertido en parte de todo en cuanto hago.

A mi madre, no me alcanzarían las palabras para expresarle todo lo que significa para mi, lo que soy se lo debo a ella, a mi segunda madre, Emma María "Emita", ella representa para mi todo lo ejemplificante y bueno de la vida., a mi novia Maria Moreno, por salir de mis sueños para convertirse en mi realidad, a mi padre, Fernando Alvarado, me hace sentir orgulloso, el orgullo que siente por mi, a mi hermana Erika Viviana, gracias por depositar tanta fe en mí. A su esposo Alejandro Acosta, quien mas que un cuñado se ha convertido en otro hermano, a su bebe, mi sobrino y ahijado, Juan David Acosta, quien coloca en mi, una sonrisa de felicidad cada vez que pienso en él, a mi otra hermana July Elizabeth, su sentido del humor me subió el ánimo en muchas ocasiones. A su esposo Jesús y mi sobrina preciosa Leidy, una verdadera familia modelo.

Gracias a toda mi familia, fuente constante de apoyo y motivación: Mi abuelito Juan Florencio, mis tios Myrian, Adriana, Jaime y Eriberto, a mis primos Harold Steven, Jhon Jario, Juan Camilo, Jesica, Jannis, Diego y David.

A todas mis amistades, especialmente a mis amigos de toda la vida: Juan Carlos Arteaga, Mario Timana, Andres Hernandez y Miguel Angel Lucero, gracias por brindarme ese tesoro invaluable, su amistad, a la Capoeira, me permitió aclarar la mente en los momentos mas agobiantes, gracias a mis maestros y amigos Galego y Navalha.

Al Doctor Ricardo Timaran, PhD, su luz no sólo guió nuestro camino en la investigación, sus enseñanzas me acompañarán el resto de la vida.

A todos ellos los cuales me ayudaron con su apoyo incondicional a ampliar mis conocimientos y estar más cerca de mis metas profesionales

Juan Carlos Alvarado Perez

Después de la dedicación y el esfuerzo llega el momento de la satisfacción en el que uno ve sus objetivos cumplidos, entonces nos damos cuenta de que no en vano dedicamos nuestro tiempo a nuestros proyectos. Por eso en este pequeño pero grato escrito y por respeto a quienes me acompañaron quiero agradecerles por todo lo compartido.

En primer lugar a mi equipo de trabajo con quienes más que una tesis construimos una amistad, de ustedes aprendí cosas muy valiosas como son: la tranquilidad, seguridad, pasión y entrega en las cosas que uno hace.

A mi Mamá, a mi Papá y a toda mi familia, quienes hombro a hombro me ayudaron a conseguir este logro, quienes me dieron ánimos en los momentos de angustia y con quienes compartimos momentos de alegría cuando había que celebrar.

A toda la gente de ParqueSoft, amantes de sus labores y entregados a la idea de ser emprendedores.

Una mención especial a mis amigos del barrio y a mis compañeros de curso, a quienes así como a mi familia les debo muchos momentos aplazados. (No puedo dejar de nombrar a mi bicicleta quien me condujo por 5 años a todo lugar, gracias.)

Y a todos los profesores de la Facultad de Ingeniería quienes dedicaron tiempo y esfuerzo en prepararnos como profesionales. Al doctor Ricardo Timarán por su valioso acompañamiento en este trabajo y un muchas gracias por todo lo aprendido a los profesores de la Facultad de Idiomas.

Este proyecto y este título no es un último peldaño, es otra meta propuesta que fue llevada a buen termino, De nuevo gracias a todas las personas que de una u otra forma tuvieron que ver en este trabajo.

Iván Dario Ramirez Freyre

AGRADECIMIENTOS

Quienes nos unimos para desarrollar este proyecto aportando trabajo con amor y desinterés, queremos ofrecerlo a todos sin distingo de raza, religión, sexo, condición política y social. Respetamos la vida y fomentamos la oportunidad que tenemos de dar y aprender. Hacemos todo con las mejores intenciones, trabajando transparentemente con respeto y humildad, sin forzar a otros a creer en lo que creemos, ofreciendo de corazón, lo que para nosotros es la filosofía del “Software Libre”.

Agradecimientos especiales al sistema de investigaciones, al profesor Carlos Cordoba Barahona, quien a parte de brindarnos sus conocimientos fue trascendental para la consecución de los objetivos , al director del programa de ingeniería de sistemas Nelson Jaramillo siempre dispuesto a ofrecernos su ayuda y amistad, a ParqueSoft, incubadora de sueños y esperanzas, en donde el emprendimiento y la sinergia de sus miembros son el motor que impulsa las ilusiones de los mismos, a GRIAS, templo de investigación que permitió la consolidación de tantas ideas, las cuales hicieron posibles el desarrollo de este proyecto, a Claudia Castro y Aleida Cabrera, compañeras de lucha en la minería de datos.

A nuestro asesor el Doctor Ricardo Timaran Pereira, PhD, el cua sembró y fertilizó la semilla de la inquietud por conocer e investigar.

CONTENIDO

LISTA DE FIGURAS

pág

LISTA DE TABLAS

Tabla pág

Resumen

En este trabajo se presenta el análisis, diseño e implementación de TariyKDD, una herramienta genérica para el Descubrimiento de conocimiento en bases de datos, débilmente acoplada con el SGBD PostgreSQL.

TariyKDD comprende cuatro módulos que cubren la conexión, a archivos planos y bases de datos relacionales, un módulo de utilidades con clases y librerías comunes, un módulo kernel que reune las etapas de preprocesamiento, minería y visualización, y el módulo de interfaz gráfica de usuario.

Dentro del kernel de minería se implementan 5 algoritmos, *Apriori, FPGrowth y EquipAsso* para asociación y *C4.5 y Mate* para clasificación.

Se evalúa el rendimiento de los algoritmos *EquipAsso*, un algoritmo para el cálculo de conjuntos de ítems frecuentes, y *Mate*, un algoritmo para la construcción de arboles de clasificación, basados en nuevos operadores del álgebra relacional, con respecto a los algoritmos *Apriori* y *FP-Growth* y *C4.5,* respectivamente.

Abstract

In this work is presented the analysis, design and implementation of TariyKDD, a generic tool for the Knowledge Discovery in Data bases, loosely coupled with the DBMS PostgreSQL.

TariyKDD includes the development of four modules that cover the connection, to text files and relational databases, a module of utilities with common classes and libraries, a kernel module that gathers the pre-processing, mining and visualization stages, and the graphical user interface module.

Within the mining kernel are implemented 5 algorithms, *Apriori*, *FPGrowth* and *EquipAsso* for association, and *C4.5* and *Mate* for classification.

The performance of both, EquipAsso, an algorithm for the calculation of sets of frequent items, and Mate, an algorithm for the construction of trees of classification, based on new operators of the relational algebra, are evaluated with respect to the algorithms Apriori, FP-Growth and C4.5, respectively.

1. INTRODUCCION

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 [42].

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, 2, 7, 23, 31]. 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 [41].

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 [31].

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 [43, 42, 45] y MateTree [46] para las tareas de Asociación y Clasificación propuestos por Timarán en [46] 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 primera 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, luego se especifican los objetivos generales y específicos del proyecto y finalmente la justificación. En la sección 2 se presenta el estado general del arte en el área de integración de DCBD y SGBD. En la sección 3 se desarrolla todo lo concerniente al análisis orientado a objetos UML que se realizo para construir la herramienta, en la sección 4 se presenta la implementación del proyecto, en la sección 5 se presenta las pruebas hechas y resultados, y finalmente en la sección 6 se presenta las conclusiones de este trabajo.

1.1 TEMA

1.1.1 TITULO TariyKDD. Una herramienta genérica para el descubrimiento de conocimiento en bases de datos, débilmente acoplada con el SGBD PostgreSQL.

1.1.2 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.

1.1.3 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, 2, 21]. En la etapa de minería de datos se implementaron las tareas de asociación y clasificación. En éstas dos tareas, se utilizaron los operadores algebráicos relacionales y primitivas SQL para DCBD, desarrollados por Timarán [44, 45], con los algoritmos EquipAsso [43, 42, 45] y Mate-tree [46]. En la etapa de visualización se desarrollo una interfaz gráfica que le permite al usuario interactuar 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[2], FP-Growth [19, 23] y C.4.5 [32, 30].

1.2. PROBLEMA OBJETO DE ESTUDIO

1.2.1 Descripción del problema. Muchos investigadores [7, 6, 19, 37] 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 [41].

Algunas aplicaciones como Alice [25], C5.0 RuleQuest [37], Qyield [8], CoverStory [28] 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 [26], especialmente para clasificación apoyadas en árboles de decisión, redes neuronales y aprendizaje basado en ejemplos. El usuario de este tipo de programas puede integrarlos a otros módulos como parte de una aplicación completa

[31]. 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 [31]. En este grupo están Clementine [39], DBMiner [20, 17, 16], DBLearn [18], Data Mine [23], IMACS [5], Intelligent Miner [9], Quest [4] entre otras. Una evaluación de un gran número de herramientas de este tipo se puede encontrar en [15].

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 aplicativos 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 programas sin ningún tipo de restricciones, a las pequeñas y medianas empresas u organizaciones de nuestro país o de cualquier parte del mundo.

1.2.2 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, facilitaría a las pequeñas y medianas empresas la toma de decisiones?

1.3. OBJETIVOS

1.3.1 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.

1.3.2 Objetivos especificos.

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.

1.4 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.

2. MARCO TEORICO

2.1 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 con 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 [11].

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.

2.2 ARQUITECTURAS DE INTEGRACION 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 [41].

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 [41].

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 [41].

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 [41].

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.

2.3 IMPLEMENTACION DE HERRAMIENTAS DCBD DEBILMENTE ACOPLADAS CON UN SGBD

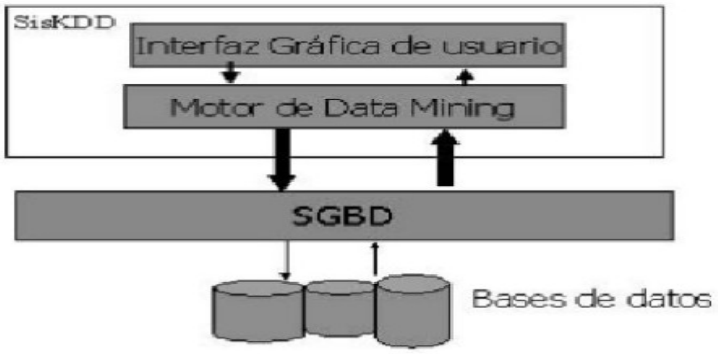
La implementación de herramientas DCBD débilmente acopladas con un SGBD se hace a través de SQL embebido en el lenguaje anfitrión del motor de minería de datos [1]. Los datos residen en el SGBD y son leídos registro por registro a través de ODBC, JDBC o de una interfaz de cursores SQL. La ventaja de 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 [6, 22] 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 2.1).

2.4 ALGORITMOS DE MINERIA DE DATOS

Los algortmos de mineria de datos son el núcleo principal del proceso KDD, dependido de su objetivo pueden ser clasificados en tareas de minería de datos, como lo son asociación y clasificación.

Dentro de asociación existen algoritmos como: Apriori, FPGrowth y EquipAsso, así como en clasificación: C.4.5 y MateBy, los cuales son explicados a continuación:

Figura 2.1. Arquitectura DCBD débilmente acoplada



2.4.1 Apriori. La notación del algoritmo Apriori [2] es la siguiente:

Tabla 2.1. 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:

L1 = { Conjunto de itemsets frecuentes 1 }
for (k=2; $L_{k-1} \neq 0$; k++) do begin
Ck = apriori-gen(L_{k-1}) // Nuevos candidatos
forall transacciones t ∈ D do begin
C_t = subconjunto (C_k, t); // Candidatos en t
forall candidatos c ∈ C_t do
c.cont++;
end
L_k = { c ∈ C_k|c.count ≥ minsup }
end Answer $U_k L_k$

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 Ck
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 € Ck tal que algún subconjunto (k − 1) de c no este en Lk−1:

forall itemsets c $\in$ C_k do
forall subconjuntos (k − 1) s de c do
if (s $\notin$ L_{k-1}) then
delete c from C_k;

2.4.2 FPGrowth. Como se puede ver en [22] 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 104 Itemsets Frecuentes, el algoritmo Apriori necesitará generar más de 107 Itemsets Candidatos, así como acumular y probar su ocurrencia. Además, para descubrir un patrón frecuente de tamaño 100, como a1,…, a100, se debe generar más de 2100 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 [22]. 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. Unicamente 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 dramáticamente 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 búsqueda.

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, Tabla 2.2 y ξ=3. Una estructura de datos puede ser diseñada de acuerdo con 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.

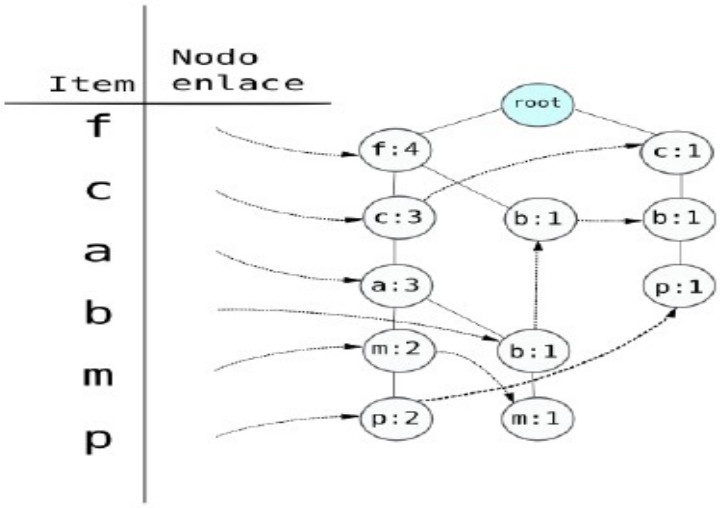
3. Si múltiples transacciones comparten un conjunto idéntico de items frecuentes, estas pueden ser fusionadas en una sola, con el número de ocurrencias como contador.

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.

Tabla 2.2: 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 2.2: Tabla de cabeceras y Árbol 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 únicoprefijo 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 idéntica 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 construída, 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 2.2.

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

- Consiste de una raíz etiquetada como null, un conjunto de árboles hijos de la raíz 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 ai, todos los posibles patrones frecuentes que contenga ai pueden ser obtenidos siguiendo los nodos enlazados de ai, comenzando desde ai 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 2.2. De acuerdo a la propiedad de nodos enlazados para obtener todos los patrones frecuentes de un nodo ai, se comienza desde la cabeza de ai (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, únicamente 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 mes (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 2.3 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 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 tabla 2.3 se muestran los Patrones Condicionales Base y los FP-trees generados.

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)}.

Figura 2.3: FP-tree condicional para m

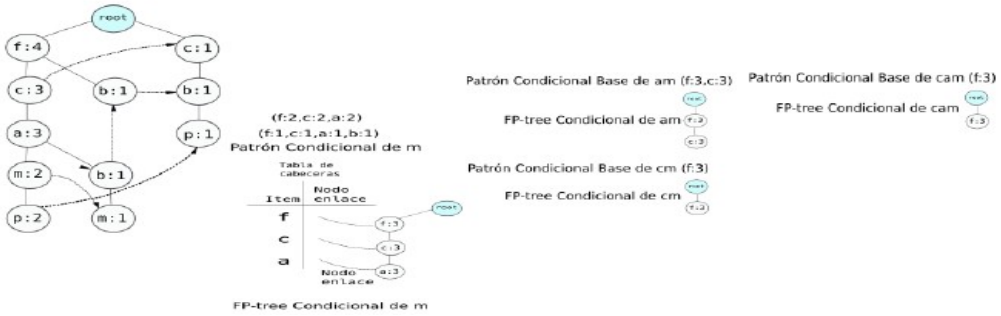


Tabla 2.3: Patrones condicionales base y FP-trees condicionales

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

2.4.3 EquipAsso nuevos operadores del algebra relacional para asociación.

- Operador Associator (α). Associator(α) es un operador algebraíco 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 mayor de operaciones en el proceso de optimización de una consulta.

  Su sintaxis es la siguiente:

  $$\alpha_{\text{tam inicial, tam final}}(R)$$

  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 Xi , de tamaño tam inicial hasta tamaño tam f inal , 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.

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

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 vi (A_k) de los atributos de cada tupla ti de R. En cada tupla X_i unicamente 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 2.4.

- 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 unicamente los valores de los atributos que satisfacen una expresión lógica.

  Su sintaxis es la siguiente: χexpresion logica (R)

  El operador EquiKeep restringe los valores de los atributos de cada una de las tuplas de la relación R a unicamente 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 o expresión lógica integrada por cláusulas de la forma A_i=const unidas por los operadores booleanos AND (^), OR (v), NOT (¬). El operador χ aplicado a la relación R con la expresión lógica p:

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

produce una relación de igual esquema R(A) de grado n y cardinalidad m , donde m≤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.

Ejemplo 2. Sea la relación R(A,B,C,D) y la operación $\chi_{A=a1vB=b1vC=c2vD=d1}$ (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

- 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 f recuentes};
forall transacciones t ∈ D do begin
    R = χL1 (D)
    K=2
    g = grado(R)
    R = α_{k,g} (R)
end
Lk = {count(R') | c.count ≥ minsup}
Respuesta = ∪Lk ;
```

2.4.4 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:

μ 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.

Tabla 2.4: Resultado de R1 = $\alpha_{2,4}$ (R)

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

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 ≠ [] 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:

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

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 unicamente 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 la 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 Tabla 2.5 obtener las diferentes combinaciones de los atributos A,B con el atributo D, es decir, R1 = $\mu_{A,B;D}$ (R).

Tabla 2.5: Relación R

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

El resultado de la operación R1 = $\mu_{A,B;D}$ (R), se muestra en el Tabla 2.6:

Función algebraíca 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.

Tabla 2.6: Resultado de la Operación R1 = $\mu_{A,B;D}$ ®

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

Formalmente, sea A = {A1, ..., An, Ac} el conjunto de atributos de la relación R o 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 :

$$Entro(A_k; Ac; R) = \{y|y = - p_{ij} \log2 (p_{ij}), i = 1..t, j = 1..q, p_{ij} = s_{ij} /|S_j |\}$$

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:

$$Entro(Ac; Ac; R) = \{y|y = - p_i \log2 (p_i), i = 1..t, p_i = r_i /m\}$$

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

Función algebraíca 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:

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

Operador Describe Classifier ([]μ) Describe Classifier ([]μ) 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:

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

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:

N null, P = null, Anull, V = null y C = null corresponde a un nodo raiz; si N null, P null, Anull, V null y Cnull corresponde a una hoja o nodo terminal y si N null, 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 Tabla 2.7 resultado del operador Describe Classifier. Construir el árbol de decisión.

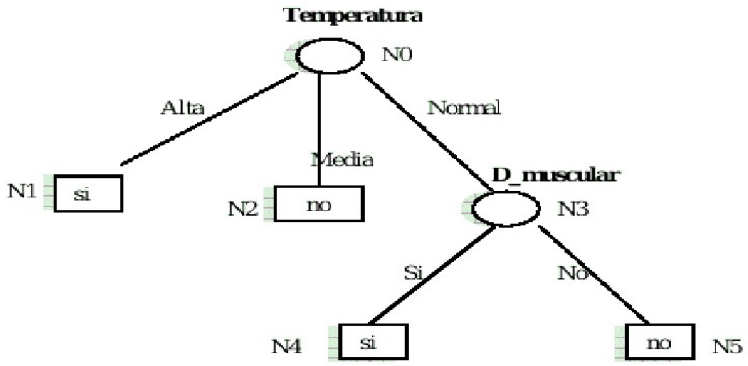
2.5 PODA DE LOS ARBOLES GENERADOS

La poda de los árboles generados por los algoritmos de clasificacion en mineria de datos, se hace necesaria por diversas razones. Entre las cuales podemos mencionar la sobregeneralización, la evaluación de atributos no relevantes o insignificantes y el gran tamaño que árbol obtenido pueda alcanzar.

Tabla 2.7: Relación Arbol

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

Figura 2.4: Árbol de decisión:



El caso de la sobregeneralización se presenta cuando un árbol puede haber sido construido a partir de de un conjunto de datos con ruido (noise data), por lo cual algunas ramas del árbol pueden ser engañosas o inciertas.

En cuanto a la evaluación de atributos intrancendentes, éstos deben eliminarse mediante poda ya que sólo agregan niveles en elárbol y no contribuyen a la ganancia de información.

Por último, si el árbol obtenido es demasiado profundo o demasiado frondoso, se dificultara la interpretación de las reglas obtenidas por parte del analista,

Existen dos tipos de metodologia para la poda de los árboles: la pre-poda (preprunning) y la post-poda (postprunning).

En pre-poda (preprunning), la construccion del árbol se efectúa al mismo tiempo que se aplica el metodo de poda, de esta forma el crecimiento del árbol se detiene cuando la ganancia de información producida sobre un conjunto se parcializa al superar un umbral determinado.

En post-poda (postprunning), se efectúa una vez que se ha terminado de construir el árbo completamente, podando algunas ramas.
Pre-poda (preprunning), tiene como ventaja que no se pierde tiempo en construir una estructura que, más tarde será simplificada en el árbol definitivo. El objetivo

específico del analista en este caso, sera buscar el mejor límite (threshold) que dividirá el subconjunto, para evaluar la partición con diferentes técnicas, como son: con enfoque estadístico, mediante la teoría de la ganancia de información, reducción de errores, etc. Si esta evaluación es menor que el límite predeterminado, dicha división se descartara y el árbol parara el subconjunto con la hoja más apropiada. Sin embargo, este tipo de método tiene la contra de que no es sencillo detener el particionamiento en el momento idoneo, ya que un límite muy alto no permitirá la extracción de conocimiento idoneo, por otro lado, un límite demasiado bajo resultará en una simplificación insignificante.

En post-poda (postprunning) es un proceso regresivo a través del árbol de decisión, partiendo de las hojas y llegando hasta el nodo raíz, podando ramas a partir de criterios propios de cada uno de los algoritmos, como la ganancia de información.

2.6 HERRAMIENTAS DE MINERIA DE DATOS DEBILMENTE ACOPLADAS

En el mercado existen algunas herramientas libres de minería de datos, débilmente acopladas con un sistema gestor, desarrolladas por universidades y centros de investigación, las cuales permiten ver el estado actual de este tipo de aplicaciones. A continuación se describen las herramientas estudiadas.

2.6.1 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 [54] 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 [53].

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 CSV 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.

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 [51].

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 metodología 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 [15].

2.6.2 ADaM - Algorithm Development and Mining System. El proyecto ADaM es desarrollado por el Centro de Sistemas y Tecnología 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 [50].
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 [49] 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 [51] que son generados por sus componentes.

Dentro de las herramientas ofrecidas por ADaM se encuentran: soporte al preprocesamiento de datos y de imágenes, clasificación, clustering y asociación. La visualización 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 diversas 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 o ADaM puede verse en [30] donde se utilizan componentes ADaM en el análisis e interpretación de imágenes satelitales de ciclones para estimar la máxima velocidad de los vientos.

2.6.3 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 [12].

Orange [10] 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 decisión 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 interfáz 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 extensión **.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 módulo para la conexión a un Sistema Gestor de Bases de Datos, pero si se revisa bien la documentación de los módulos se encuentra que ha sido desarrollado uno para la conexión a MySQL (orngMySQL [13]). El módulo provee una entrada a MySQL, a través de este y de sencillos comandos en Python los datos de las tablas pueden transferidos desde y hacia MySQL. Así mismo, programadores independientes han desarrollado varios módulos 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 [12] 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 [12] 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.

2.6.4 TANAGRA - A Free Software for Research and Academic Purposes. TANAGRA [35] 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 Ingènierie des Connaissances [47]) 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 arquitectura que le 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 librería 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 Construcción de Modelos métodos para la construcción de modelos de clasificación, que incluyen árboles de clasificación, clasificadores Naive-Bayes y 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 [34], el cual contiene tutoriales, referencias y conjuntos de datos.

2.6.5 AlphaMiner. AlphaMiner es desarrollado por el E-Business Technology Institute (ETI) de la Universidad de Hong Kong [24] bajo el apoyo del Fondo para la

Innovación y la Tecnología (ITF) [14] 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 sea 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, además 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. Alpha-Miner proporciona las Tecnología 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 OpenSource,

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 item.

2.6.6 YALE - Yet Another Learning Environment. YALE [47, 29] 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 [48] 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.

Los procesos de descubrimiento de conocimiento son vistos frecuentemente como invocaciones secuenciales de métodos simples. Por ejemplo, después de cargar datos, se 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 ahí, 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 vértice del gráfico corresponde a un operador simple. Si dos operadores son conectados, la salida del primer operador sería usada como entrada en el segundo operador. Por un lado, diseñar procesos de descubrimiento de conocimiento con la ayuda de gráficos direccionados 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 a la aplicación de un filtro de preprocesamiento. Los nodos interiores del árbol corresponden a más complejos o abstractos pasos en el proceso. Esto 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 raíz 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.

2.7 HERRAMIENTAS DE DESARROLLO PARA APLICACIONES

Entre las principales herramientas de desarrollo podemos encontrar al 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.

2.7.1 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 creá con cinco objetivos principales:

1. Deberá usar la metodología de la programación orientada a objetos.
2. Deberá permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Deberá incluir por defecto soporte para trabajo en red.
4. Deberá diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Deberá 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 están 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, deberá 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. Se puede 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 librería 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 unicamente 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.

2.7.2 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).

2.7.3 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.

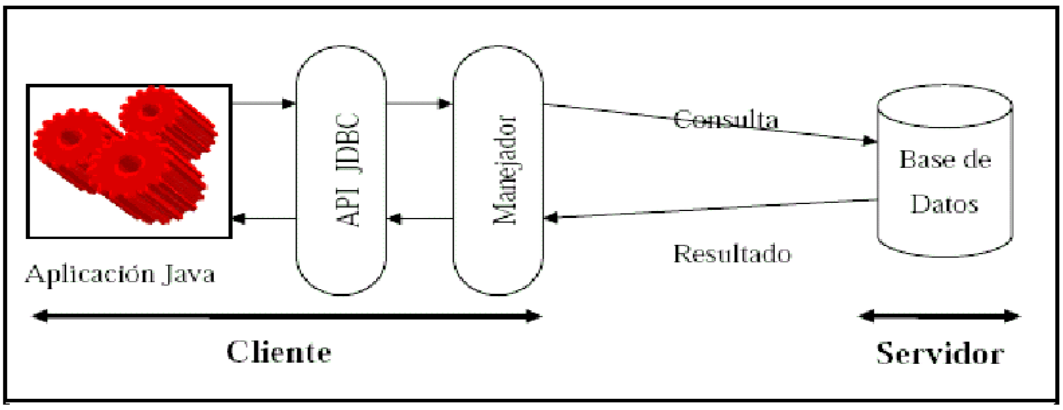
El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan 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 se esta accediendo, ya que se usan comandos estándar. Estos comandos serán traducidos por el manejador a comandos propietarios de cada sistema de BD concreto. Si se quiere cambiar el sistema de BD empleado lo unico que se debe hacer es reemplazar el antiguo manejador por el nuevo.

2.7.4 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.

Figura 2.5: Controlador JDBC tipo 4



2.7.5 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.

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 tecnología de las Java Foundation Classes. Además de los componentes ligeros suministrados originalmente por la IFC, Swing introdujo un mecanismo que permitió 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 indice 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 JS-plitPane 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 ventana sobre 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

- 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 despejar 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.

2.8 ANALISIS, DISEÑO Y MODELADO

En todas las disciplinas de la Ingeniería se hace evidente la importancia de los modelos ya que describen el aspecto y la conducta de "*algo*". Ese "*algo*" puede existir, estar en un estado de desarrollo o estar, todavía, en un estado de planeación.

Es en este momento, cuando los diseñadores del modelo deben tanto investigar como analizar los requerimientos del producto terminado y dichos requerimientos pueden incluir áreas, tales como: funcionalidad, *performance* y confiabilidad. Además, a menudo, el modelo es dividido en un número de vistas, cada una de las cuales describe un aspecto específico del producto o sistema en construcción.

2.8.1 Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*). El modelado sirve no solamente para los grandes sistemas, aun en aplicaciones de pequeño tamaño se obtienen beneficios de modelado, sin embargo es un hecho que entre más grande y más complejo es el sistema, más importante es el papel de que juega el modelado por una simple razón: "El hombre hace modelos de sistemas complejos porque no puede entenderlos en su totalidad".

Es asi como UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

El punto importante para notar aquí es que UML es un "lenguaje" para especificar y no un método o un proceso. UML se usa para definir un sistema de software; para detallar los artefactos en el sistema; para documentar y construir, es el lenguaje en el que está descrito el modelo. UML se puede usar en una gran variedad de formas para soportar una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar.

UML preescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales se puede modelar sistemas.

- Diagramas de Casos de Uso para modelar los procesos.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.

- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

2.8.2 Diagramas de Clases. Las *clases* representan los bloques de construcción más importantes de cualquier sistema orientado a objetos. Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.

Los Diagramas de Clases son utilizados durante el proceso de Análisis y Diseño de los sistemas informáticos donde se crea el diseño conceptual de la información que se manejara en el sistema, los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones.

Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones.

Un diagrama de clases esta compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

2.8.3 Diagrama de Paquetes. Cualquier sistema grande se debe dividir en unidades más pequeñas, de modo que las personas puedan trabajar con una cantidad de información limitada, a la vez y de modo que los equipos de trabajo no interfieran con el trabajo de los otros.

Un paquete es una parte de un modelo. Cada parte del modelo debe pertenecer a un paquete. Pero para ser funcional, la asignación debe seguir un cierto principio racional, tal como funcionalidad común, implementación relacionada y punto de vista común. UML no impone una regla para componer los paquetes.

Los paquetes ofrecen un mecanismo general para la organización de los modelos/subsistemas agrupando elementos de modelado. Cada paquete

corresponde a un submodelo (subsistema) del modelo (sistema). Los paquetes son unidades de organización jerárquica de uso general de los modelos de UML. Pueden ser utilizados para el almacenamiento, el control de acceso, la gestión de la configuración y la construcción de bibliotecas que contengan fragmentos reutilizables del modelo.

Un paquete puede contener otros paquetes, sin límite de anidamiento pero cada elemento pertenece a (está definido en) sólo un paquete. El Diagrama de Paquetes, muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido.

Los paquetes se dibujan como rectángulos, las dependencias se muestran como flechas con líneas discontinuas. El operador "::" permite designar una clase definida en un contexto distinto del actual.

2.8.4 Diagrama de casos de uso. Este diagrama muestra el comportamiento del sistema visto desde la perspectiva del usuario, indica la funcionalidad a cumplir, es una especie de diagrama de comportamiento.

UML define una notación gráfica para representar casos de uso llamada modelo de casos de uso, representados por elipses y los actores están representados por las *figuras humanas*. Una caja define los limites del sistema, por ejemplo, los casos de uso se muestran como parte del sistema que está siendo modelado, los actores no.

Existen 3 relaciones principales entre los casos de uso:

- Include. En una forma de interacción, un caso de uso dado puede "incluir" otro. El primer caso de uso a menudo depende del resultado del caso de uso incluido. Esto es util para extraer comportamientos verdaderamente comunes desde múltiples casos de uso a una descripción individual. La notación es una flecha rayada desde el caso de uso que lo incluye hasta el caso de uso incluido, con la etiqueta "«include»". Este uso se asemeja a una expansión de una macro donde el comportamiento del caso incluído es colocado dentro del comportamiento del caso de uso base. No hay parámetros o valores de

retorno.

- Extend. En otra forma de interacción, un caso de uso dado, (la extensión) puede *extender* a otro. Esta relación indica que el comportamiento del caso de uso extensión puede ser insertado en el caso de uso extendido bajo ciertas condiciones. La notación es una flecha rayada desde el caso de uso extensión al caso de uso extendido, con la etiqueta «extend». Esto puede ser útil para lidiar con casos especiales, o para acomodar nuevos requisitos durante el mantenimiento del sistema y su extensión.
- Generalization. En la tercera forma de relación entre casos de uso, existe una relación generalización/especialización. Un caso de uso dado puede estar en una forma especializada de un caso de uso existente. La notación es una línea solida terminada en un triángulo dibujado desde el caso de uso especializado al caso de uso general. Esto se asemeja al concepto orientado a objetos de sub-clases, en la práctica puede ser útil factorizar comportamientos comunes, restricciones al caso de uso general, descríbelos una vez, y enfrentate a los detalles excepcionales en los casos de uso especializados.

2.8.5 Diagrama de Secuencia. Es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de caso de uso permite el modelado de una vista del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos.

Típicamente, se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si tienes modelada la descripción de cada caso de uso como una secuencia de varios pasos, entonces puedes "caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos. Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como vectores horizontales. Los mensajes se dibujan cronológicamente desde la parte superior del diagrama a la parte inferior; la distribución horizontal de los objetos es arbitraria.

Durante el análisis inicial, el modelador típicamente coloca el nombre de un mensaje en la línea del mensaje. Más tarde, durante el diseño, el nombre es reemplazado con el nombre del método que está siendo llamado por un objeto en el otro. El método llamado, o invocado, pertenece a la definición de la case instanciada por el objeto en la recepción final del mensaje.

2.8.6 Diagramas de Colaboración. Diagrama que muestra interacciones organizadas alrededor de los roles. A diferencia de los diagramas de secuencia, los diagramas de

colaboración muestran explícitamente las relaciones de los roles. Por otra parte, un diagrama de colaboración no muestra el tiempo como una dimensión aparte, por lo que resulta necesario etiquetar con números de secuencia tanto la secuencia de mensajes como los hilos concurrentes.

Un diagrama de colaboración es también un diagrama de clases que contiene roles de clasificador y roles de asociación en lugar de sólo clasificadores y asociaciones. Los roles de clasificador y los de asociación describen la configuración de los objetos y de los enlaces que pueden ocurrir cuando se ejecuta una instancia de la colaboración. Cuando se instancia una colaboración, los objetos están ligados a los roles de clasificador y los enlaces a los roles de asociación. El rol de asociación puede ser desempeñado por varios tipos de enlaces temporales, tales como argumentos de procedimiento o variables locales del procedimiento. Los símbolos de enlace pueden llevar estereotipos para indicar enlaces temporales.

Un diagrama de colaboración muestra la implementación de una operación. La colaboración muestra los parámetros y las variables locales de la operación, así como asociaciones más permanentes. Cuando se implementa el comportamiento, la secuencia de los mesajes corresponde a la estructura de llamadas anidadas y el paso de señales del programa.

Un diagrama de secuencia muestra secuencias en el tiempo como dimensión geométrica, pero las relaciones son implícitas. Un diagrama de colaboración muestra relaciones entre roles geométricamente y relaciona los mensajes con las relaciones, pero las secuencias temporales están menos claras.

Mensajes Los mensajes se muestran como flechas etiquetadas unidas a los enlaces. Cada mensaje tiene un número de secuencia, una lista opcional de mensajes precedentes, una condición opcional de guarda, un nombre y una lista de argumentos y un nombre de valor de retorno opcional. El nombre de serie incluye el nombre (opcional) de un hilo. Todos los mensajes del mismo hilo se ordenan secuencialmente. Los mensajes de diversos hilos son concurrentes a menos que haya una dependencia secuencial explícita.

- Flujos, generalmente un diagrama de colaboración contiene un símbolo para un objeto durante una operación completa. Sin embargo, a veces, un objeto contiene diferentes estados que se deban hacer explícitos. Por ejemplo, un objeto pudo cambiar de localización o sus asociaciones pudieron diferenciarse.

- Los diferentes símbolos de objeto que representan un objeto se pueden coenctar usando flujos "become" o "conversion". Un flujo "become" es una transición, a aprtir de un estado de un objeto a otro. Se dibuja como una flecha

de línea discontinua con el estereotipo "become" o "conversion" y puede ser eqtiquetado con un número de serie para mostrar cuando ocurre. Un flujo de conversión también se utiliza para mostrar la migración de un objeto a partir de una localización a otra distinta.

2.8.7 El Proceso Racional Unificado o RUP (Rational Unified Process). Los orígenes de RUP se remontan al modelo espiral, es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP es en realidad un refinamiento realizado por Rational Software del más genérico Proceso Unificado.

Sus principales características, son:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo)
- Pretende implementar las mejores prácticas en Ingeniería de Software
  - Desarrollo iterativo
  - Administración de requisitos
  - Uso de arquitectura basada en componentes
  - Control de cambios
  - Modelado visual del software
  - Verificación de la calidad del software

El RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, etc.) y roles (papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso).

El RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al final de cada ciclo, cada ciclo se divide en fases que finalizan con un hito donde se debe tomar una decisión importante:

- inicio: se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos
- elaboración: se hace un plan de proyecto, se completan los casos de uso y se

eliminan los riesgos

- construcción: se concentra en la elaboración de un producto totalmente operativo y eficiente y el manual de usuario

- transición: se implementa el producto en el cliente y se entrena a los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados.

- fases del RUP:
    - Establece oportunidad y alcance
    - Identifica las entidades externas o actores con las que se trata
    - Identifica los casos de uso

3. DESARROLLO DEL PROYECTO

Para el análisis y diseño de TariyKDD se utilizo el RUP (Rational Unified Process) con la notación UML.

3.1 ANALISIS UML

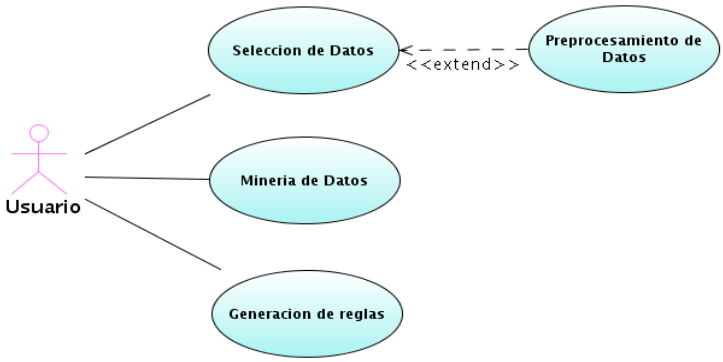
3.1.1 Funciones

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 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.		Obligatorio
R3.1.2	Cargar el conjunto de datos.	Oculto.	Información	Establecer relaciones correctas	Obligatorio
R3.2	Autorizar conexiones a Archivos Planos.	Evidente.	Interfaz.		Opcional.
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 da- tos nulos	Evidente.	Interfaz.		Opcional.

Ref#	Función	Cat.	Atributo	Detalles y Restricciones	Cat.
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.
R5.2	Implementar algoritmo EquipAsso	Evidente.	Interfaz.		Opcional.
R5.2.1	Proveer algoritmos que cumplan tareas de clasificación.	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

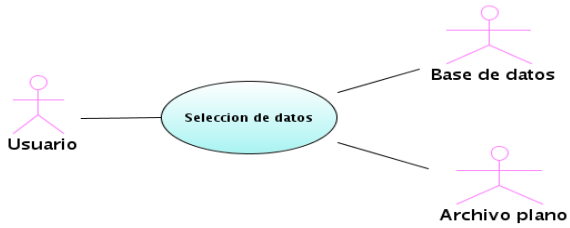
3.1.2 Diagramas de Casos de Uso

Figura 3.1: Tariy



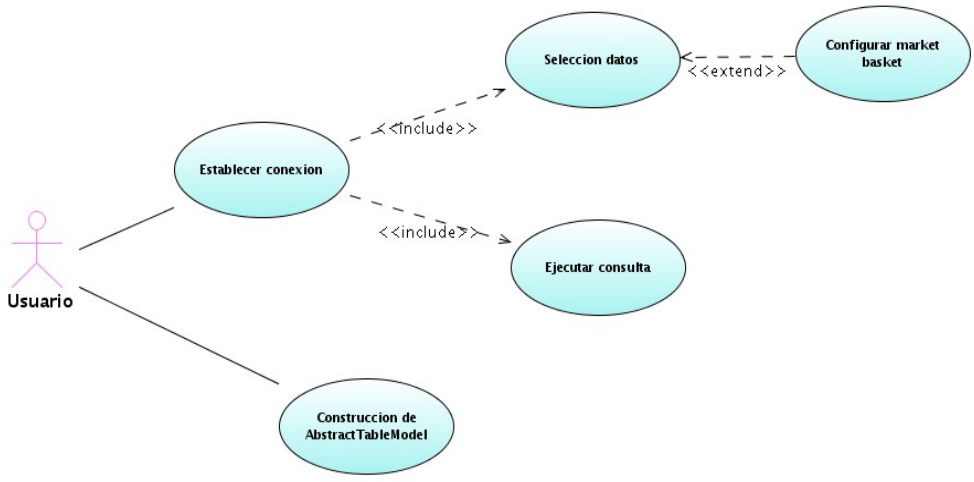
Módulo de Selección

Figura 3.2: Módulo de Selección



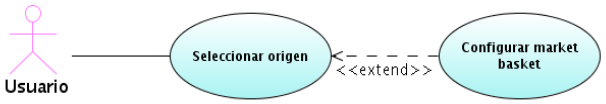
Módulo de Conexión a Base de Datos

Figura 3.3: Base de Datos



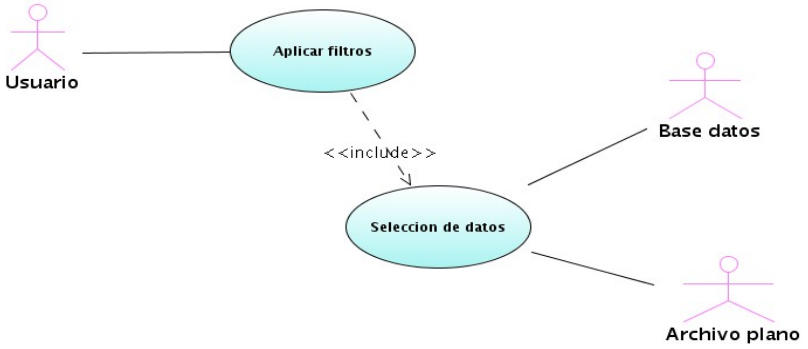
Módulo de Conexión a Archivo Plano

Figura 3.4: Archivo Plano



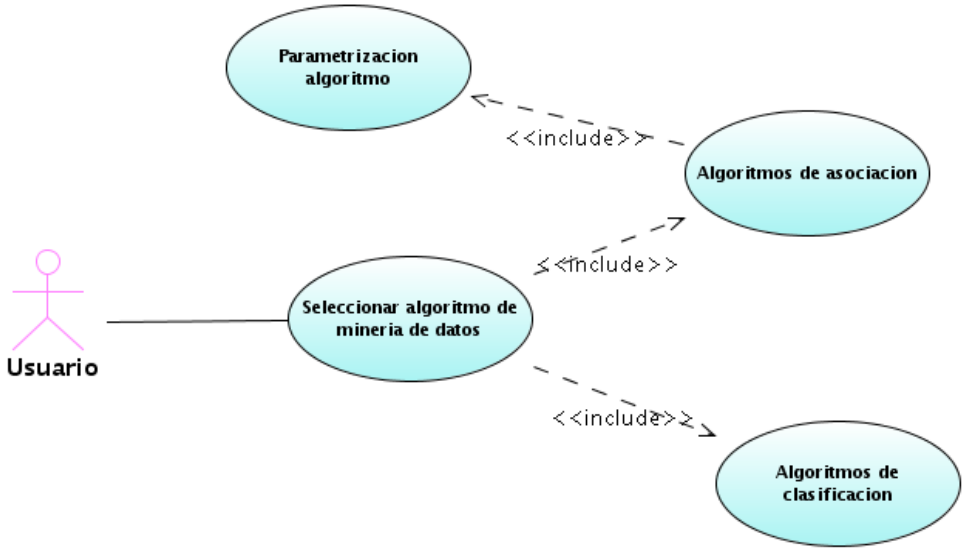
Módulo de Preprocesamiento

Figura 3.5: Preprocesamiento



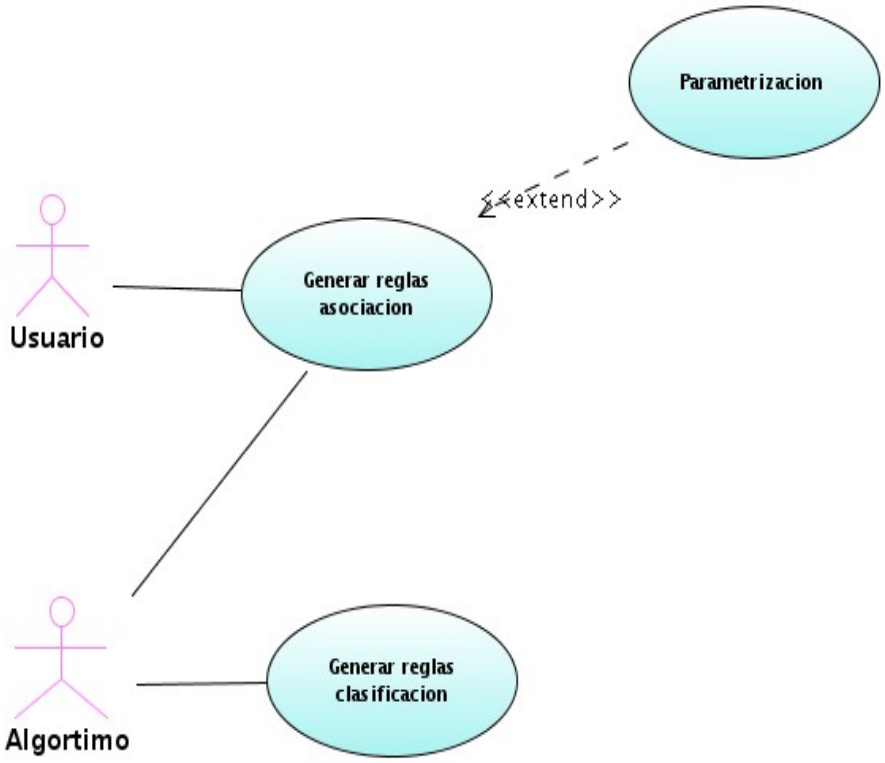
Módulo de Minería de Datos

Figura 3.6: Minería de Datos



Módulo de Reglas

Figura 3.7: Reglas



3.1.3 Diagramas de Secuencia

Clase Apriori

Figura 3.8: run

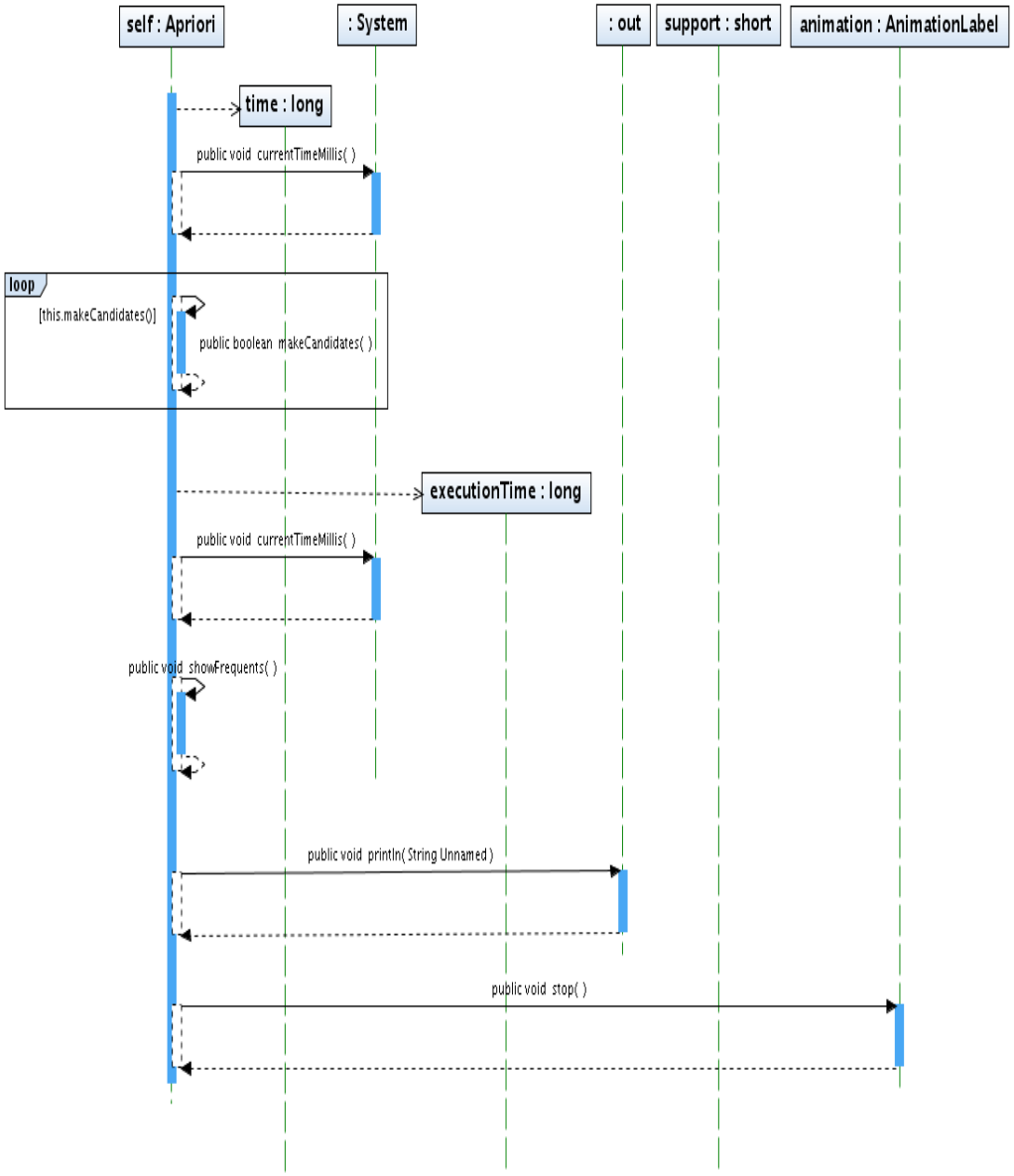


Figura 3.9: increaseSupport

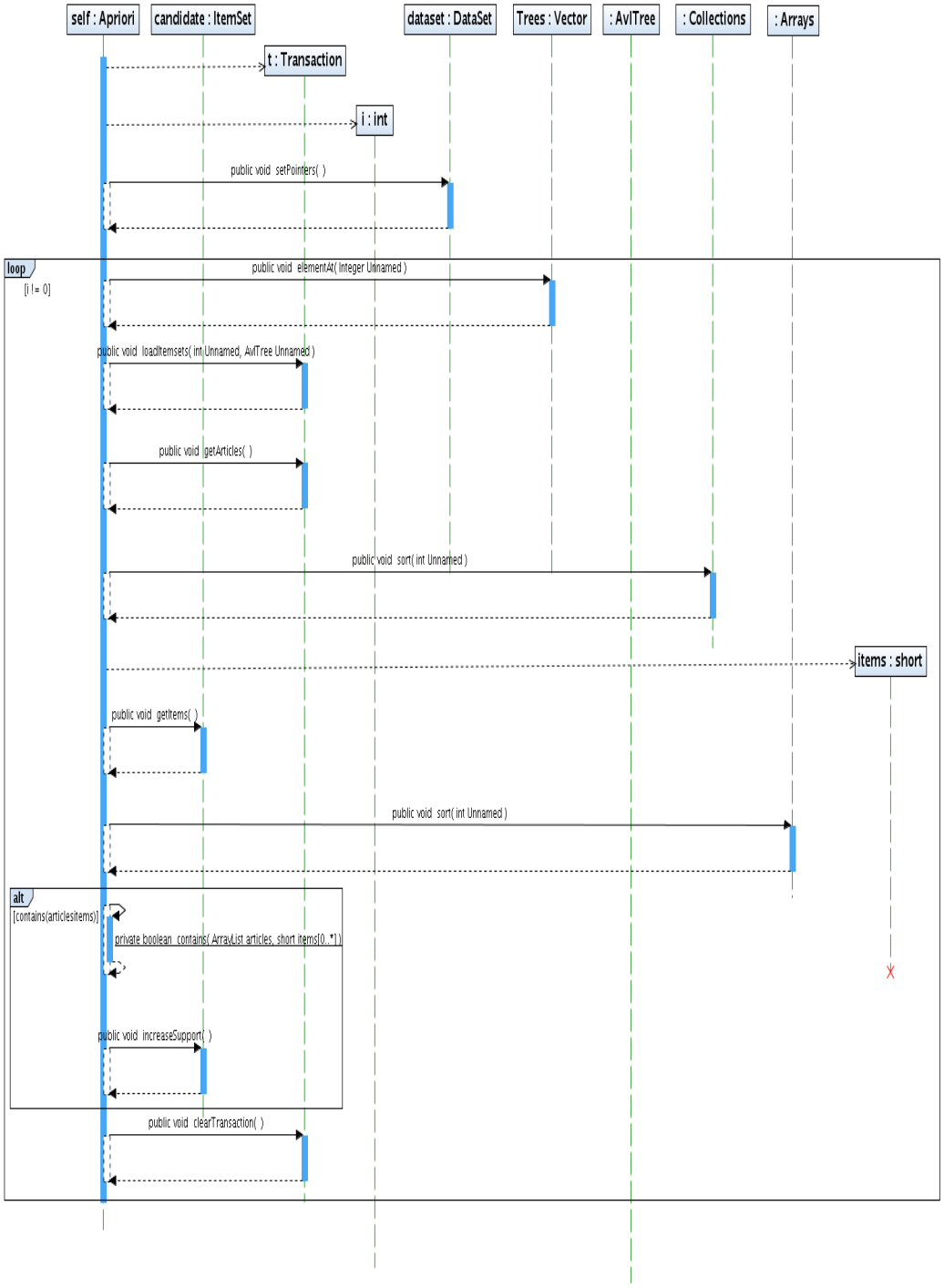


Figura 3.10: Combinations

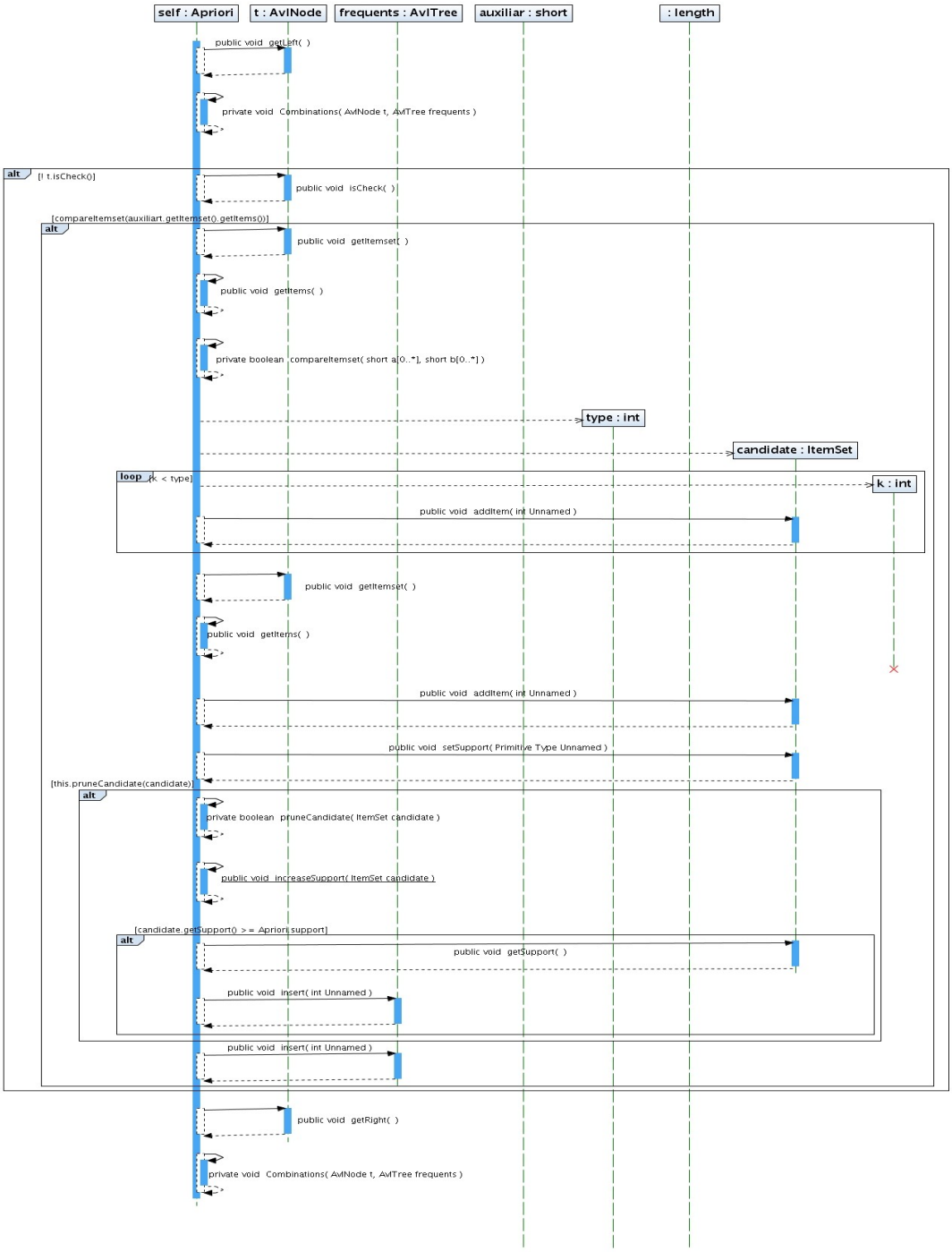


Figura 3.11: makeCandidates

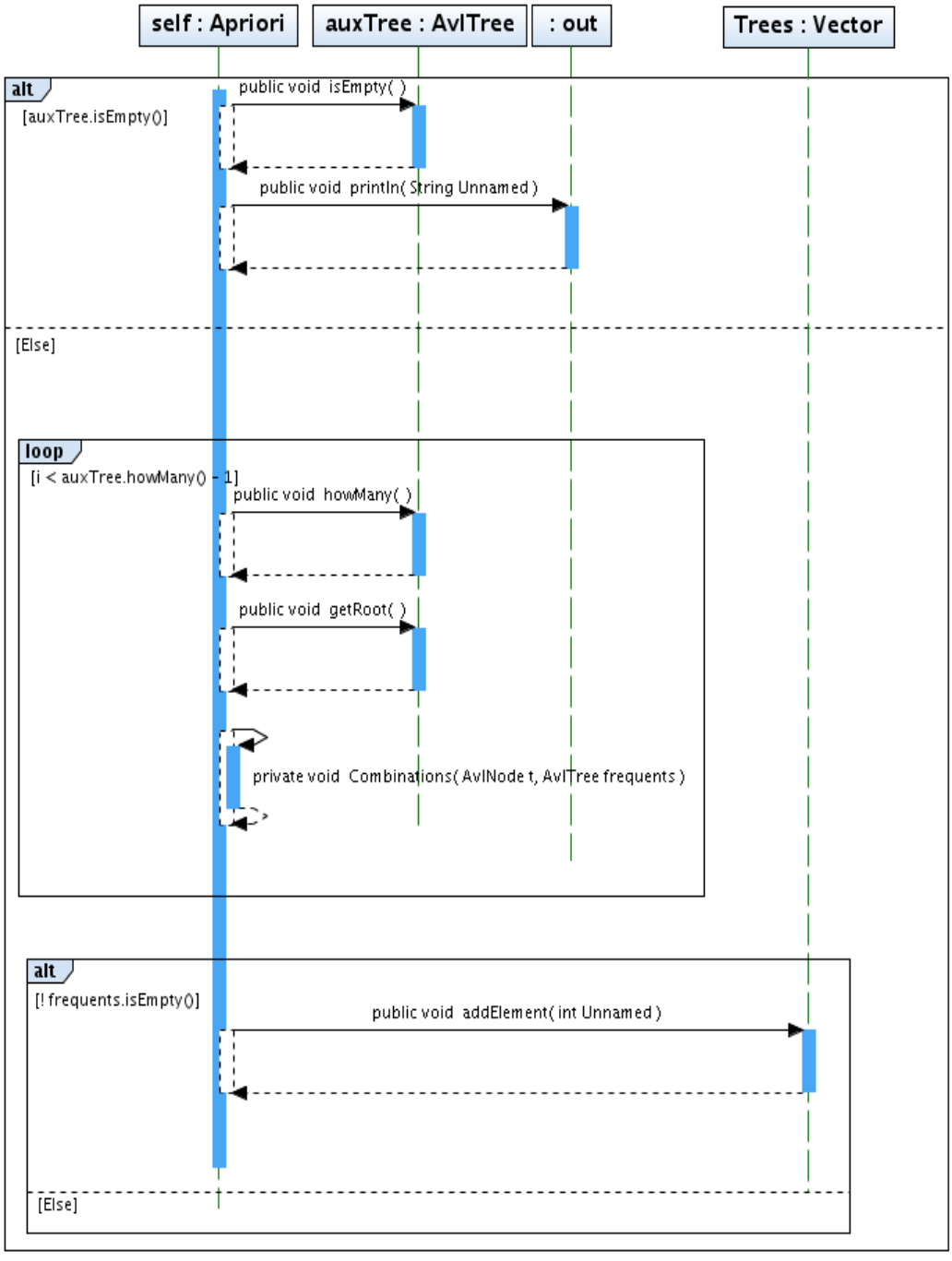
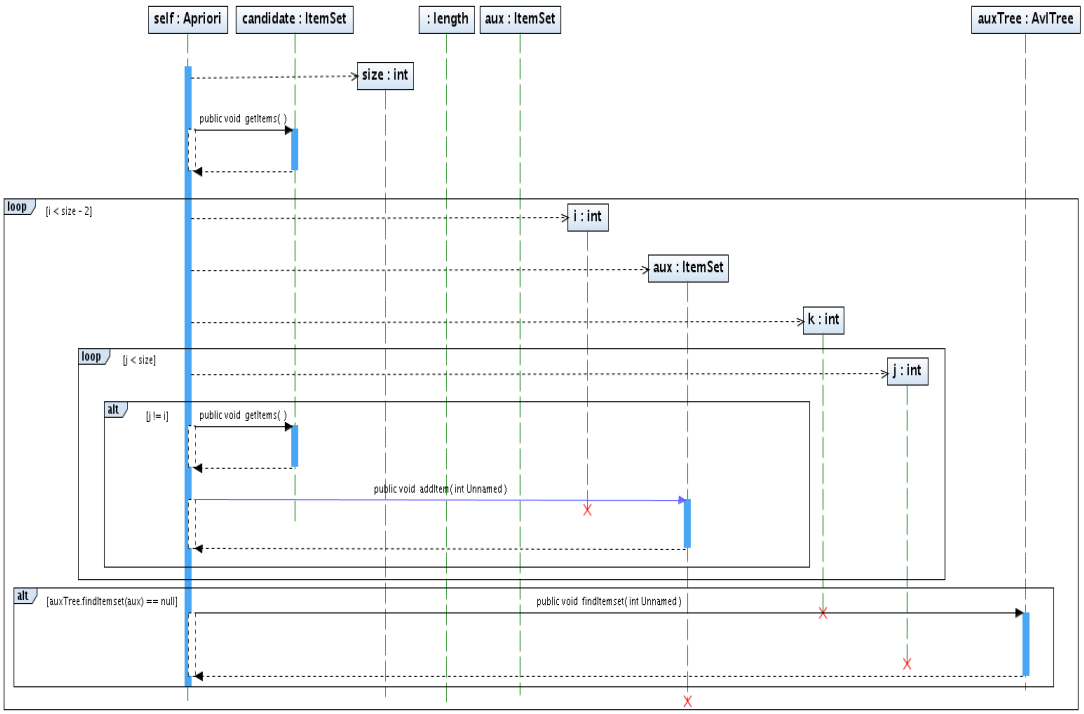


Figura 3.12: pruneCandidates



Clase EquipAsso

Figura 3.13: findInDataSet

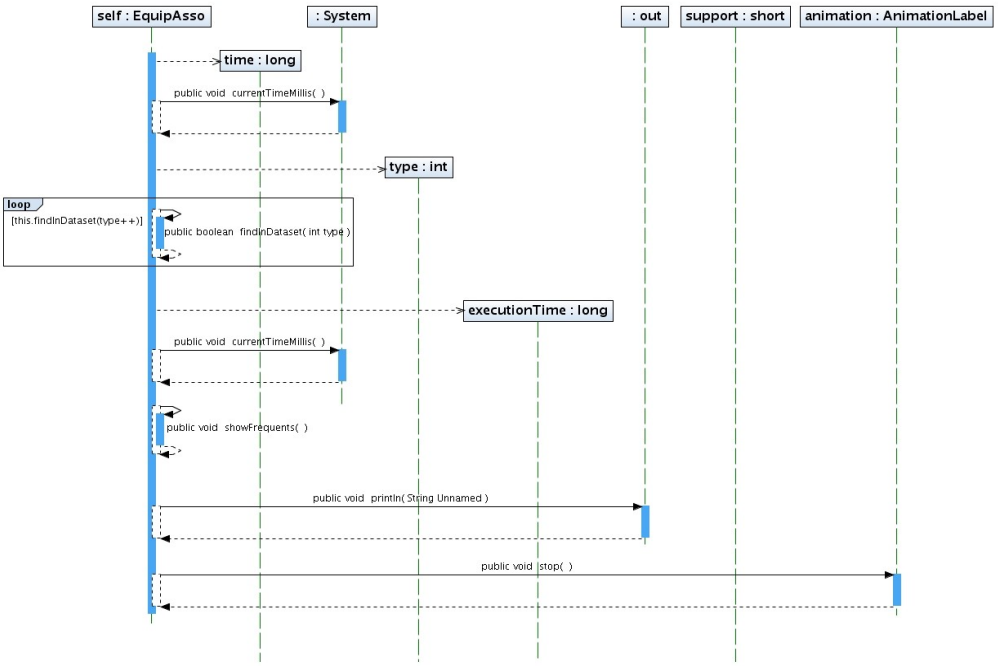


Figura 3.14: pruneCandidate-recursive

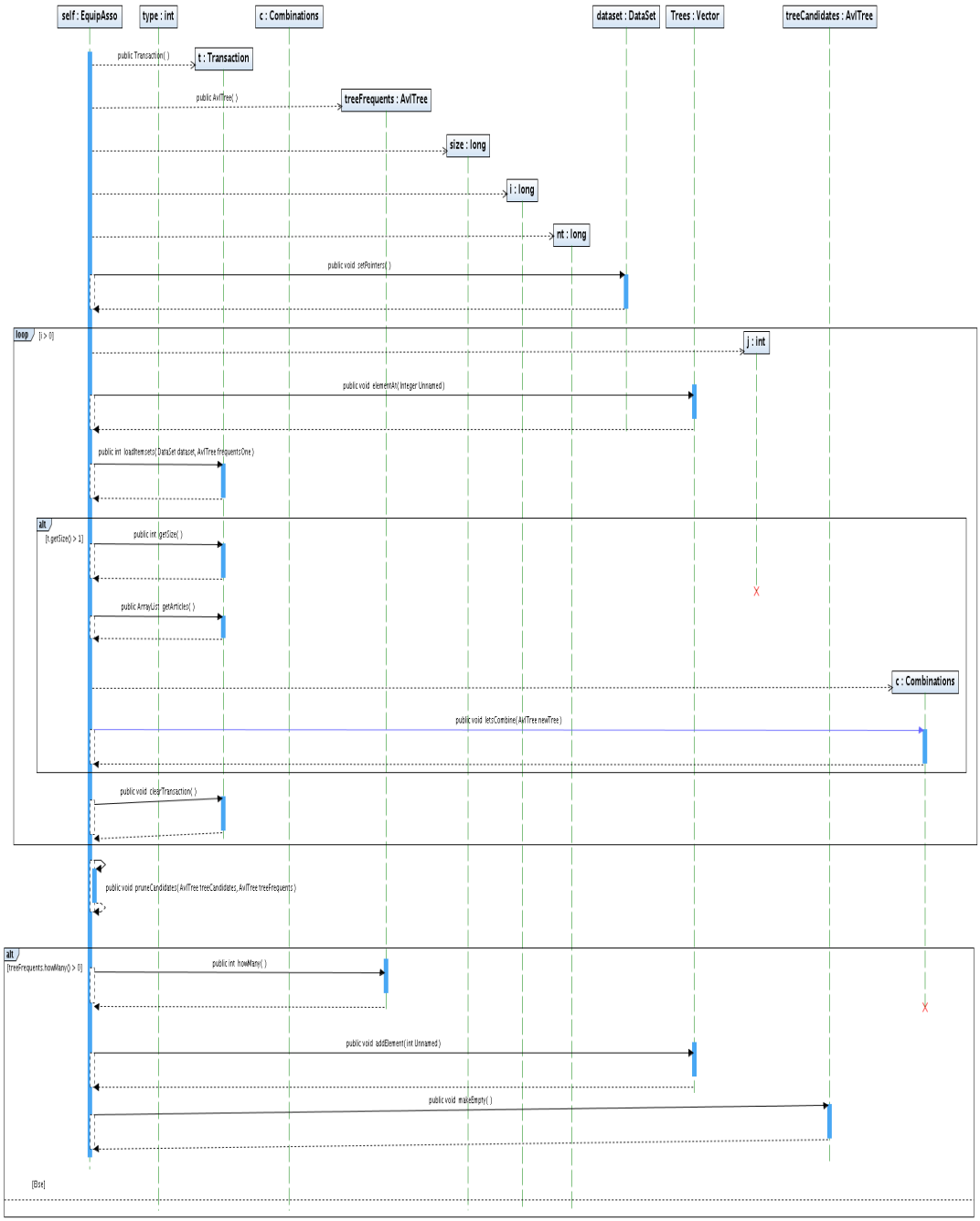


Figura 3.15: pruneCandidates

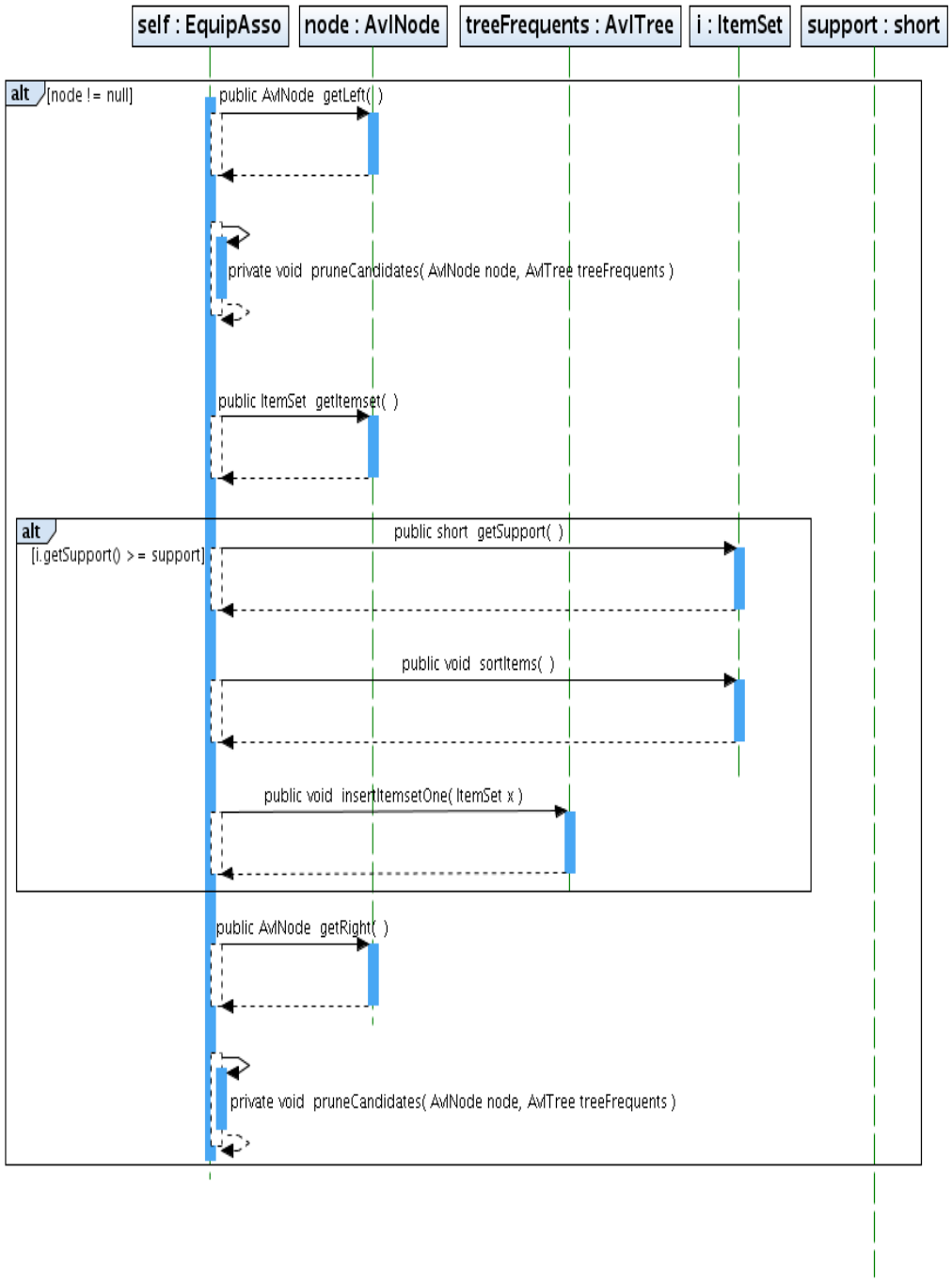
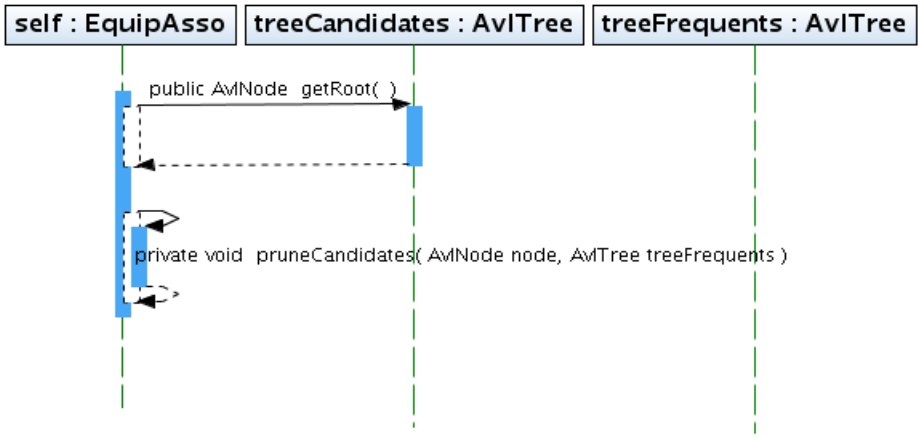


Figura 3.16: run



Clase ItemSet

Figura 3.17: addItems

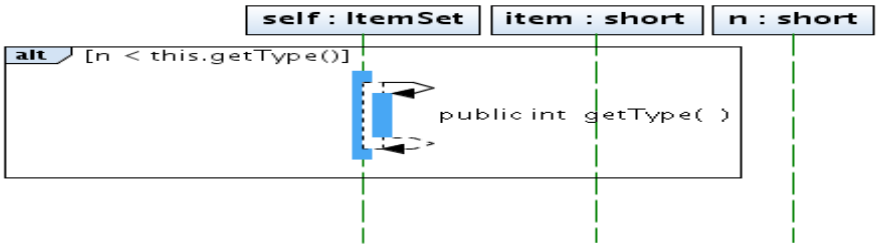


Figura 3.18: sortItems

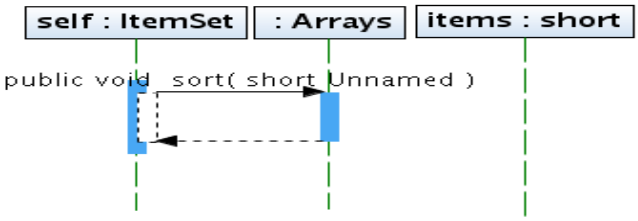
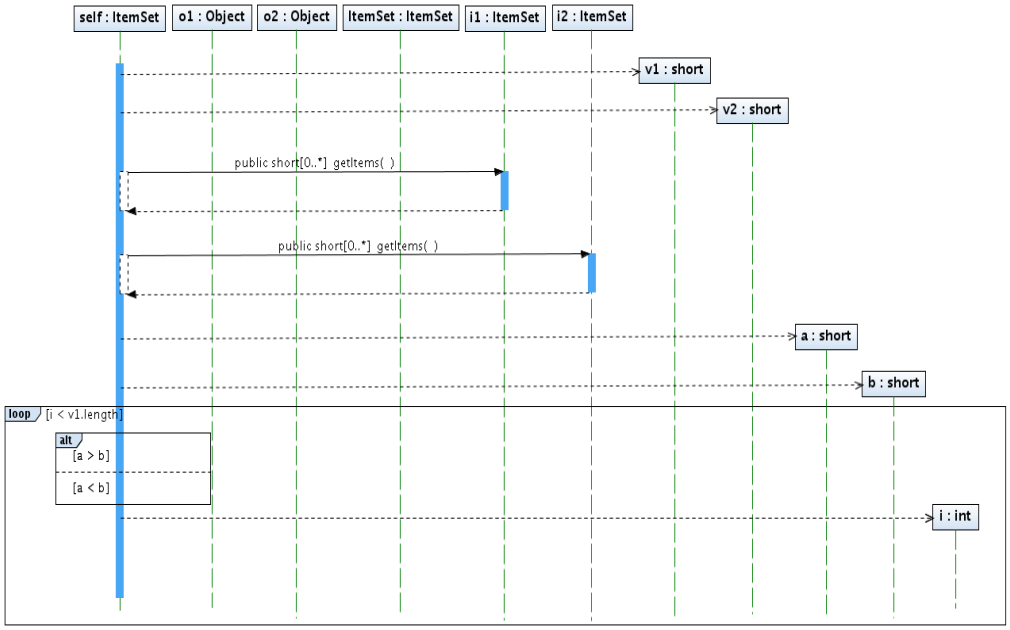


Figura 3.19: Compare



Clase DataSet

Figura 3.20: buildDictionary

Figura 3.21: buildNTree

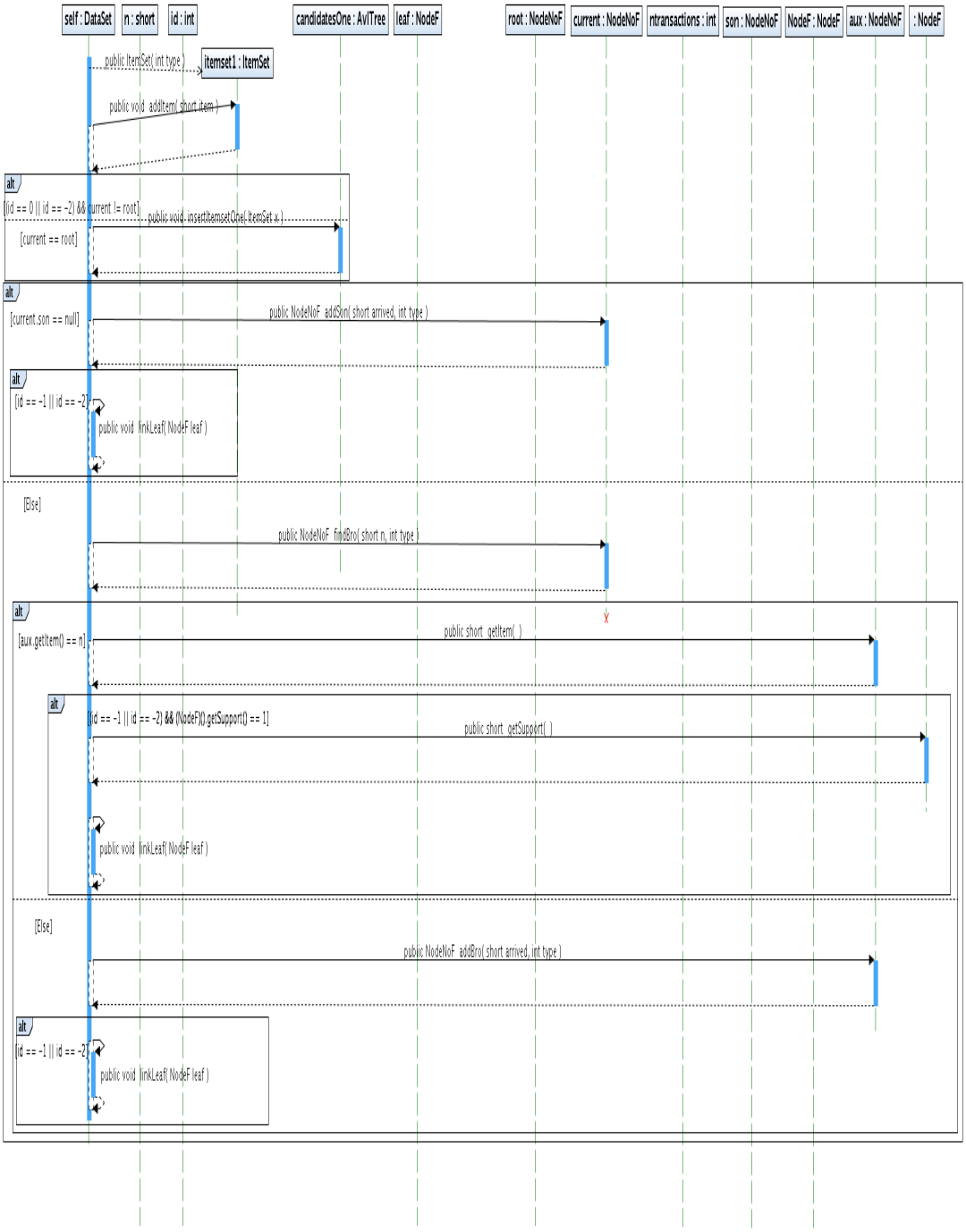


Figura 3.22: findAttrName

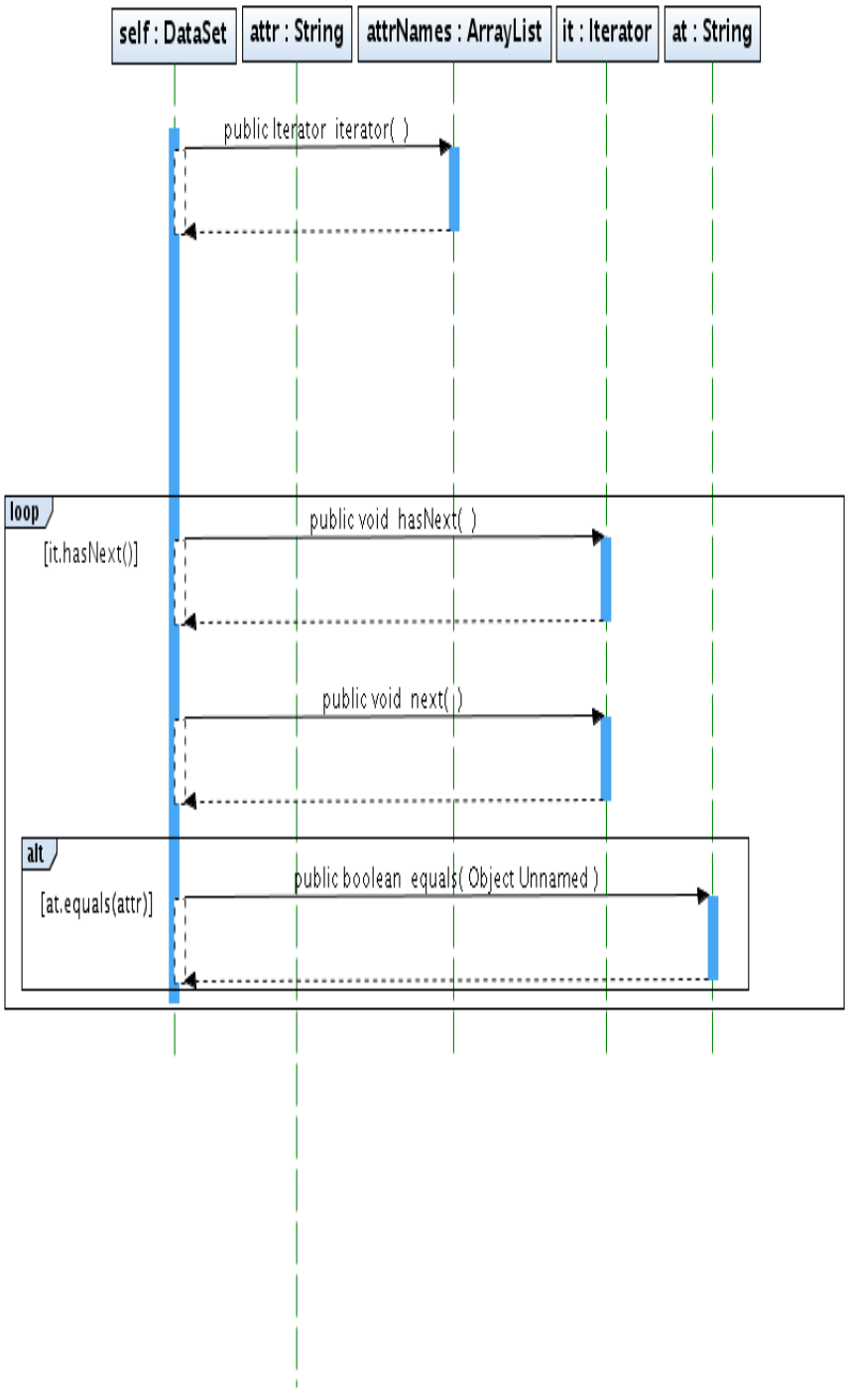


Figura 3.23: pruneCandidatesOne

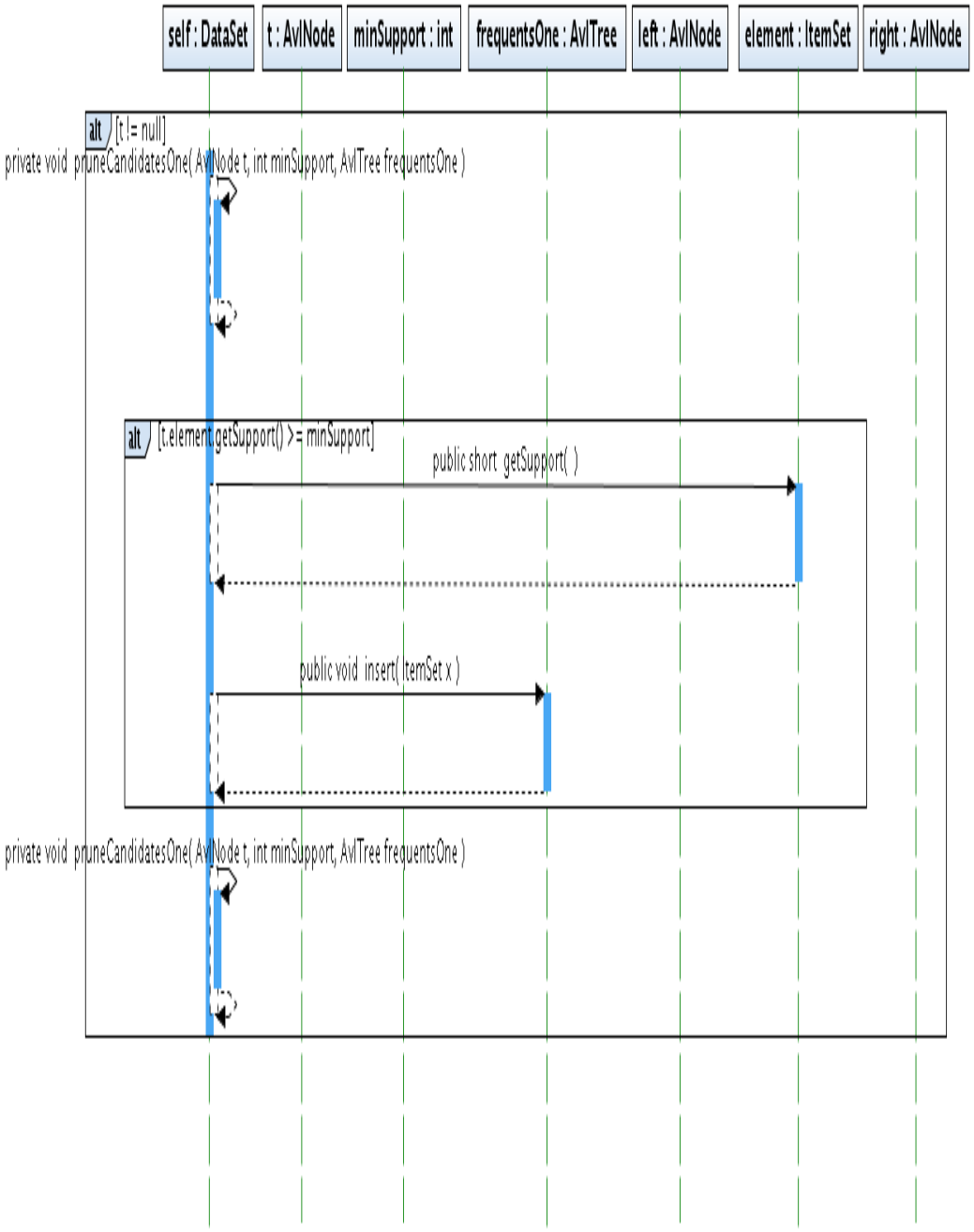
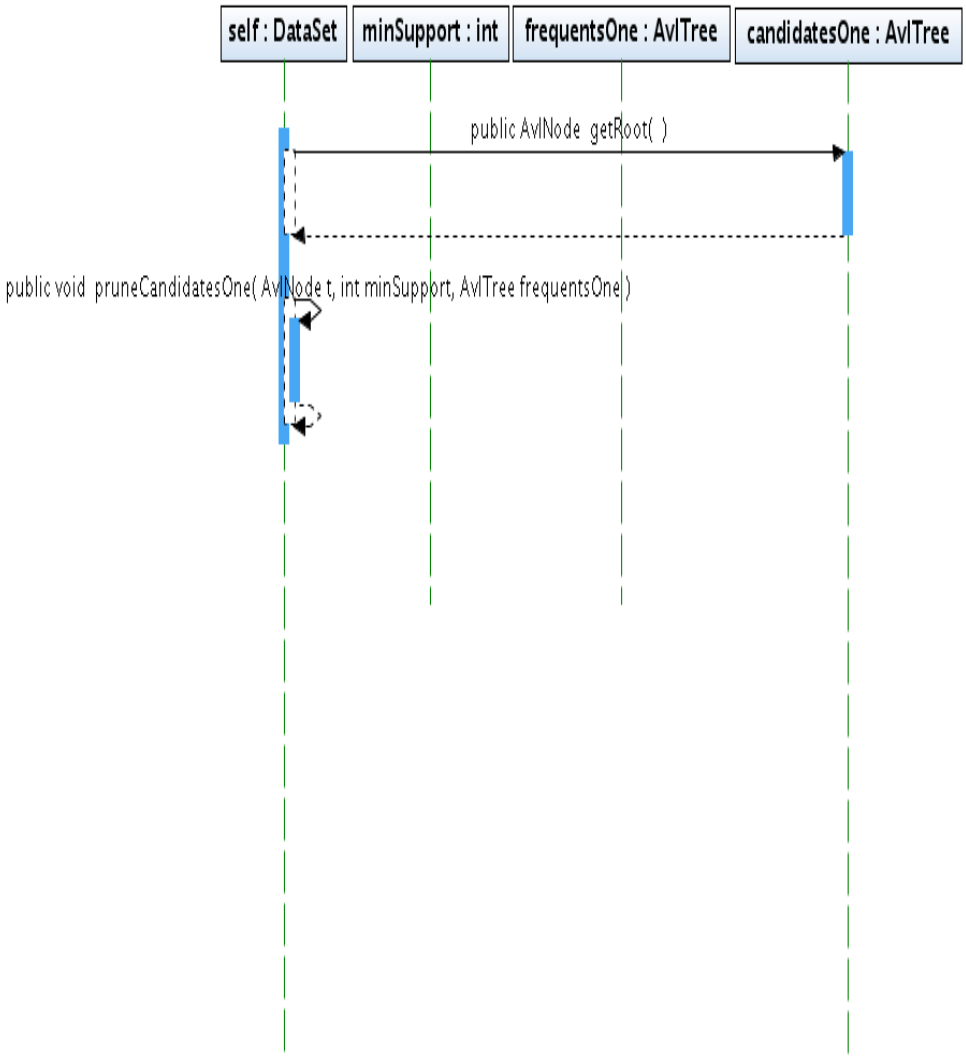


Figura 3.24: public-pruneCandidatesOne



Clase FPGrowth

Figura 3.25: buildFrequentsNodes

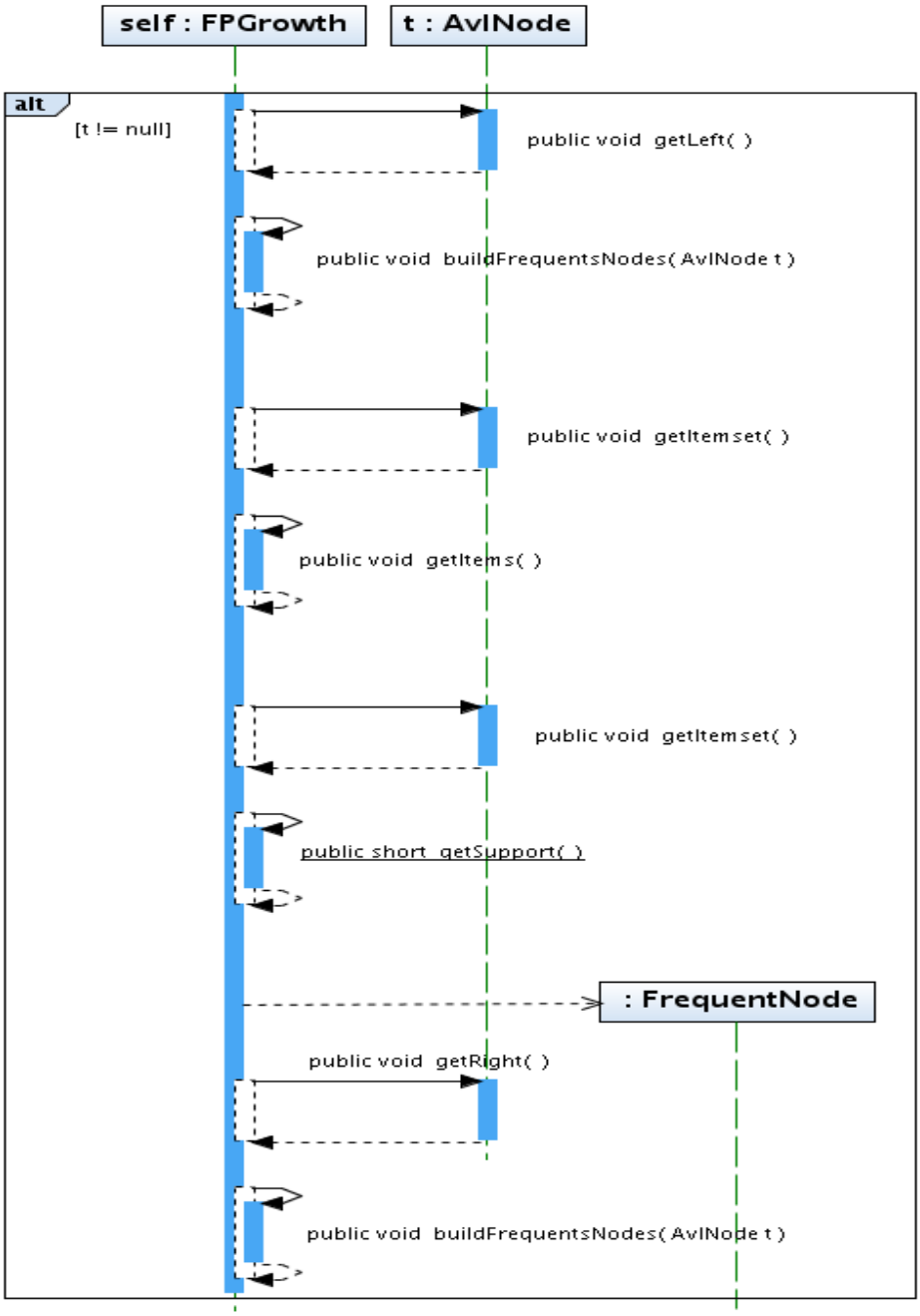


Figura 3.26: findNode

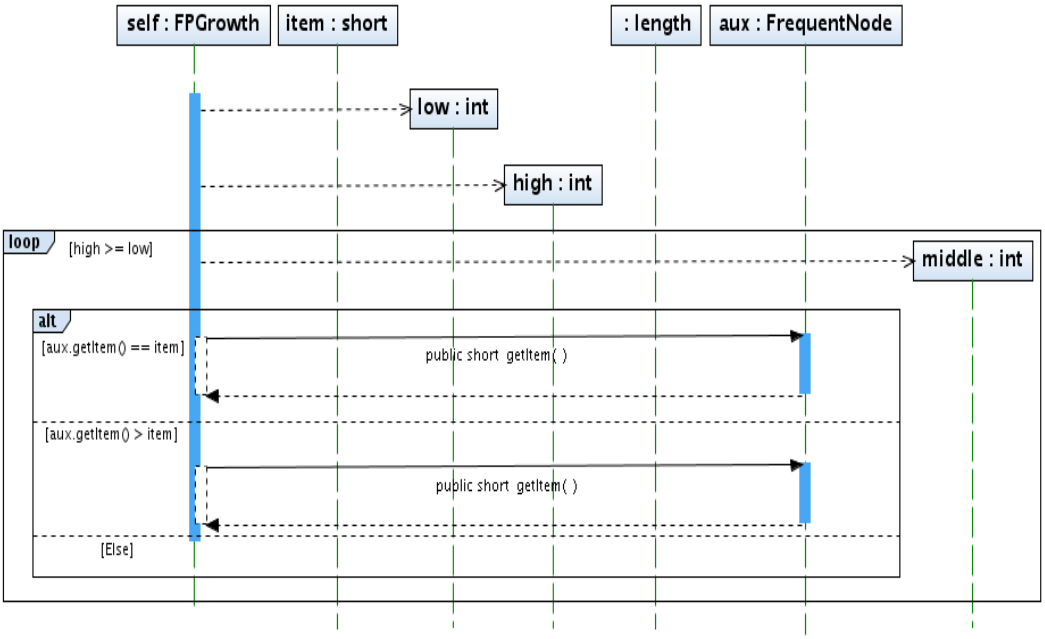
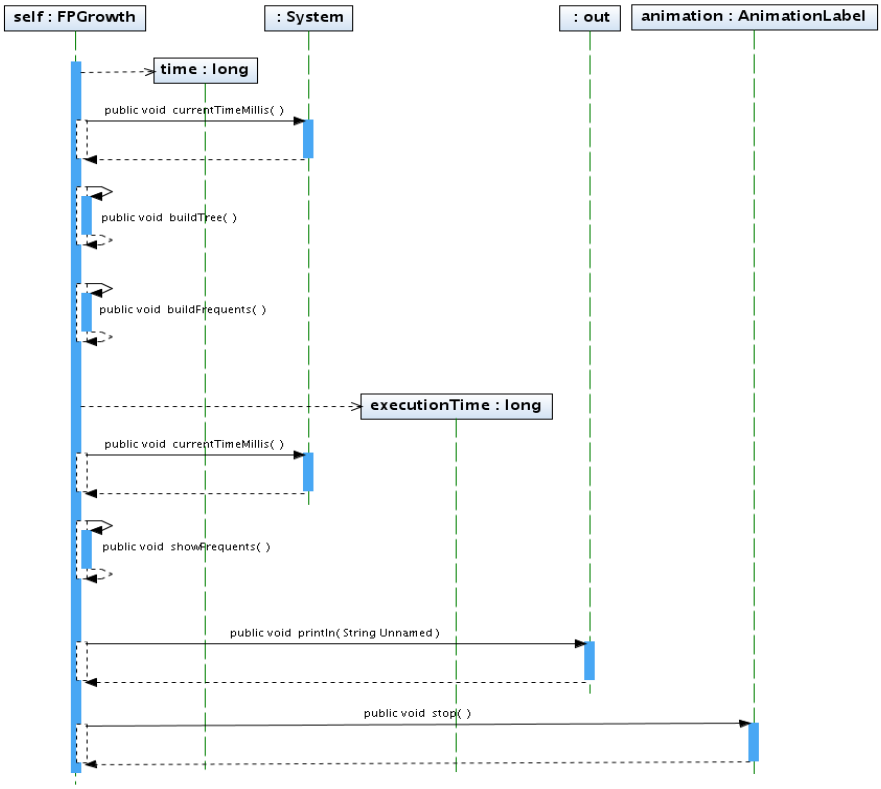


Figura 3.27: run



Clase AvlTree

Figura 3.28: compareItemSet

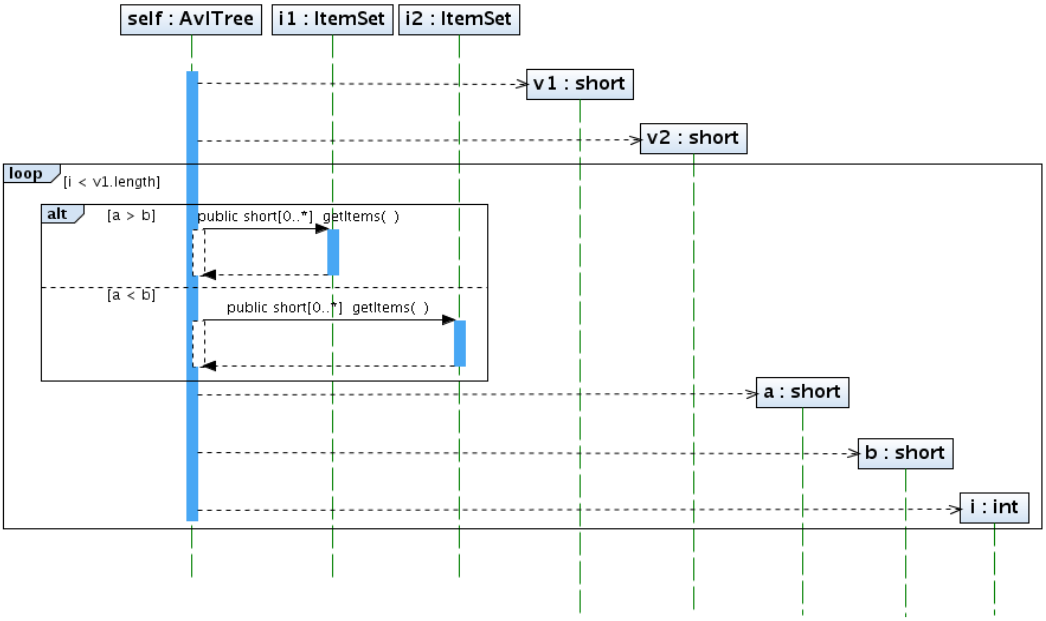


Figura 3.29: find

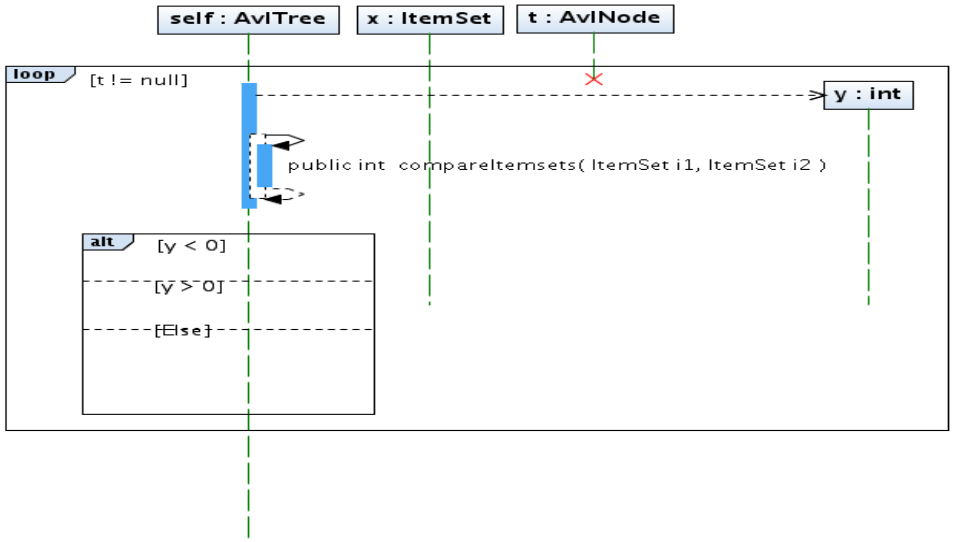


Figura 3.30: doubleWithRightChild

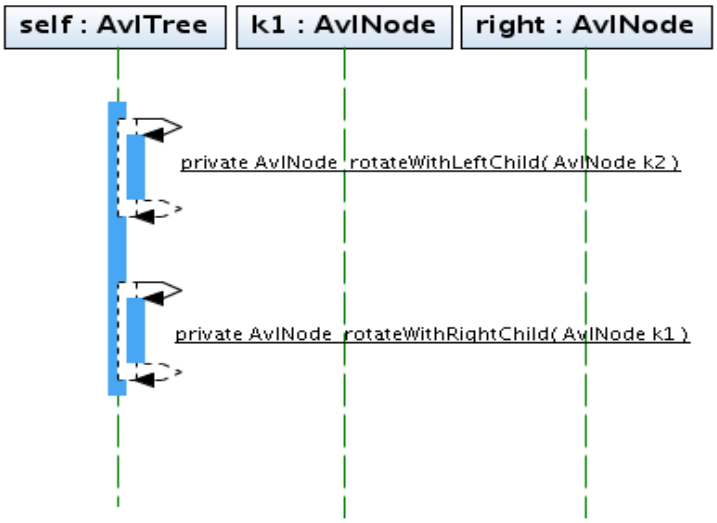
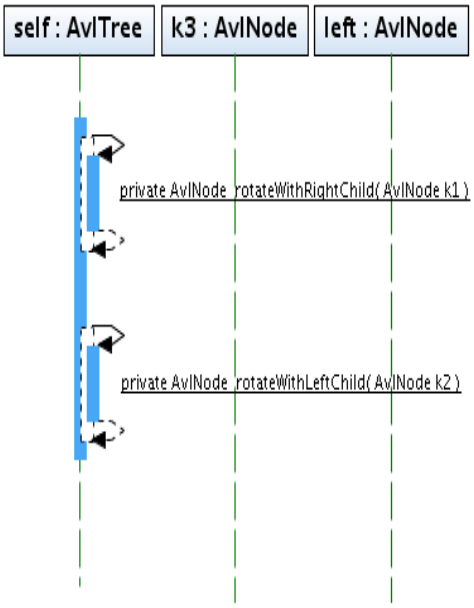


Figura 3.31: doubleWithLeftChild



Clase Transaction

Figura 3.32: loadItemset

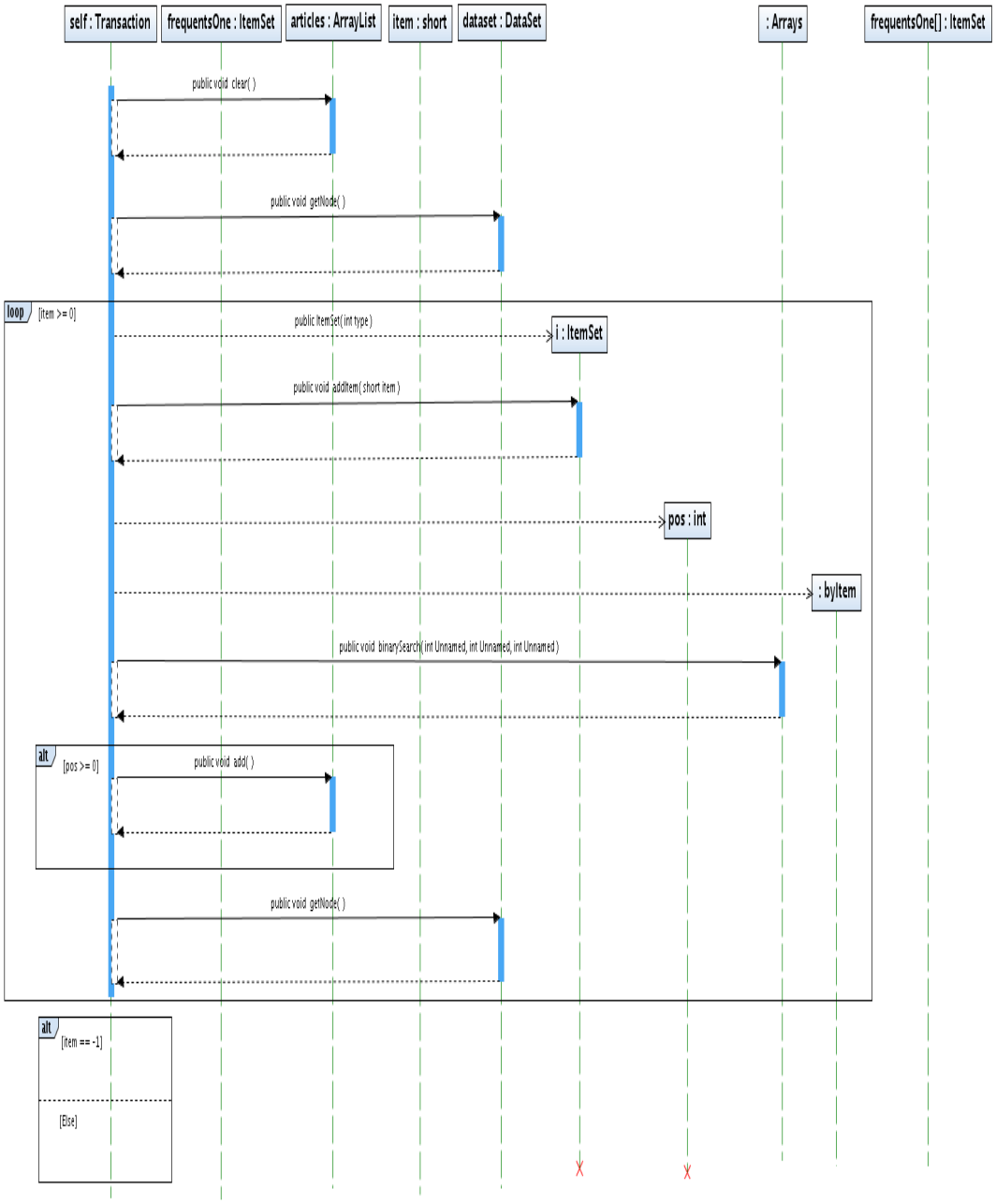


Figura 3.33: sortByItem

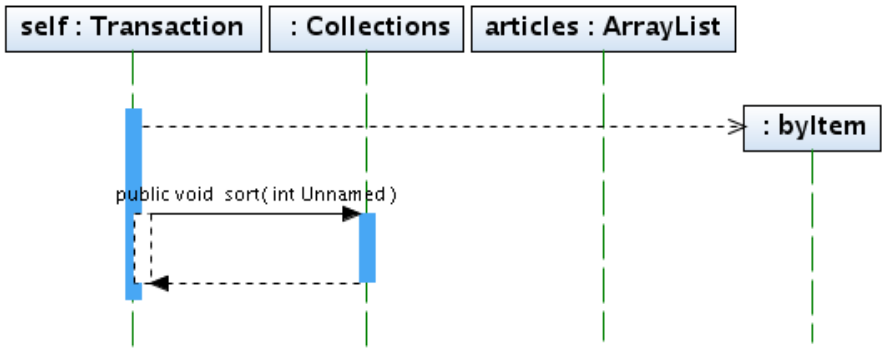
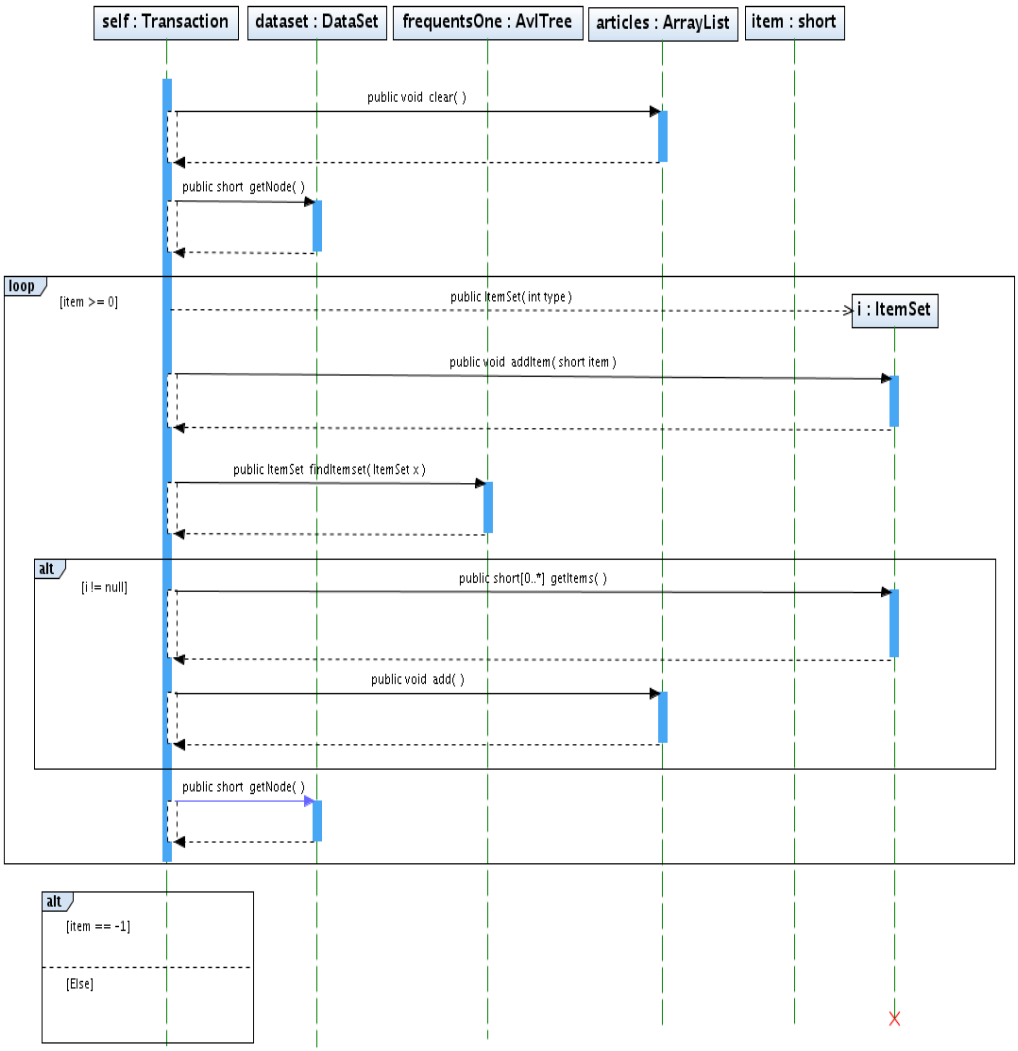


Figura 3.34: public-loadItemset



Clase NodeF

Figura 3.35: getBranch

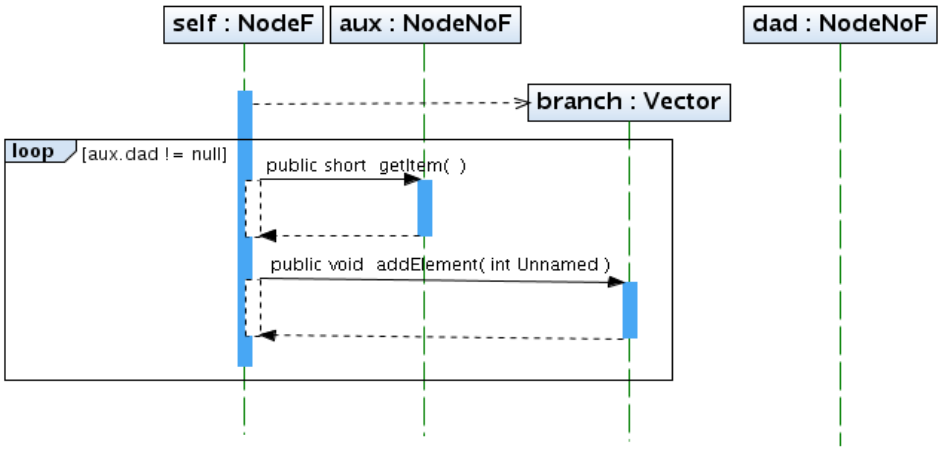
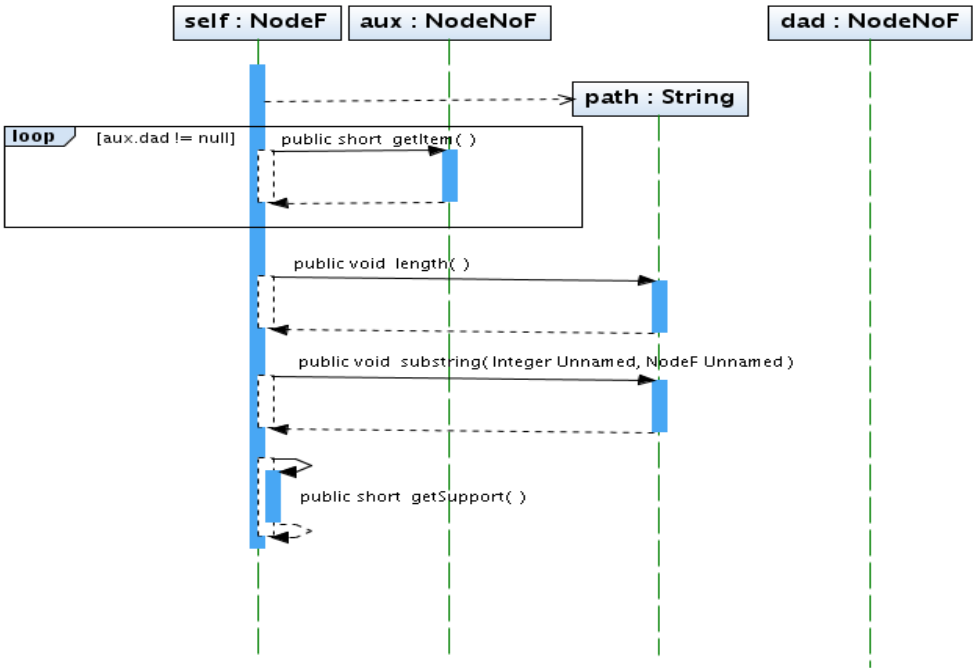


Figura 3.36: getPath



Clase NodeNoF

Figura 3.37: getChild

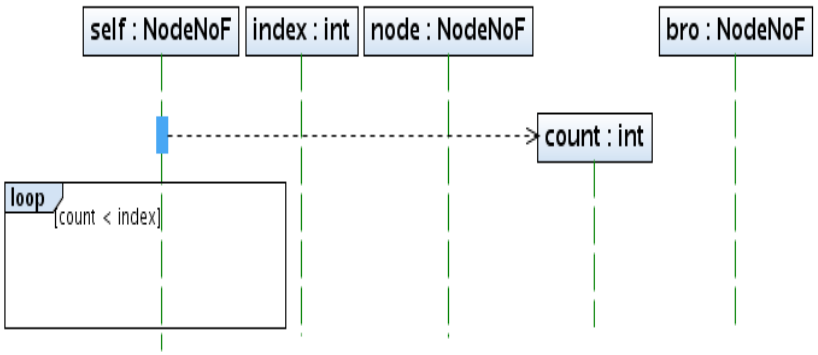


Figura 3.38: getIndexOfChild

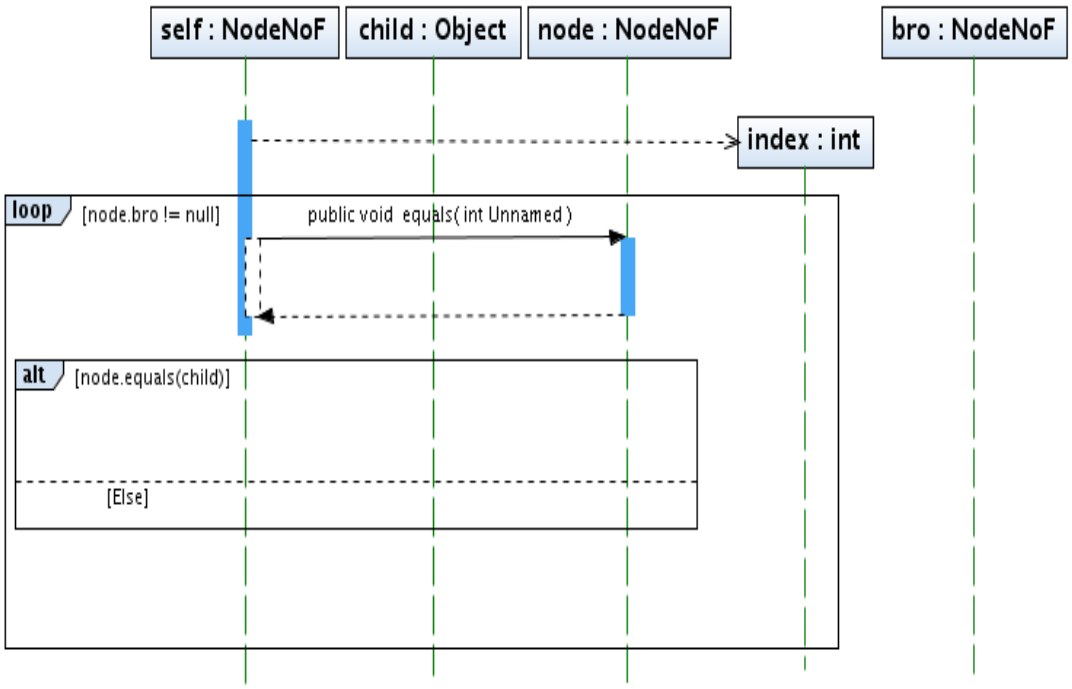
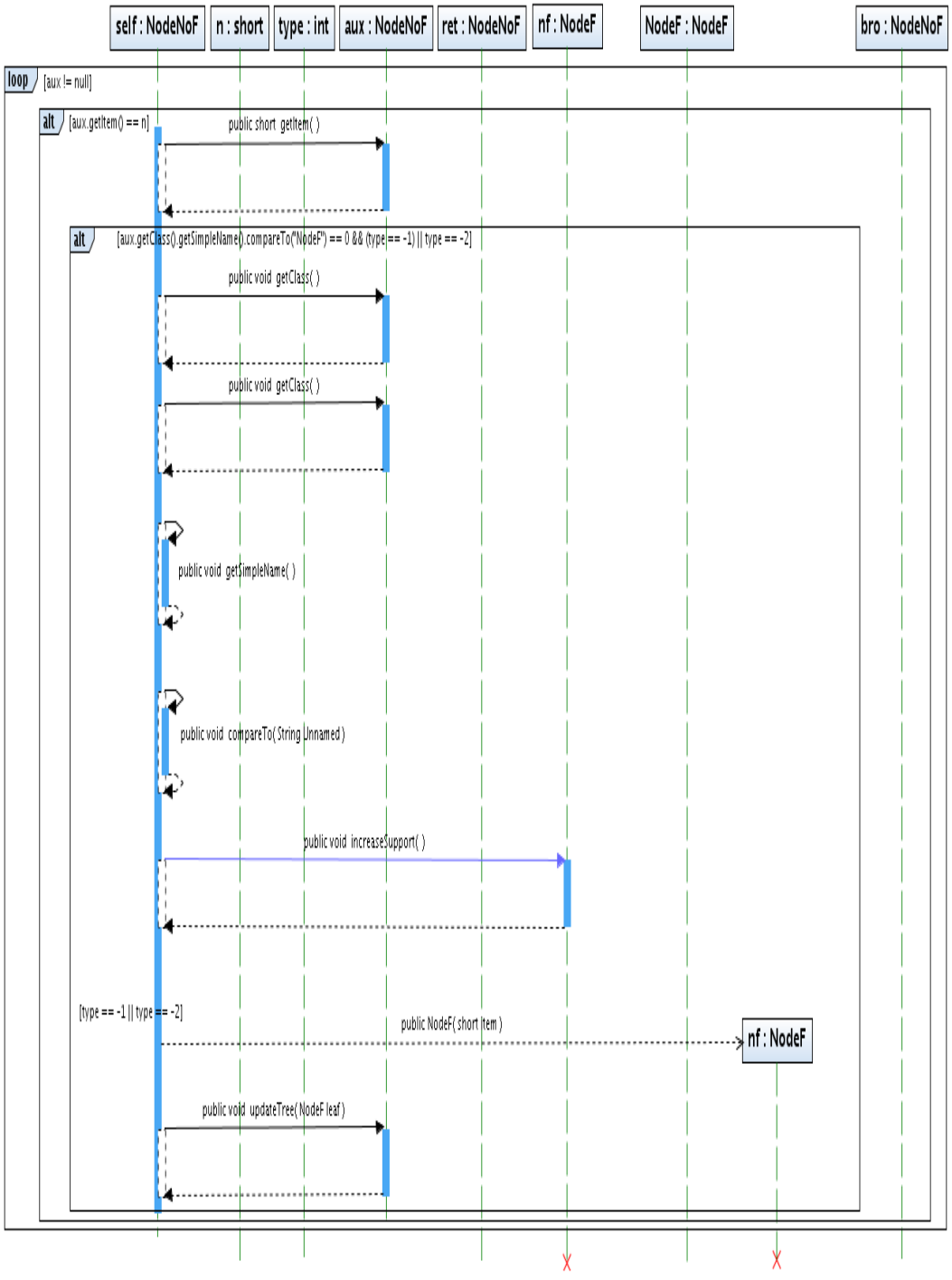
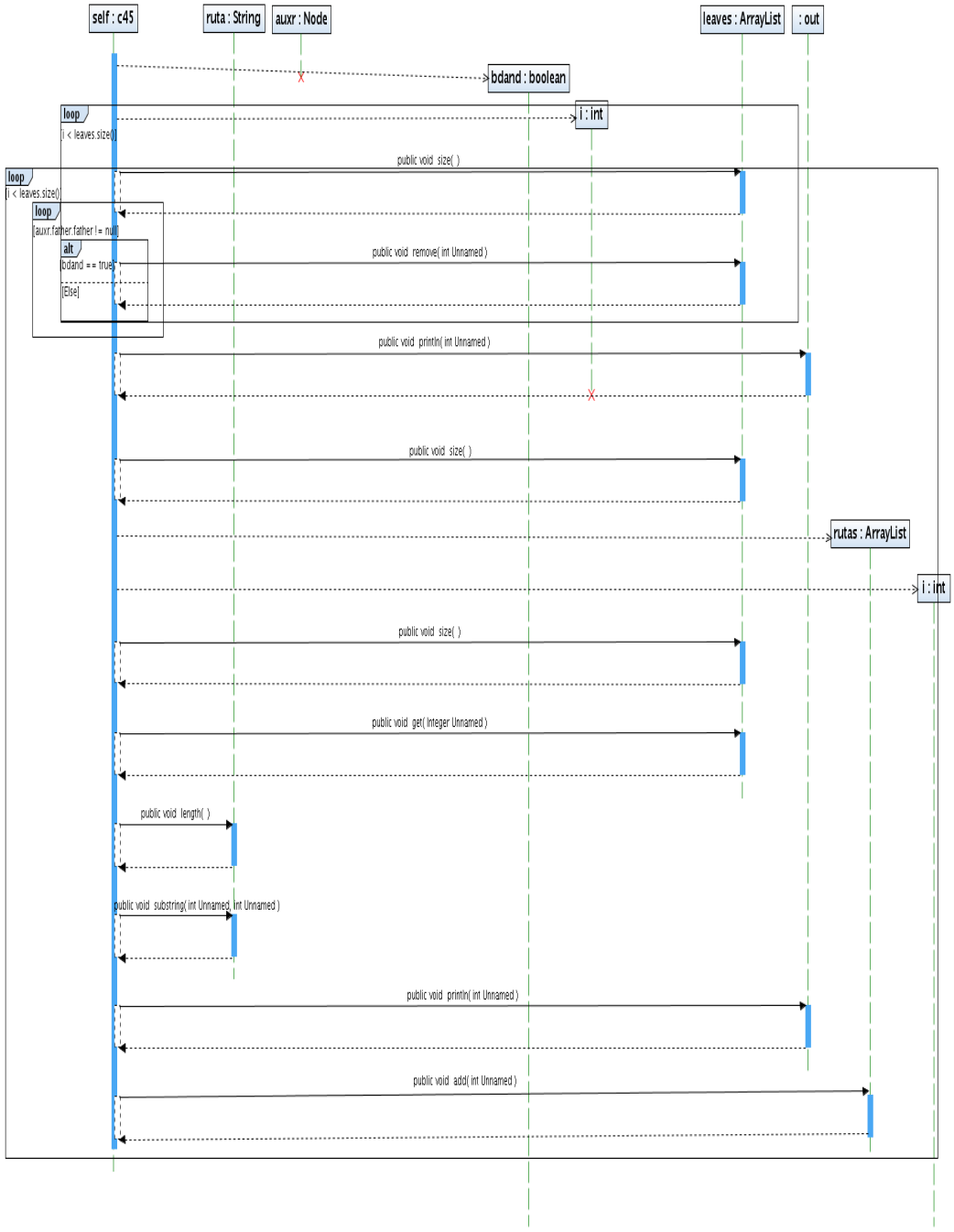


Figura 3.39: findBro



Clase C45

Figura 3.40: C45Rules



Clase myHasMap

Figura 3.41: addColumn

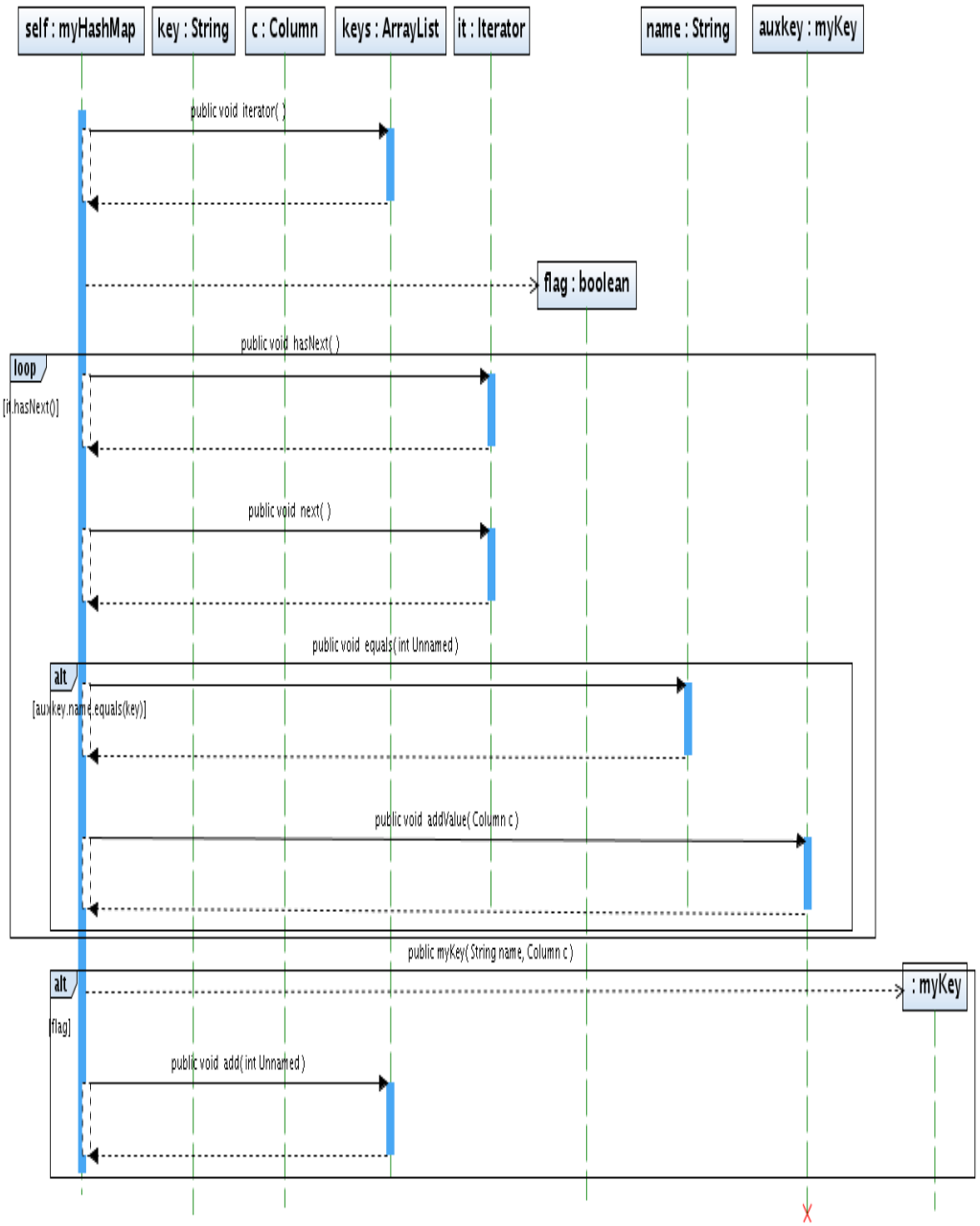
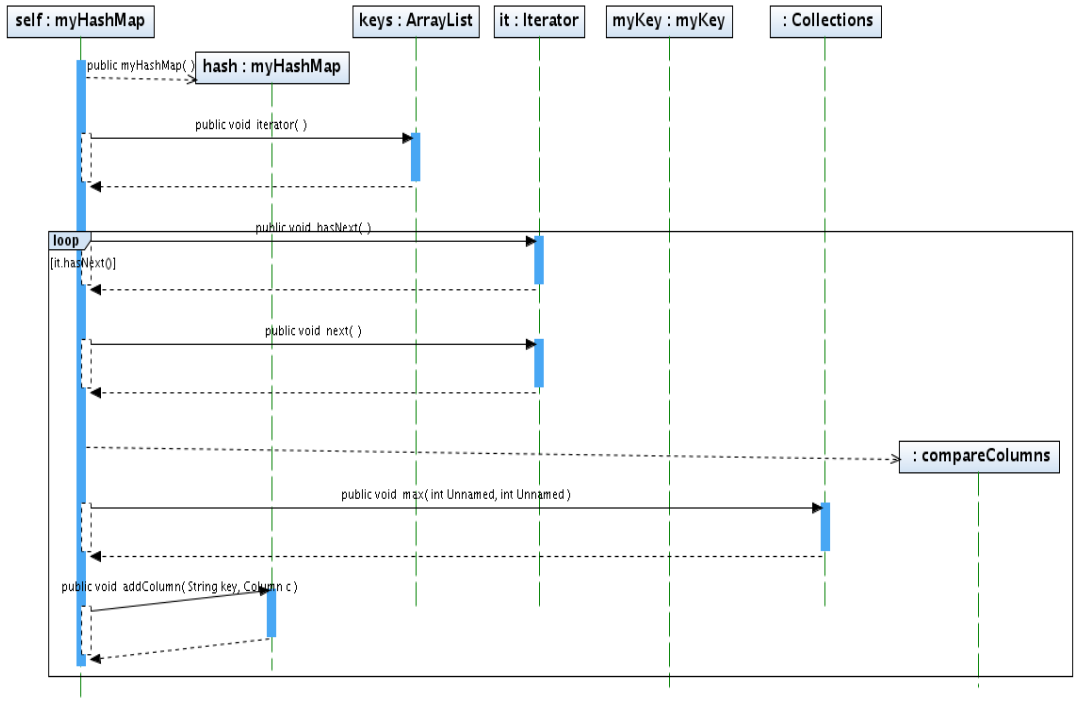


Figura 3.42: SearchColumn



Clase Route

Figura 3.43: avanceVariable

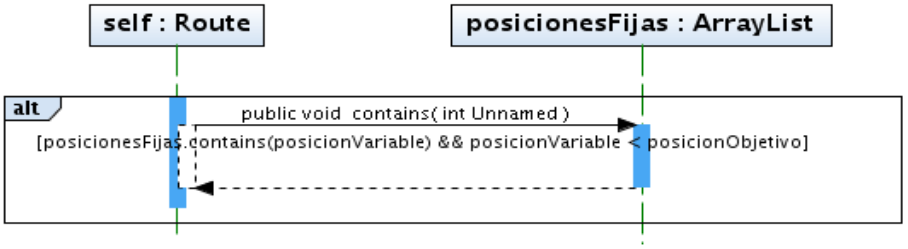


Figura 3.44: setPoscicionVariable

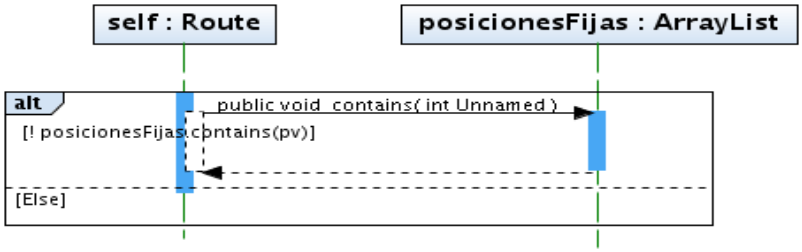
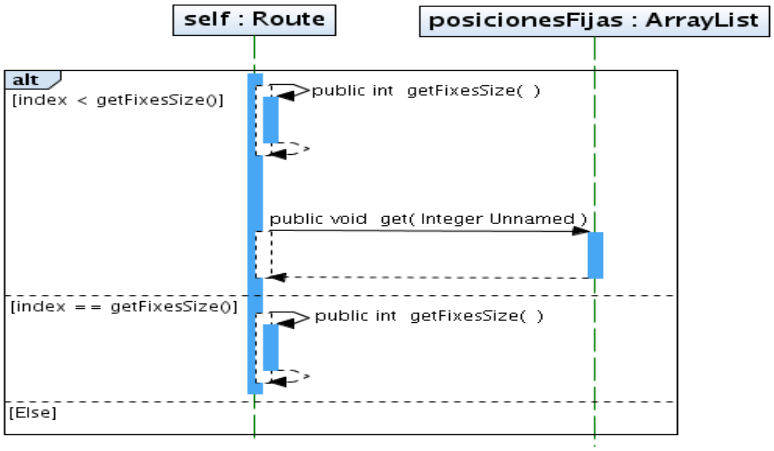


Figura 3.45: getIndex



Clase TreeCounter

Figura 3.46: firstGain

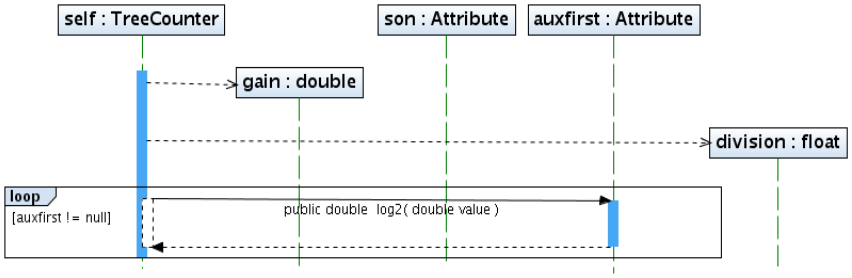
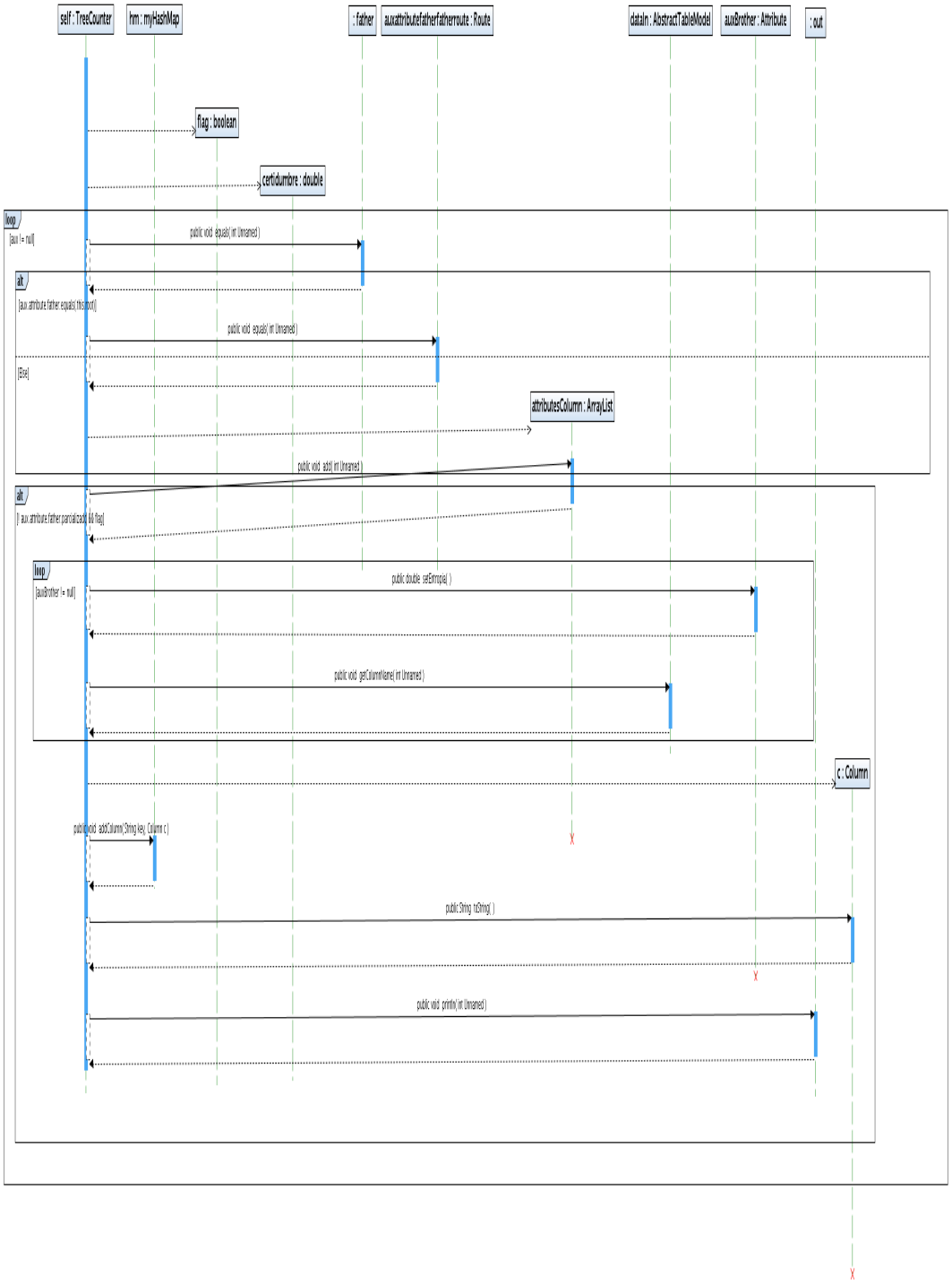


Figura 3.47: gain



Clase Attribute

Figura 3.48: log2

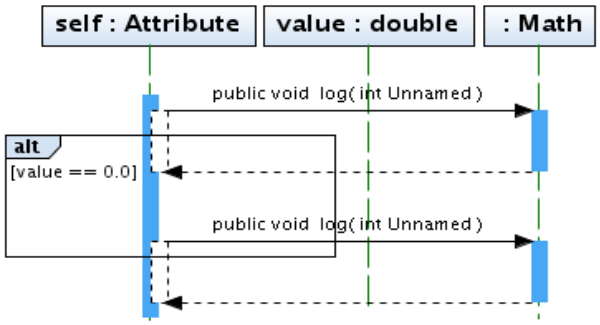
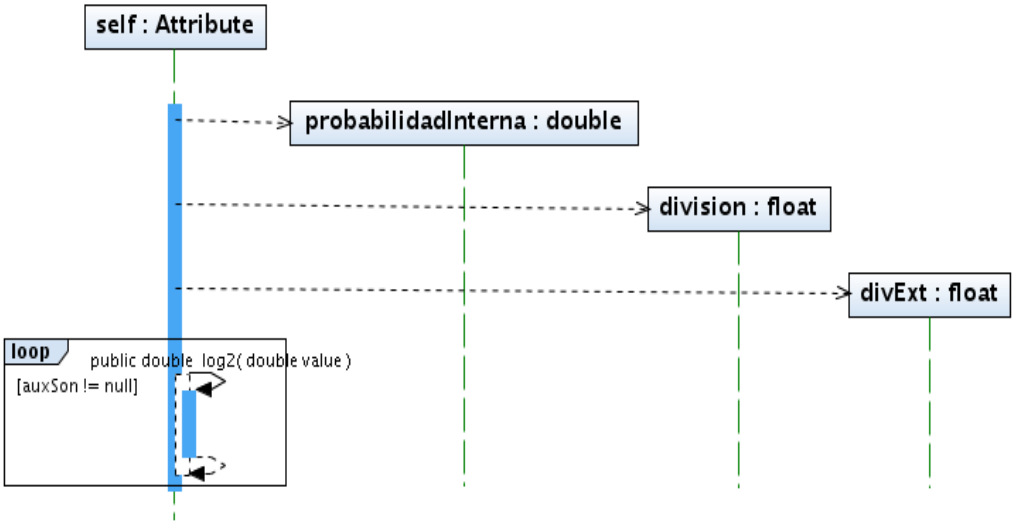


Figura 3.49: setEntropia



Clase Node

Figura 3.50: addSon

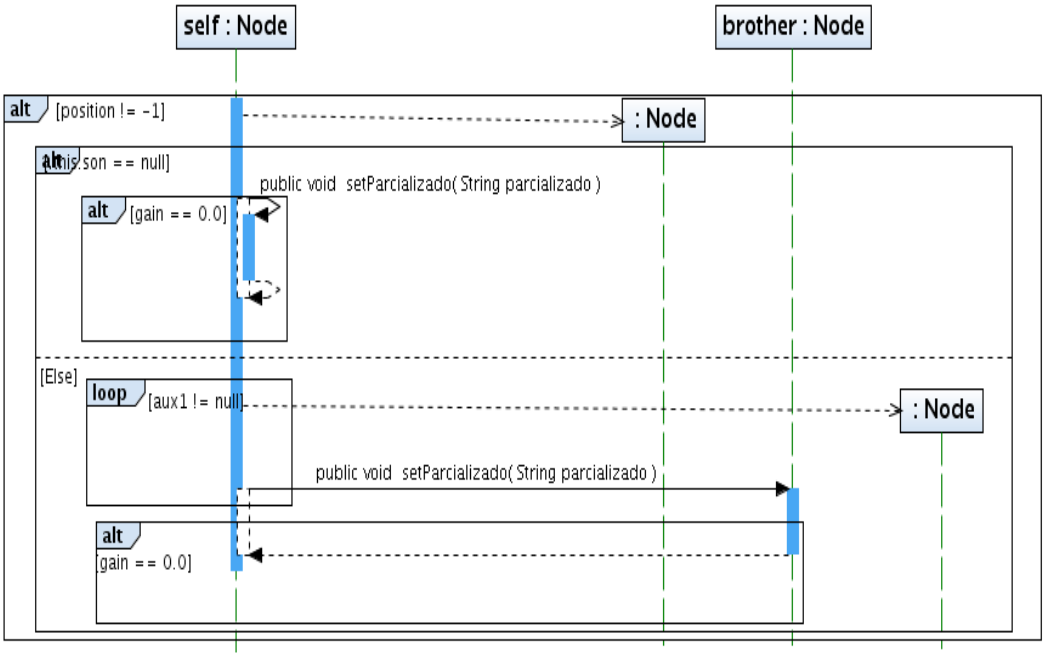
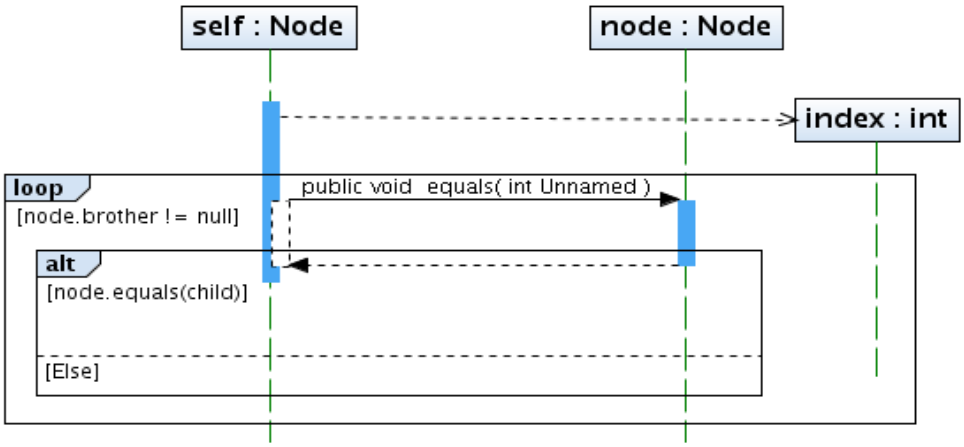


Figura 3.51: getIndexOfChild



3.2 Diseño

3.2.1 Diagramas de Colaboración

Clase Apriori

Figura 3.52: Combinations

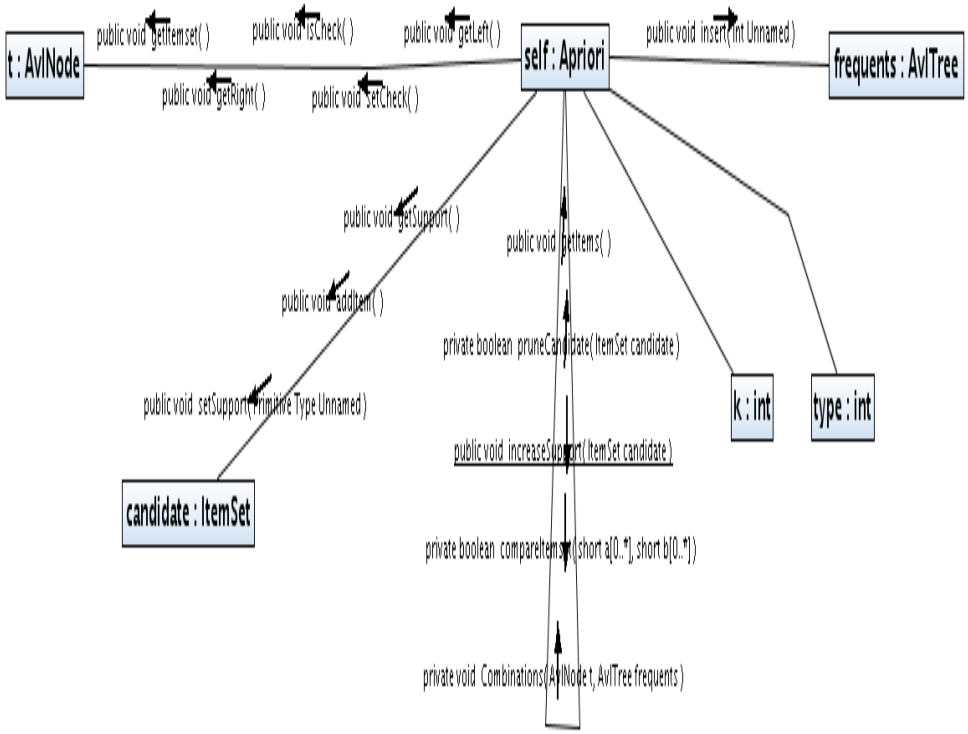


Figura 3.53: IncreaseSuport

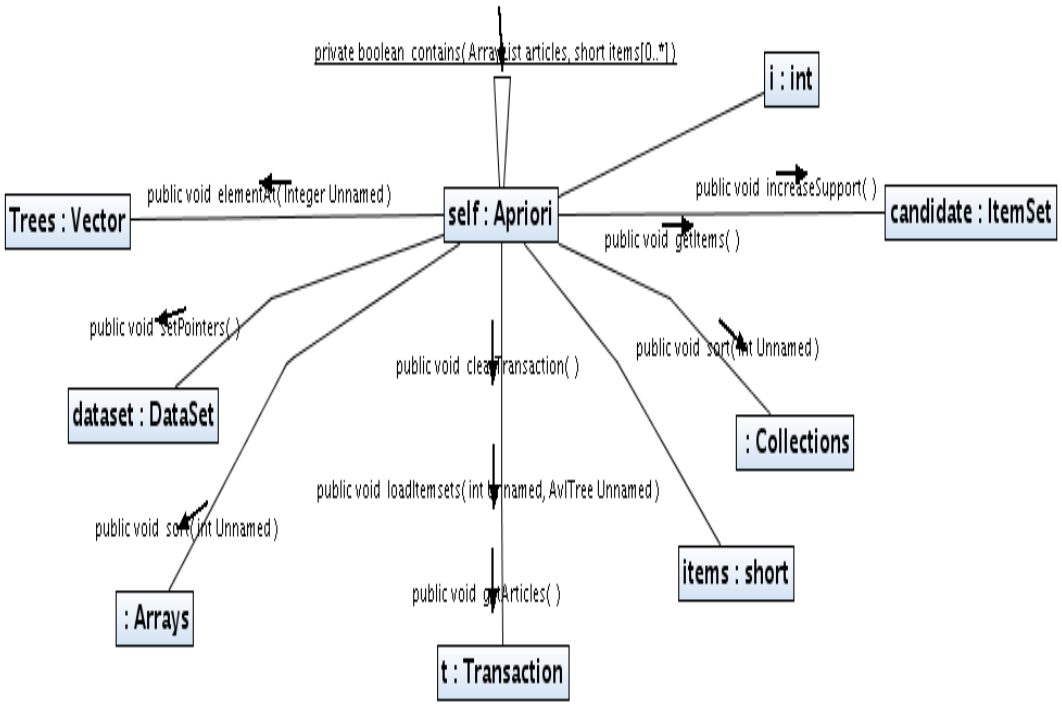


Figura 3.54: main

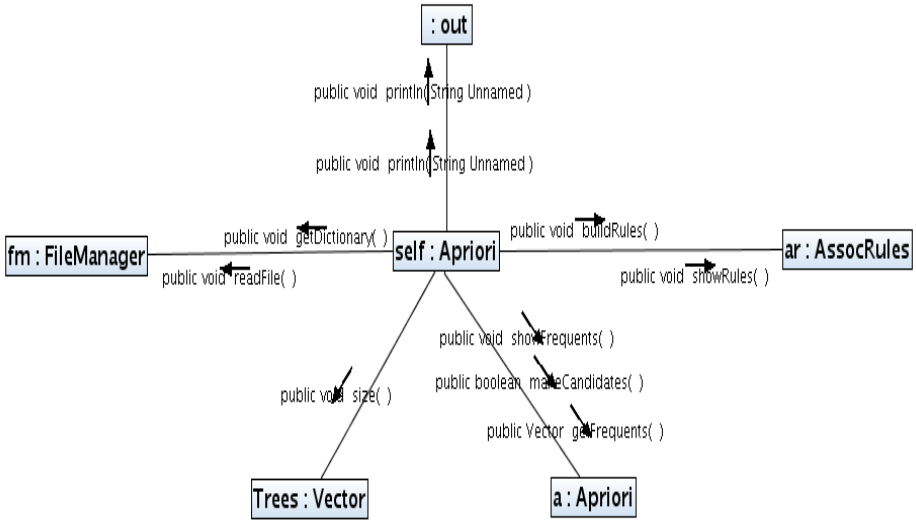


Figura 3.55: makeCandidates

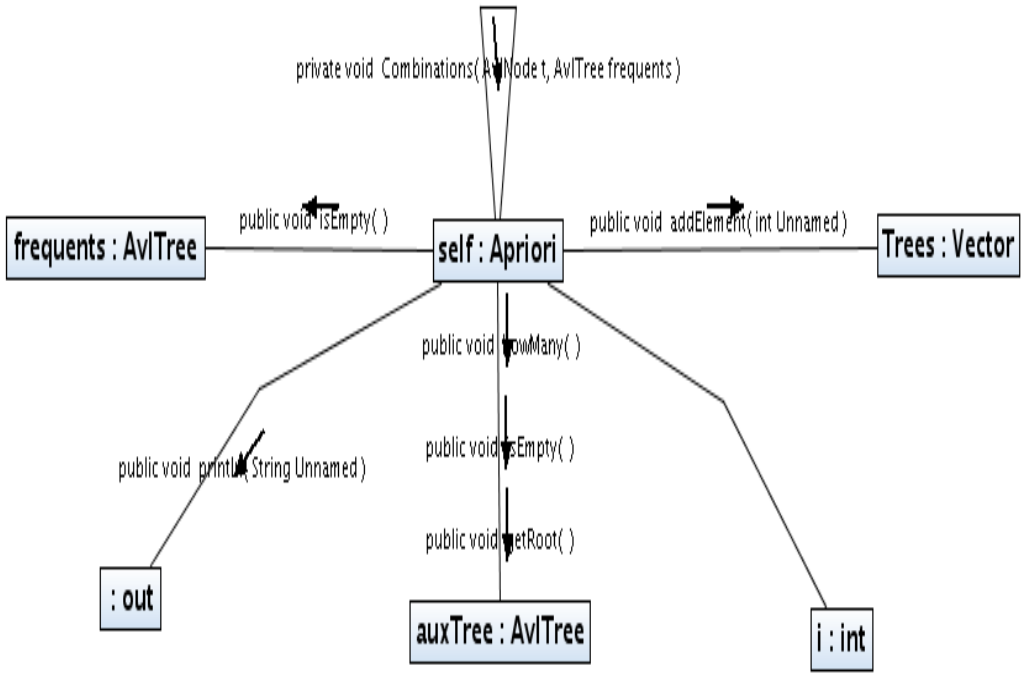
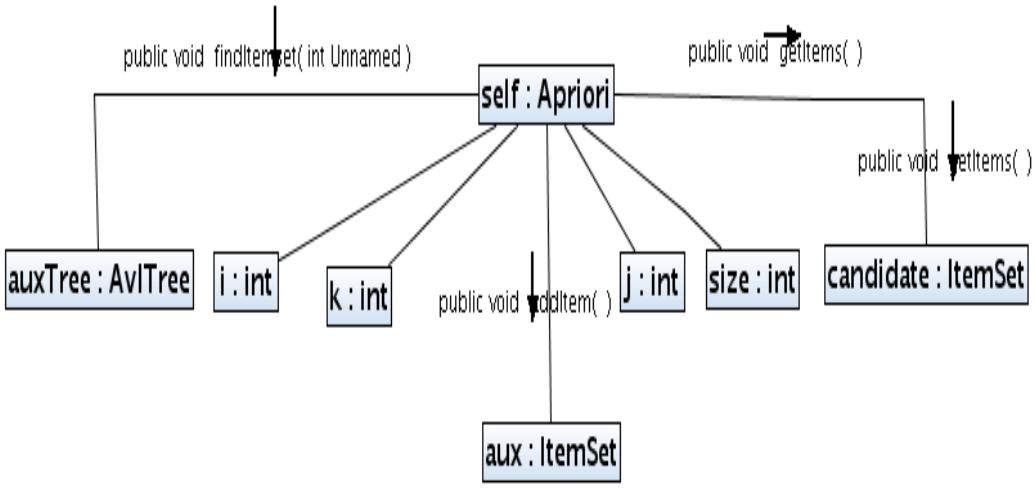


Figura 3.56: pruneCandidate



Clase EuipAsso

Figura 3.57: main

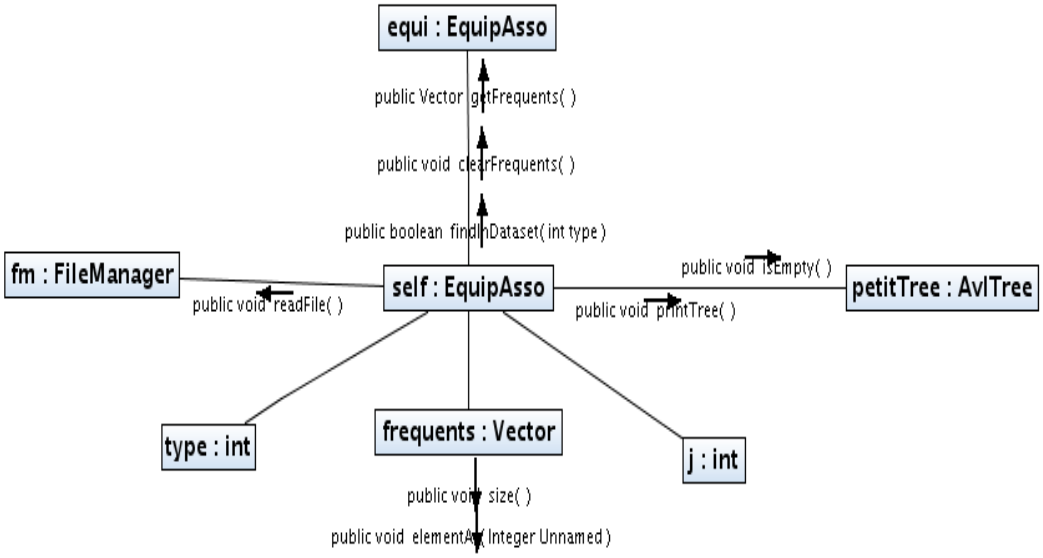


Figura 3.58: run

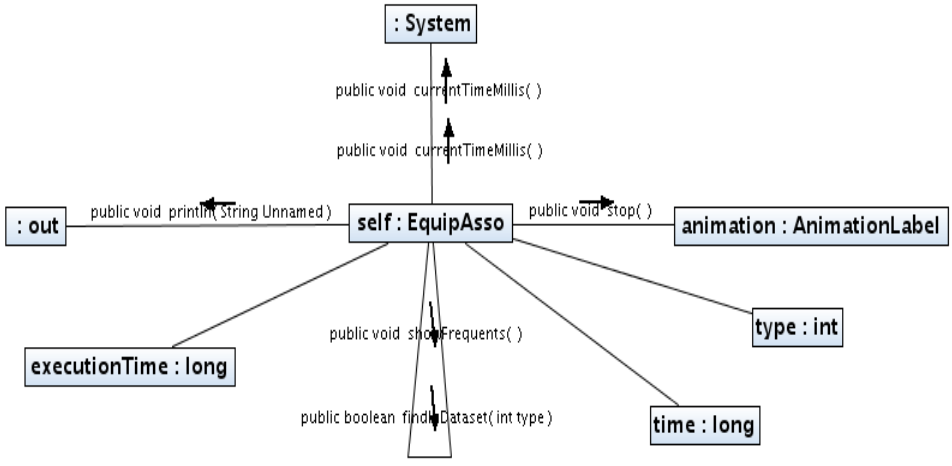
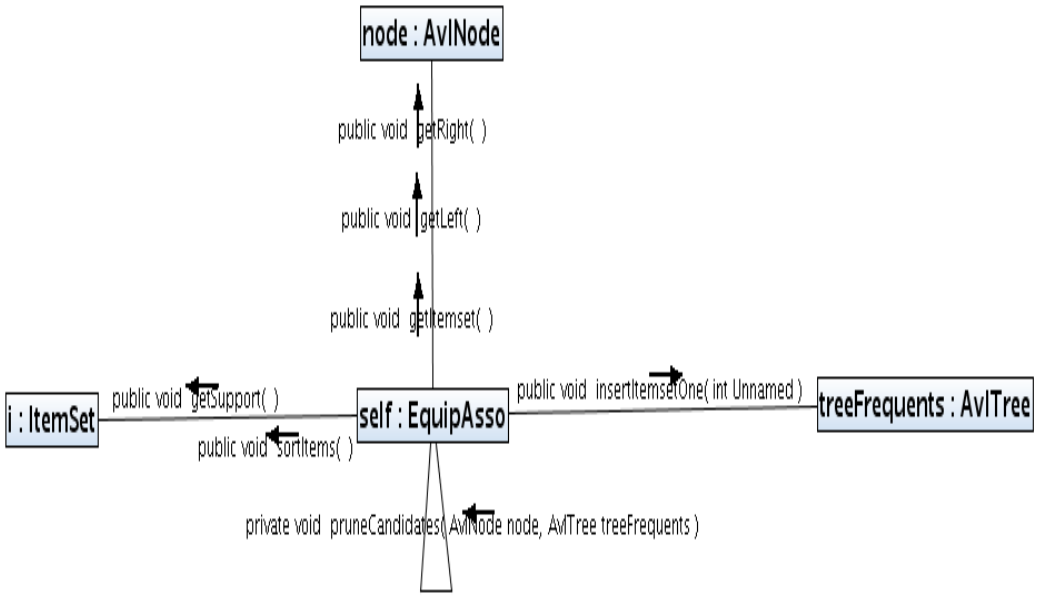
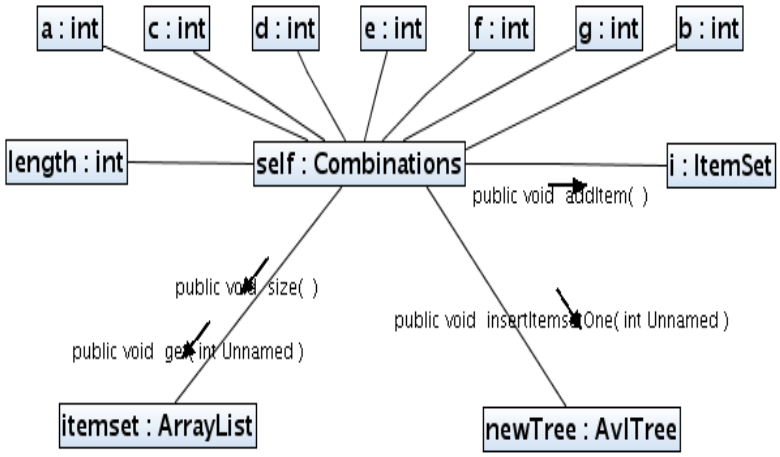


Figura 3.59: pruneCandidates



Clase Combinations

Figura 3.60: combine7



Clase BaseConditionals

Figura 3.61: addBaseConditionals

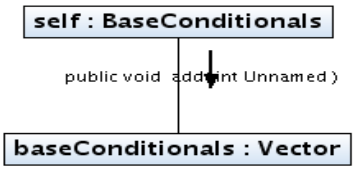
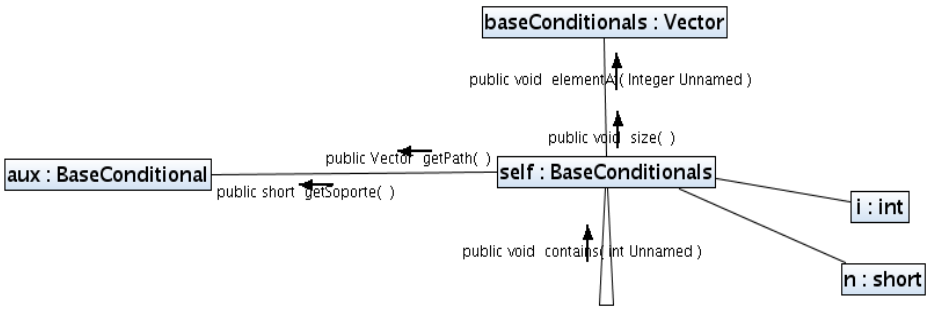


Figura 3.62: findItem



Clase Combinations

Figura 3.63: addCandidates

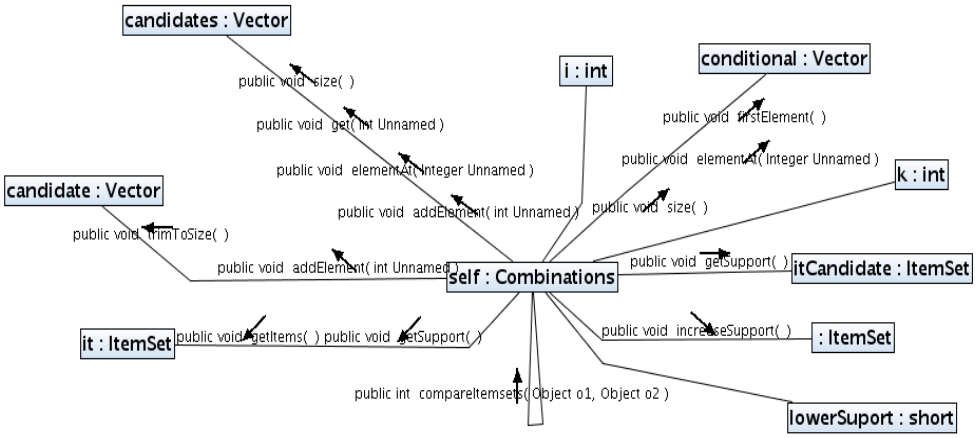
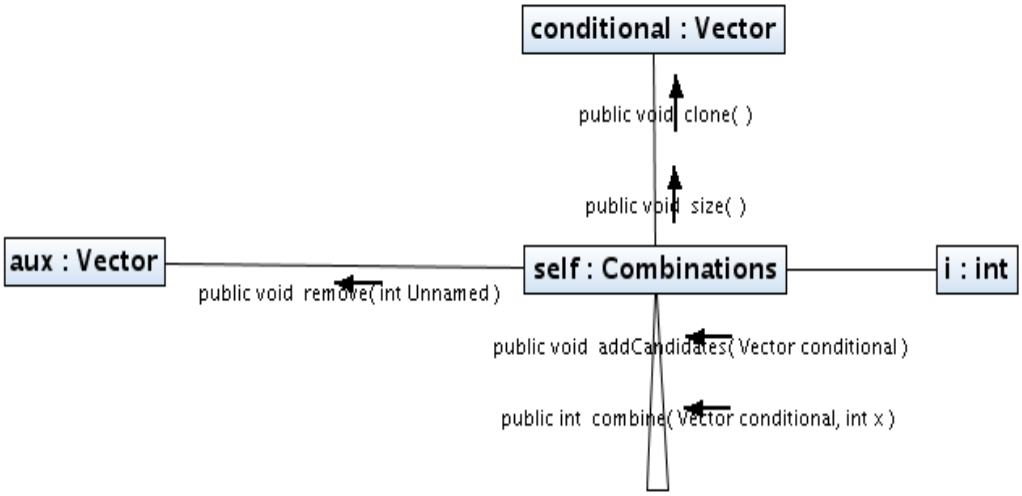
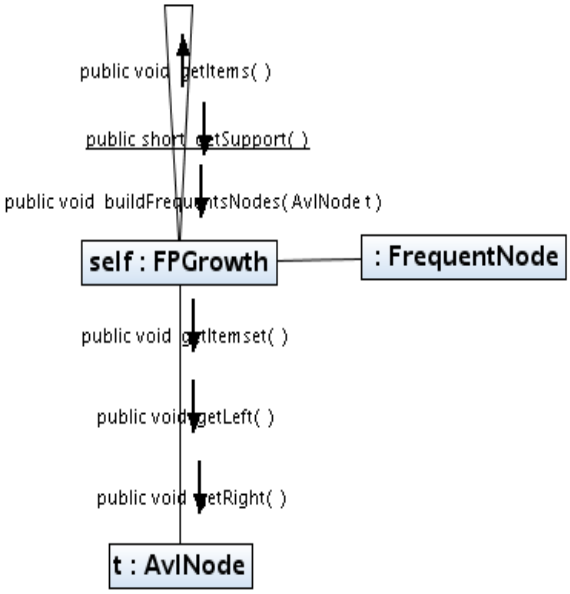


Figura 3.64: Combine



Clase FPGrowth

Figura 3.65: builtFrecuenceNode



Clase AvlTree

Figura 3.66: compareItems

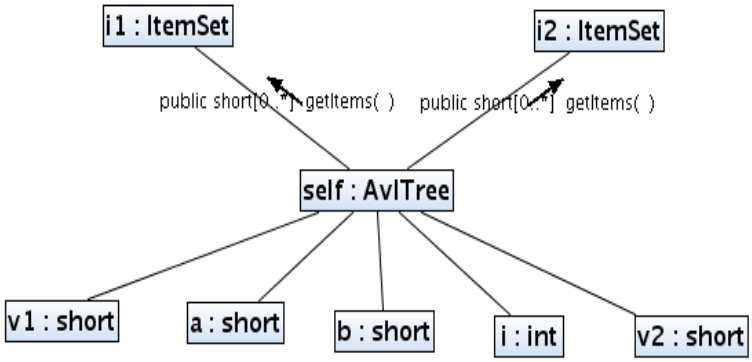
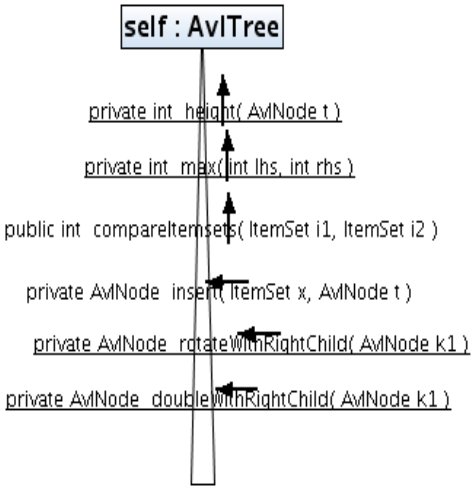


Figura 3.67: insert



3.2.2 Diagramas de Clase

Figura 3.68: Paquete KnowledgeFlow

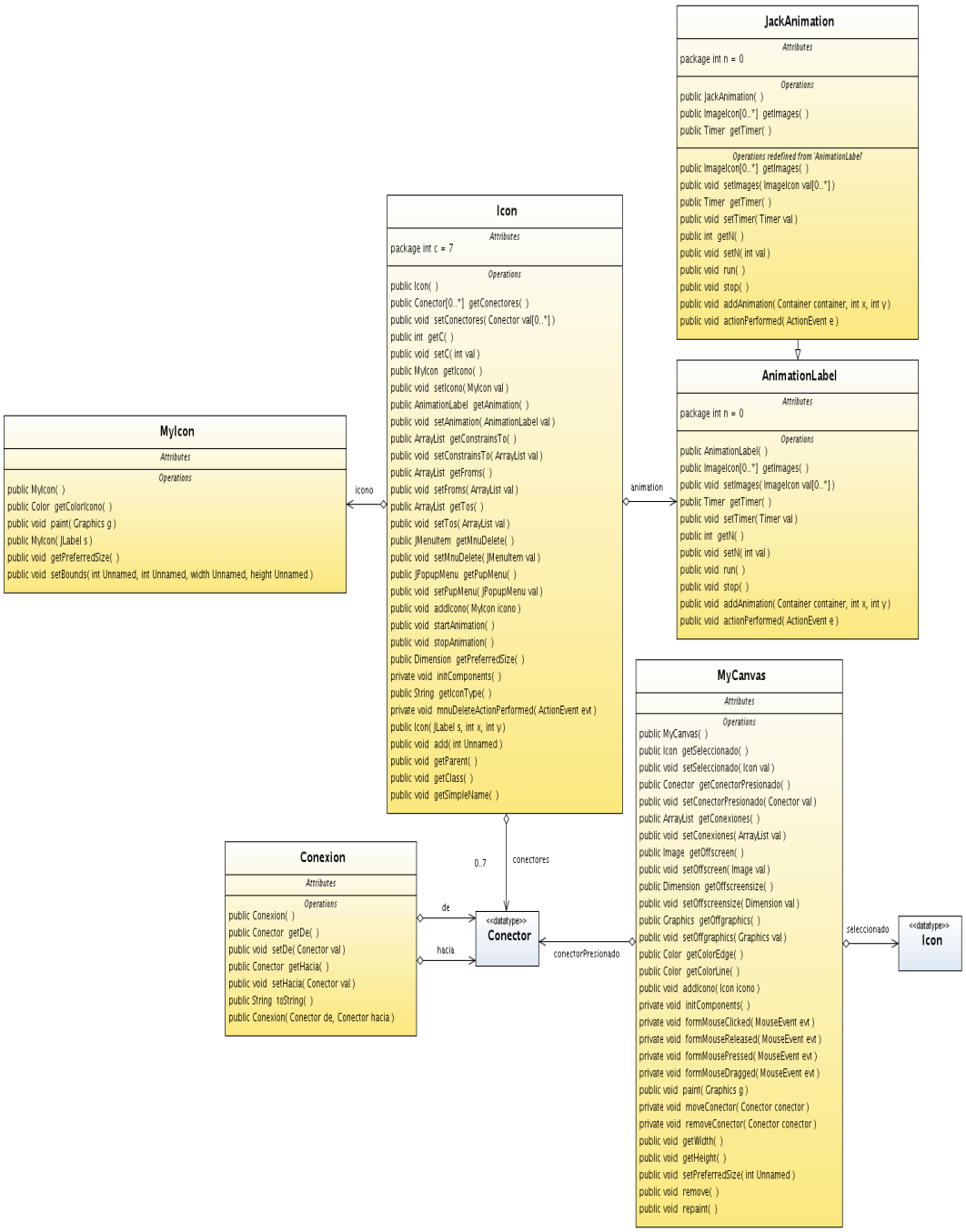


Figura 3.69: Paquete Utils

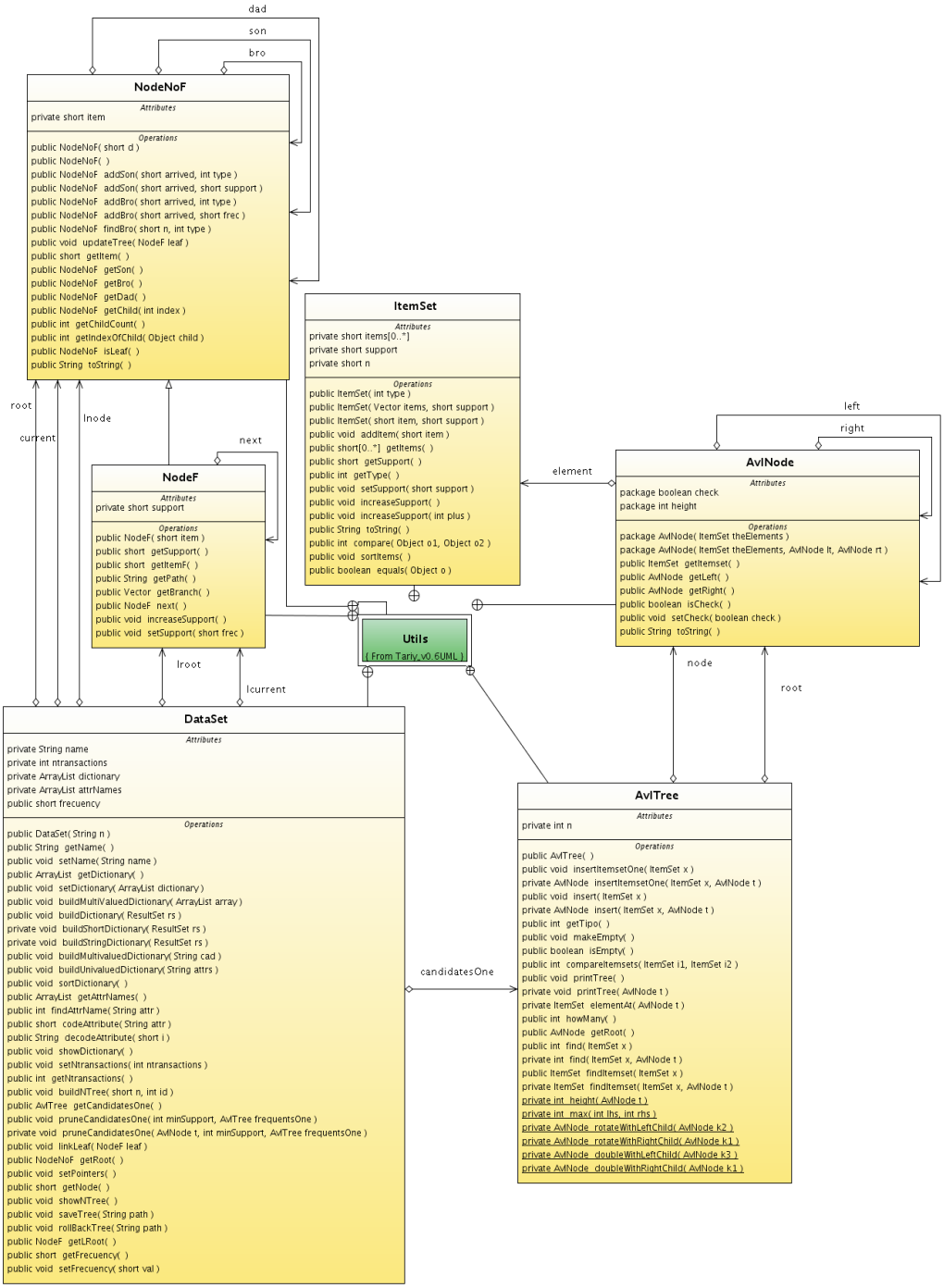


Figura 3.70: Paquete FPGrowth

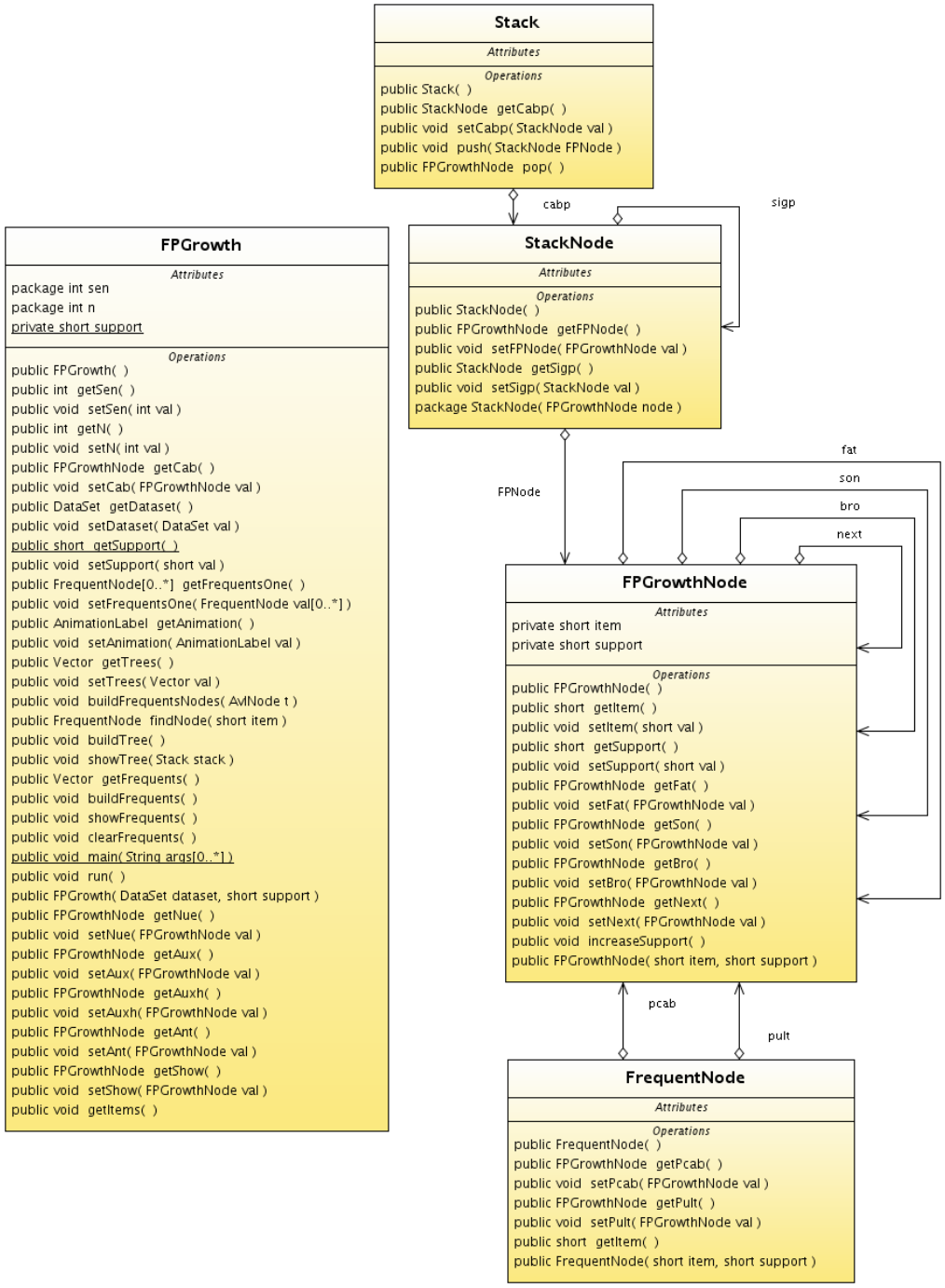


Figura 3.71: Paquete EquipAsso

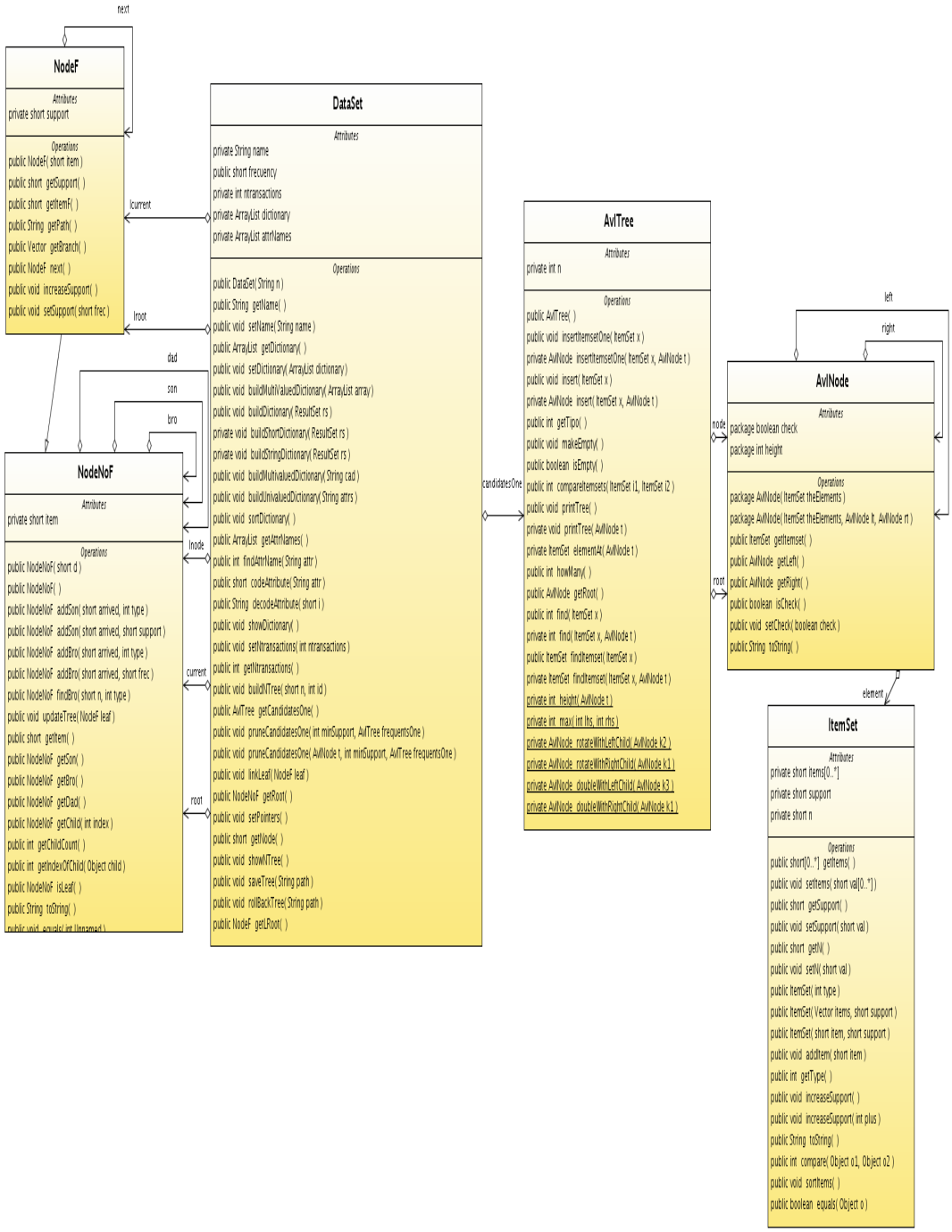


Figura 3.72: Paquete Mate

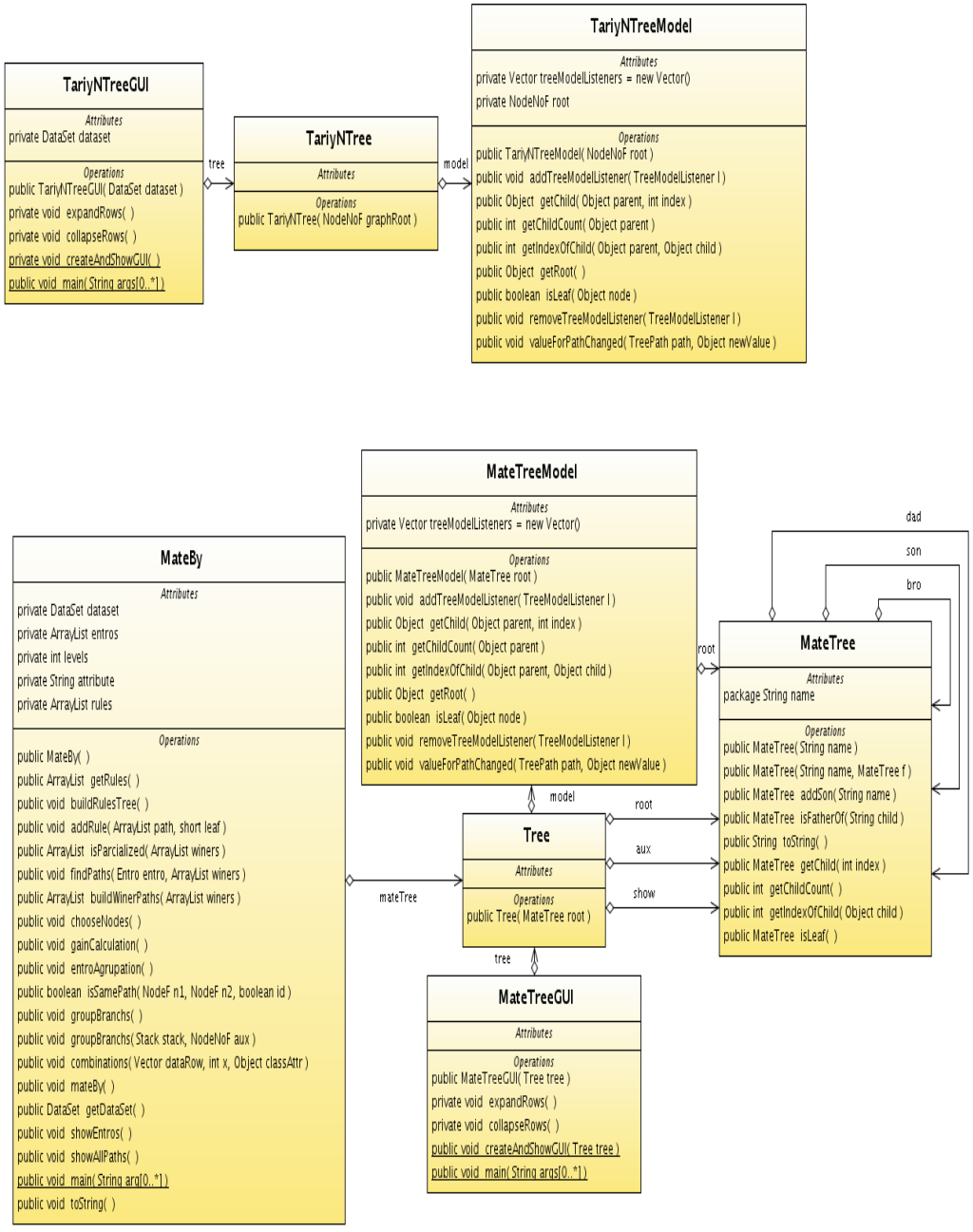
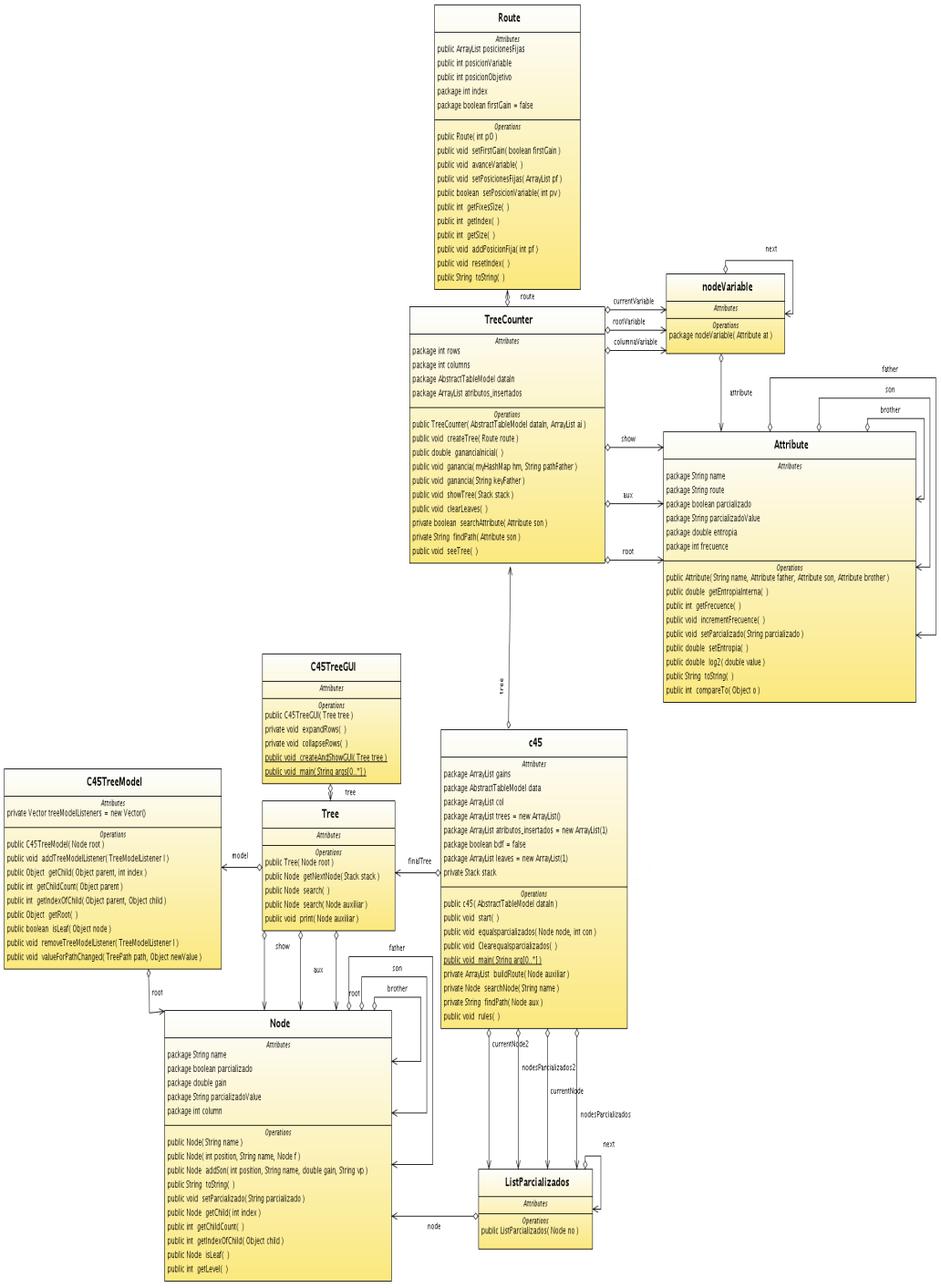


Figura 3.73: Paquete C45



3.2.3 Diagramas de Paquetes

Figura 3.74: Paquete Principal

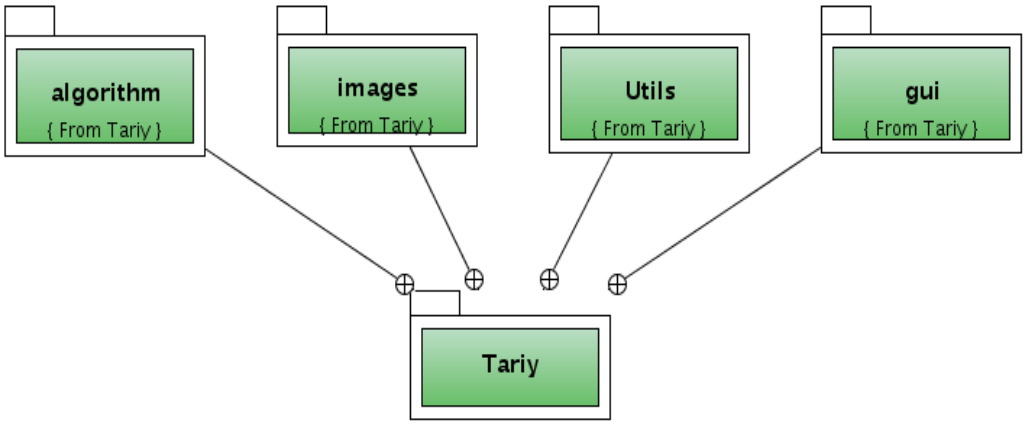


Figura 3.75: Paquete Algoritmos

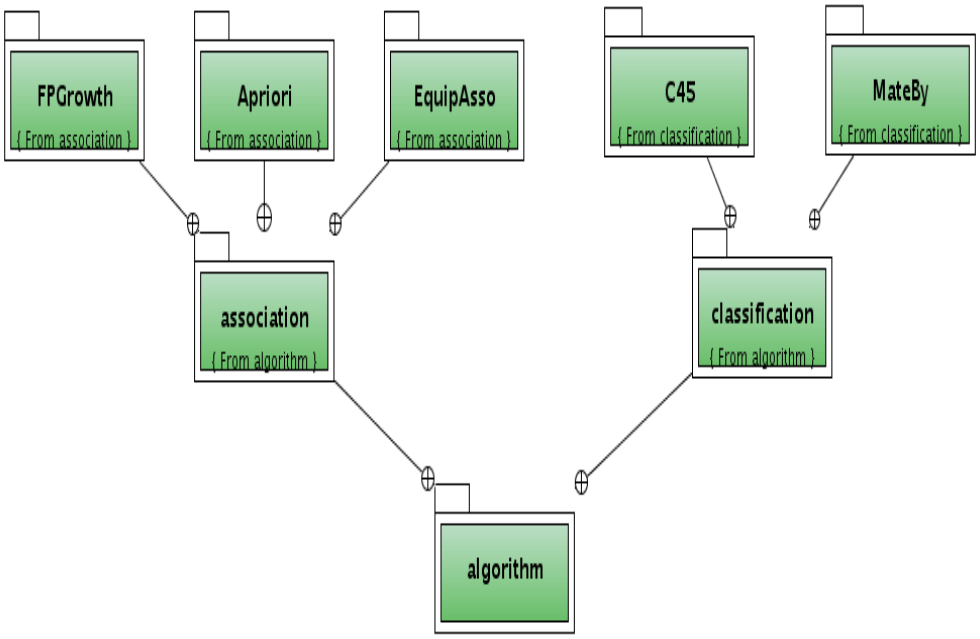


Figura 3.76: Paquete Interfaz Gráfica

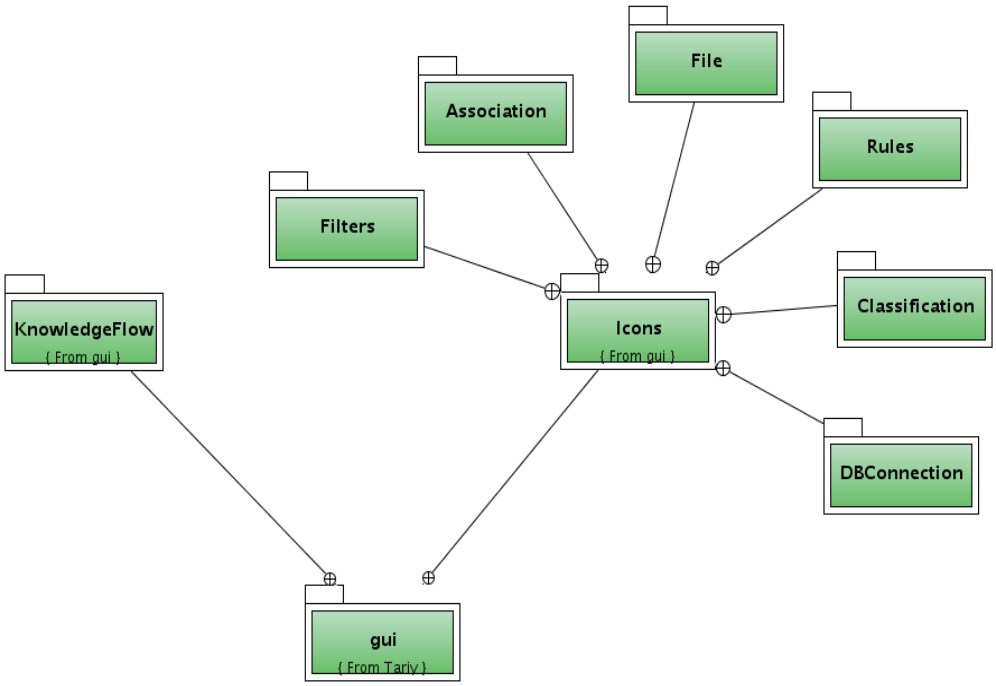
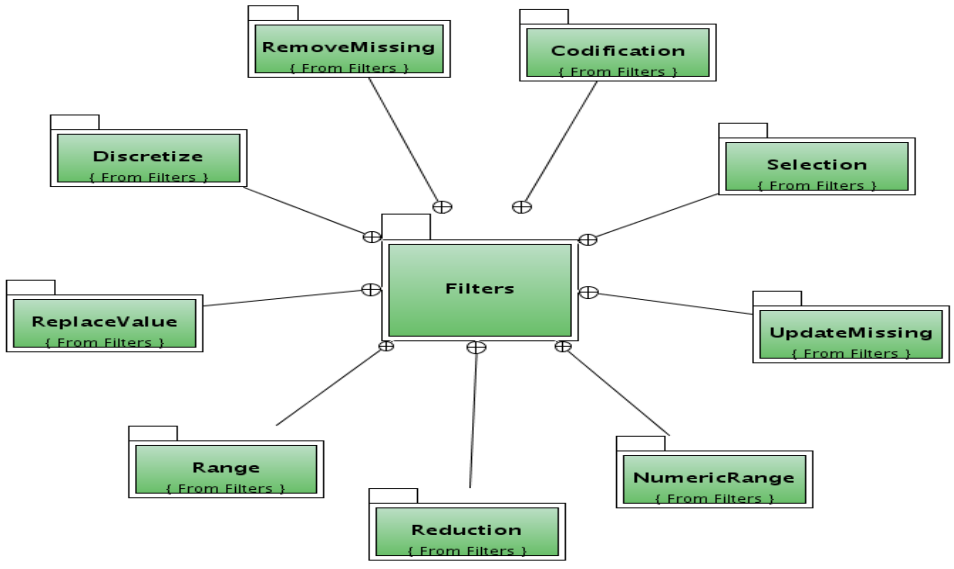


Figura 3.77: Paquete GUI Filtros



4. IMPLEMENTACIÓN

4.1 INTRODUCCIÓN

El sistema operativo sobre el cual se trabajó 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 09, lo que lo convierte en una herramienta independiente a la plataforma donde se ejecute.

Durante el desarrollo del proyecto se usaron en su mayoría herramientas de software libre con aplicaciones como NetBeans 5.5, como IDE de desarrollo, Subversion, para el control de versiones, The Gimp y KIconEdit, para la manipulación de imágenes e iconos, etc.

4.2 ARQUITECTURA DE TARIYKDD

Dentro del proceso de Descubrimiento de Conocimiento, *TariyKDD* comprende las etapas de Conexión, 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 5.1

4.2.1 Módulo de Conexión. Este módulo es el encargado de la conexión a una fuente de datos, esta puede ser un base de datos o directamente un archivo plano que repose en disco duro. El objetivo de este módulo es proveer al usuario de herramientas amigables para la selección y construcción de una vista o conjunto de datos minable.

4.2.2 Módulo de Utilidades. En este módulo se mantiene una colección de clases principales y librerías que son utilizadas por otras clases a lo largo de la aplicación para el desarrollo de tareas comunes como visualización y manipulación de datos. El objetivo de este módulo es tener centralizada la administración de este tipo de clases y aprovechar la reutilización de código.

4.2.3 Módulo del Kernel TariyKDD. En este módulo reposan los paquetes primordiales para la ejecución de las tareas de descubrimiento de conocimiento. Este reúne los módulos de Preprocesamiento, Algoritmos y Visores.

El módulo de preprocesamiento contiene los diferentes filtros y rutinas necesarias para la transformación y adecuación de los datos, si es necesario, antes de aplicar las técnicas de minería de datos.

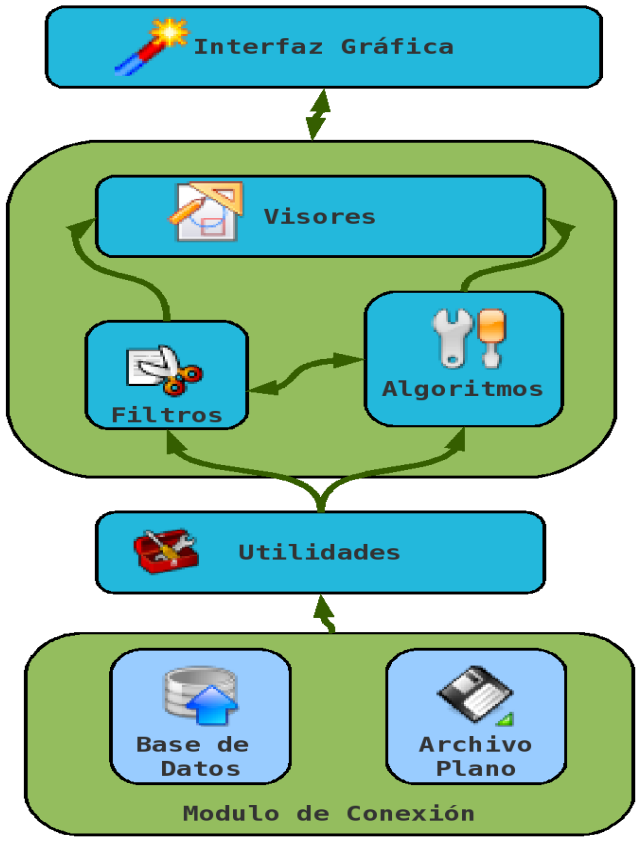
El módulo de algoritmos contiene las clases encargadas de aplicar las técnicas especificas de minería de datos. En TariyKDD se manejan rutinas para las tareas de asociación y clasificación.
El módulo de Visores contiene las clases necesarias para construir y desplegar las estructuras que permitirá visualizar de manera gráfica y dinámica los resultados de la

aplicación de técnicas de minería y preprocesamiento.

El objetivo de este módulo es permitir la interacción y comunicación entre los diferentes componentes de los módulos involucrados, así como de proveer soporte en el caso de no ser necesaria la utilización de alguno de ellos, por ejemplo, cuando una conexión se hace directamente hacia uno de los algoritmos de minería sin pasar por algún filtro.

4.2.4 Módulo de Interfaz Gráfica. Este módulo da soporte visual a los módulos de conexión, filtros, algoritmos y visores, y se encarga de brindar al usuario una medio agradable y fácil de usar para la construcción de experimentos que involucren la interacción entre los diferentes componentes de la herramienta.

Figura 4.1: Arquitectura *TariyKDD*



4.3 ESTRUCTURA DE PAQUETES Y CLASES

A continuación se describe de manera general la estructura de paquetes de *TariyKDD* y las clases que pertenecen a cada paquete. Posteriormente, se amplia y se detalla la forma en como fueron desarrolladas dichas clases.
Paquete utils: Utilidades de *TariyKDD*. Dentro de este paquete se encuentran las clases necesarias y comúnmente utilizadas por las demás clases.

Paquete algorithm: Esta compuesto 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*).

Paquete gui: Comprende los paquetes *Icons y KnowledgeFlow*, los cuales a través de sus clases implementan la interfaz gráfica de la herramienta.

4.3.1 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 raíz 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 ordenar las reglas generadas de acuerdo a varios parámetros: soporte, confianza u orden 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: aquí 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 árbol *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 son los siguientes:

root de tipo *Avlnode*: raíz 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 controlador 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 *contraseña*. 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 un ítem y el nombre del ítem 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 .*ar*ff

attributes de tipo *Object*: arreglo con los nombres de los atributos de un *archivode* acceso aleatorio .*ar*ff.

dictionary de tipo *ArrayList*: arreglo en el que se almacena el diccionario de datos de un archivo .*ar*ff

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.
Figura 4.2.1

writeString: método utilizado para escribir una cadena de caracteres en un archivo. Figura 4.2.2.

Figura 4.2.1: Escribir un *item* a archivo

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

Figura 4.2.2: Escribir una cadena a archivo

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

getfileSize: retorna el tamaño de un archivo en bytes. Figura 4.2.3

Figura 4.2.3: Tamaño de un archivo

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

getfileName: retorna el nombre del archivo de acceso aleatorio. Figura 4.2.4

Figura 4.2.4: Nombre de un archivo

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

closefile: cierra el flujo hacia el archivo de acceso aleatorio. Figura 4.2.5

Figura 4.2.5: Cierra conexión a archivo

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

deletefile: borra físicamente el archivo de acceso aleatorio. 4.2.6

Figura 4.2.6: Borrar un archivo

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

setOutChannel : ubica el flujo en una posición determinada del archivo de acceso aleatorio. Figura 4.2.7.

Figura 4.2.7: Manejo de posición en un archivo

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

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. Figura 4.2.8

Figura 4.2.8: Lectura de 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();
  }
  }
...
```

getAttributes: retorna el arreglo que tiene los nombres los atributos del archivo de acceso aleatorio .*ar*ff. Figura 4.2.9

Figura 4.2.9: Nombres de atributos de un archivo plano

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

getData: retorna la matriz que almacena los datos del archivo de acceso aleatorio .*ar*ff. Figura 4.2.10

Figura 4.2.10: Matriz de datos del archivo plano

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

getDictionary: retorna el diccionario de datos construido a partir del archivo de acceso aleatorio .*ar*ff.

buildMultivaluedDataset: este es uno de los métodos más importantes en la fase de carga de los datos desde el archivo plano a memoria. El archivo de extensión *ar*ff se recorre completamente pero solo se cargan los datos necesarios al *DataSet*. Por ejemplo, en un archivo *ar*ff 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 *.ar*ff, 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 .*ar*ff con los nombres de sus atributos y sus datos.

Clase itemset Esta clase se encarga del manejo de los conjuntos de ítem. Un *itemset* puede ser entendido como una transacción. Los atributos de esta clase son:

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

support de tipo short: soporte asociado a este *itemset.*

n de tipo short: tamaño del *itemset.*

Los métodos implementados son:

getítem: 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 ítem 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 artículos en una transacción.

getSize: devuelve el tamaño de una transacción.

getitemset: devuelve un ítem de la transacción.

srtBySupport: ordena las transacciones por soporte.

srtByItem: ordena las transacciones por el ítem.

loaditemsets: carga los artículos o ítem de una transacción.

clearTransaction: borra los ítem de una transacción.

4.3.2 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 Clasificación (*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 interactúa 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 parámetros: 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 alimentado haciendo uso del método *pruneCandidatesOne* de la clase *DataSet*, el cual recibe el soporte del sistema como parámetro. En este punto también se instancia e inicializa 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 posición 0 de este arreglo. Los detalles del constructor de esta clase se aclaran en el siguiente listado:

Figura 4.2.11: Constructor de la clase *Apriori*

```
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;
}
```

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 último árbol de *itemsets* 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 ítem, siendo n el tamaño de ese *itemset*, con el próximo *itemset* del árbol. De ser así, se instancia un nuevo *itemset* con los n − 1 ítem coincidentes y los 2 últimos ítem 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 evalúa 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 evalúan 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 la figura 4.2.12.

Figura 4.2.12: Función *pruneCandidates*

```
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;
}
```

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 ultimo 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 incluído 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 *FPGrowth* se basa en la generación de *itemsets* candidatos a partir de un árbol N-Ario. Para comprender mejor la implementación de *FPGrowth*, 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 arreglo de objetos de tipo *FrequentNode* los *itemsets* frecuentes-1 o de tamaño 1. Cada uno de los ítem 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 la figura 4.2.13.

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* frecuentes. Su estructura se puede ver en la figura 4.2.14.

Figura 4.2.13: Estructura FPGrowthNode

```
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.
```

Figura 4.2.14: Estructura FPGrowth

```
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.
```

Los parámetros que el algoritmo *FPGrowth* 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 arreglo *frequentsOne*. El siguiente paso es construir el *FPtree*, para esto se lee cada una de las transacciones de *DataSet* y para aquellos ítem cuyo soporte sea mayor o igual al mínimo se los almacena en la clase del paquete *Utils Transaction*. A partir de estos ítem filtrados se construye el *FPtree*, cuyo pseudocódigo se puede observar en la figura 4.2.15.

Para una mejor compresión sobre la construcción del *FPTree* se puede observar el conjunto de datos de la Tabla 4.1 y su representación gráfica en la figura 4.2.17 y para profundizar más en la implementación se puede observar en el Tabla 4.2.16 el código fuente de la función que construye el *FPTree (buildTree*):
Como se puede observar en la figura 4.2,17 cada uno de los ítem frecuentes-1 se encuentra enlazado a un nodo del *FPtree*, por ejemplo el ítem 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.

Figura 4.2.15: Pseudocódigo construcción FPtree

```
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
```

El siguiente paso de la implementación consiste en recorrer la estructura de los ítem frecuentes-1 y por cada uno de estos construir sus Patrones Condicionales Base, los cuales se obtienen recorriendo *FPtree* desde cada nodo enlazado por los ítem frecuentes-1 a través de su puntero al nodo padre hasta la raíz. Por ejemplo, como se puede observar en la figura 4.2.17 para el ítem 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 ítem frecuente-1 en cuestión. Cada uno de los Patrones Condicionales Base se almacenan en la clase *BaseConditional* y todo el conjunto de Patrones Condicionales Base de un ítem frecuente-1 se almacenan en la clase *BaseConditionals*. En el Tabla 4.2 se pueden observar los Patrones Condicionales Base de cada uno de los ítem frecuentes-1.

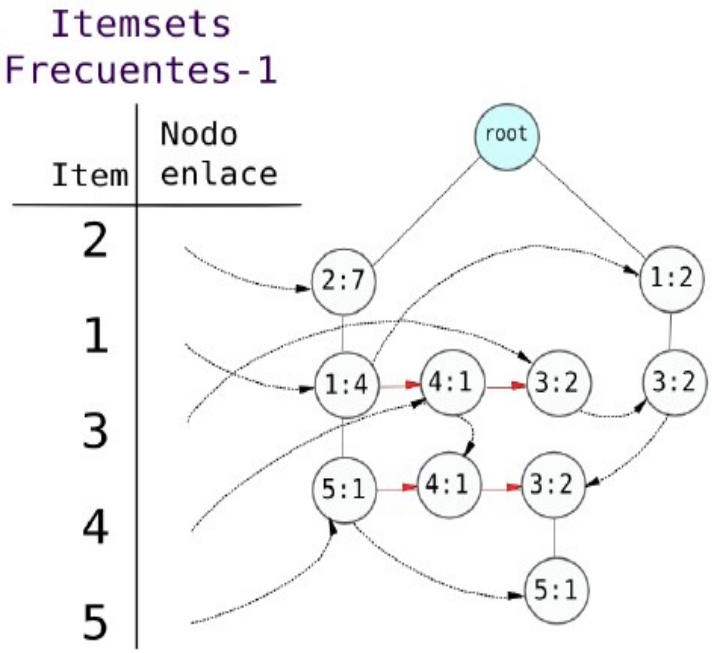
Figura 4.2.16: Función *buildTree*

```
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);
}
```

Tabla 4.1: Conjunto de datos transaccional

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)}

Figura 4.2.17: Árbol *Fptree*



Cuando se han construido todos los patrones condicionales Base de un ítem frecuente-1, el siguiente paso de la implementación es determinar cuales de sus ítem tienen soporte mayor o igual al mínimo. Para esto se debe sumar cada uno de los soportes que un ítem tenga en cada patrón condicional base, los ítem 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 ítem frecuente-1 son (1:2) y (2:2), se puede observar que el ítem 3 no cumple con el soporte, por tanto no es patrón condicional. Los patrones condicionales de los ítem frecuentes-1 se pueden observar en el Tabla 4.2.

Tabla 4.2: Patrones Condicionales e *itemsets* frecuentes

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)

El último paso de la implementación es determinar el conjunto de *itemsets f*recuentes. Para lo cual, se hace uso de la clase *Combinations*, la cual toma los patrones condicionales y los combina con los ítem 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 ítem, podemos ver sus *itemsets* Frecuentes en el Tabla 4.2.

Así como para *Apriori y EquipAsso* los *itemsets* Frecuentes se almacenan en un Vector de árboles 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 ítem que pasan el umbral 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 fue 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*.

Figura 4.2.18: 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);
}
```

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 ítem que sobrepasen el soporte del sistema y están 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:

Figura 4.2.19: Llamado de *loadTransaction*

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

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 ítem que están 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 ítem provenientes del *dataset* supere el soporte, ni sea encontrado entre los *itemsets* Frecuentes uno, por lo que se cargará una transacción vacía esta es descartada y se continua con la siguiente transacción.

La clase *Combinations* recibe como parámetro un arreglo que contiene los ítem 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 genera 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 *ítemset*. 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 árbol *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:

Figura 4.2.20: Función *pruneCombinations*

```
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);
  }
}
```

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 repetir 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 fórmulas 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 se calcula 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 TariyKDD C4.5 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 es hace 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.

Figura 4.2.21: Estructura 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;
                        }
                }
          }
        }
```

Al resultado de este conteo, son aplicadas las fórmulas 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:

$$Gain(S, A) = entropía(S) - \sum v \in V\,alores(A)|Sv|/|S|entropía(Sv)$$

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*.

entropía(S) = − ∑*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:

Figura 4.2.22: Función para encontrar la 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);
}
```

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 ene ario gráfico denominado *finalTree*, el cual extiende la clase de Java *Jtree*.

Ejemplo de funcionamiento del algoritmo C 45 en TARYI:

Primero en la tabla 4.3 se pueden observar datos de entrada, para el algoritmo C45:

En la tabla 4.3 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 se obtiene 9 valores ”SI” y 5 valores ”NO”, como se observa en la figura 4.3.

A estos resultados se aplica la fórmula para obtener la entropía inicial.

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

Se aplica 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 la figura 4.4.

Figura 4.3: Conteo *target*

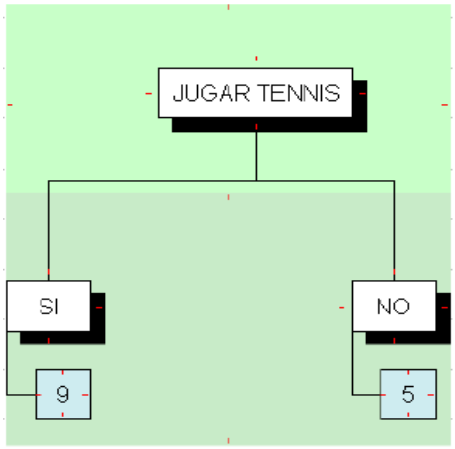
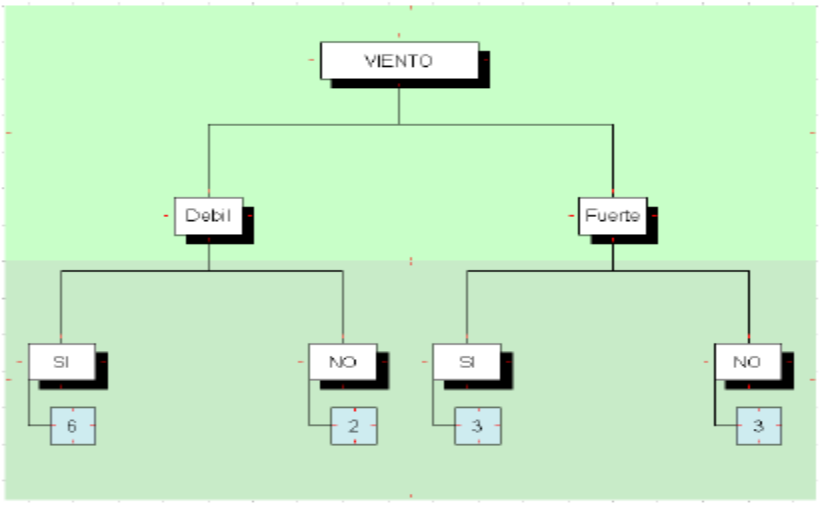


Tabla 4.3: Conjunto de datos

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

Figura 4.4: Conteo viento



Con estos resultados y la entropía inicial se aplica la fórmula para obtener la entropía del atributo ”VIENTO” relacionado con el atributo objetivo, de la siguiente forma:

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

$Gain(S,Viento) = entropía(S)-(Sdebil*entropía(Sdebil)+Sf\ uerte*entropía(Sf\ uerte)$ = 0,940 − 8/14entropía(Sdebil) − 6/14entropía(Sf uerte)

Calculamos $E(Sdebil) = E([6+, 2-]) = -(6/8)\ Log_2(6/8) - (2/8)\ Log_2(2/8) = 0,811$
$E(Sf\ uerte) = E([3+, 3-]) = -(3/6)log_2(3/6) - (3/6)log_2(3/6) = 1$

Reemplazando :

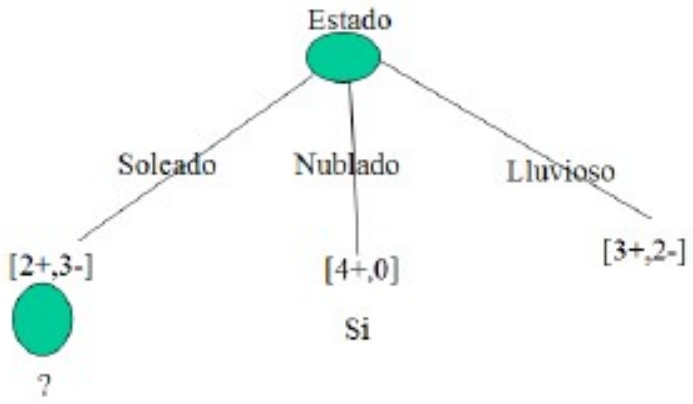
$Gain(S, viento) = 0,940 - 0,463 - 0,428 = 0,048$

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:

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

Se puede 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 se pueden encontrar 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 4.5: *árbol parcial*



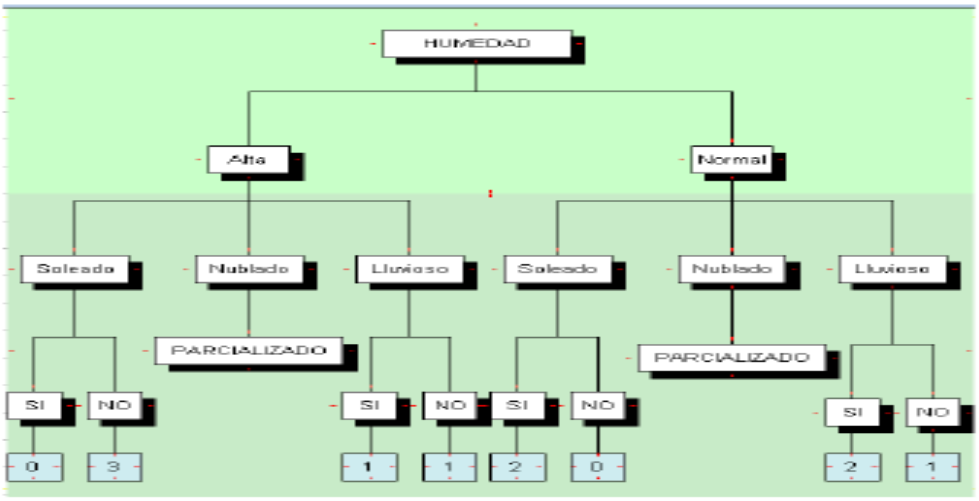
Ahora se encuentra el atributo que brinde la mayor ganancia para cada valor del atributo ”ESTADO”, haciendo el conteo de forma eficiente al encontrar los resultados para los tres valores simultáneamente. Como ejemplo lo haremos para Humedad, combinado con el atributo Estado, tal y como se puede observar en la figura 4.6.

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(Débil[1+, 2−], fuerte[1+, 1−])
Gain(Soleado, Viento) = 0,970 − (3/5) * 0,918 − (2/5) * 1 = 0,019

Figura 4.6: Conteo humedad estado



Se observa 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 fórmula de entropía

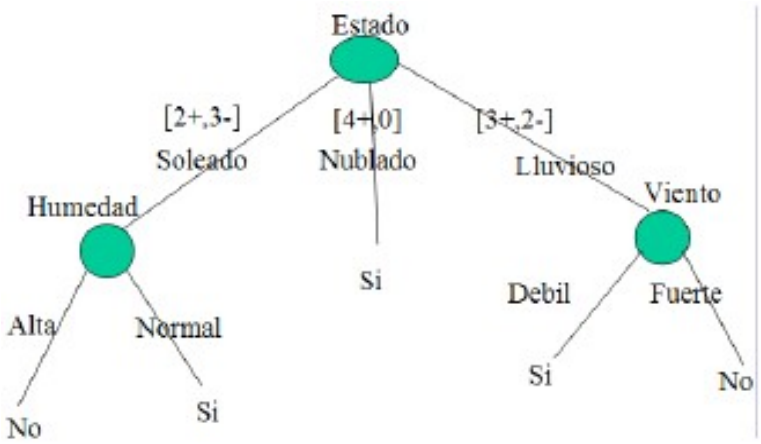
Humedad(Alta[1+, 1−], Normal[2+, 1−])
Gain(Lluvioso, Humedad) = 0,97 − (2/5) * 1 − (3/5) * 0,917 = 0,0198

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(Débil[3+, 0−], fuerte[0+, 2−])
Gain(Lluvioso, Viento) = 0,970 − (3/5) * 0 − (2/5) * 0 = 0,970

Se observa 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 4.7: árbol definitivo



Paquete mate A continuación se describen las clases implementadas en la programación del algoritmo *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 A*rrayList*: es un arreglo de datos en el que se va a almacenar las agrupaciones de combinaciones que tienen que hacerse para el cálculo de la entropía.

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

attribute de tipo *String*: aquí se almacena el nombre del atributo que se encuentra inmediatamente después de la raíz y por el cual se inició el recorrido del árbol en una búsqueda.

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

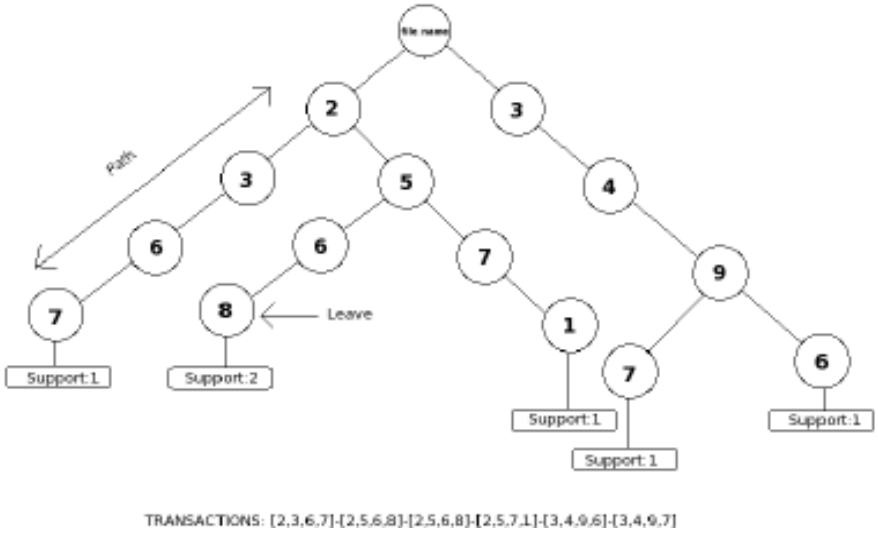
mateTree de tipo *Tree*: este es un árbol N-ario se crea para ser mostrado gráficamente.

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 árbol tienen una estructura diferente al resto de nodos en el árbol que le permite guardar el número de ocurrencias de una combinación. Este campo en las hojas es llamado *soporte* y serán de gran utilidad posteriormente en el cálculo de la ganancia. La figura 4.8 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 están asociadas y que deben unirse para luego calcular la ganancia. El criterio de agrupación se basa en la coincidencia del número de niveles y de la ruta de 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 4.10. Cada uno de los elementos entro es almacenado en un arreglo llamado *entros* para mejorar la manipulación de estos objetos. El proceso continua hasta recorrer todas las ramas del árbol y hasta haber agrupado todas las hojas relacionadas entre si.

Figura 4.8: 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 la figura 4.9.

Método gainCalculation Dentro de esta clase se recurre nuevamente al arreglo *entros*. Hasta este momento se ha calculado la entropía de cada grupo de hojas y ahora se pasa 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 están inmediatamente después de la raíz 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 entropía 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.

Figura 4.9: Cálculo de Entropía

```
...
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);
  }
...
```

Figura 4.10: 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 ganancia se debe establecer la hoja que hace parte de la regla 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 construido 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 *dataSet* y el diccionario.

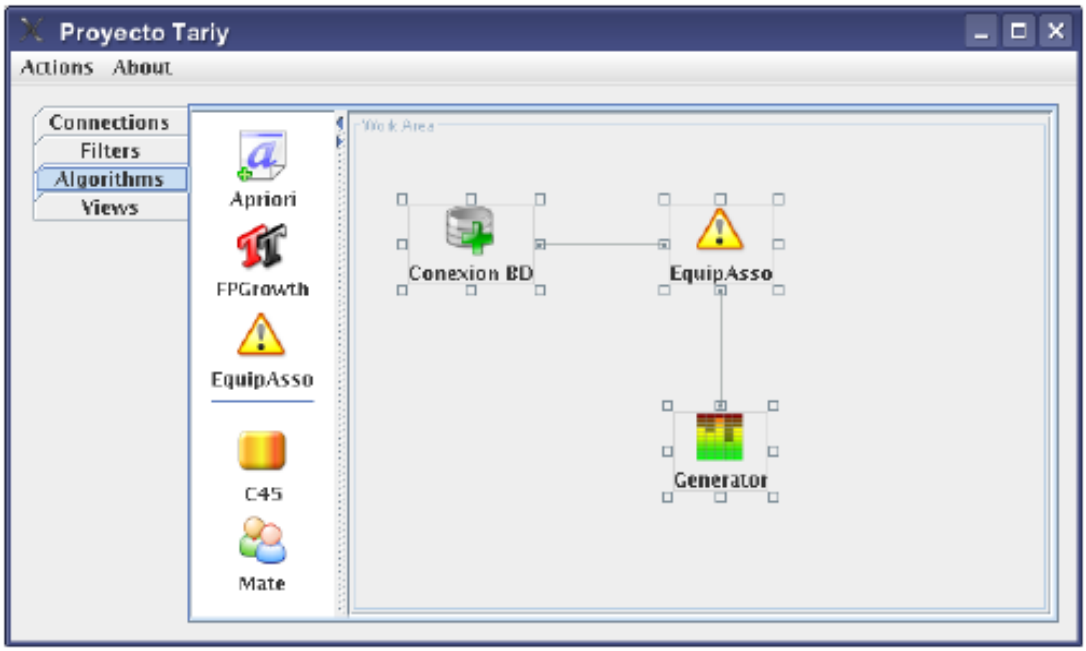
4.3.3 Paquete GUI Dentro de la implementación de una interfaz gráfica amigable para el proyecto *TariyKDD*, se trabajó utilizando las funcionalidades del proyecto *Matisse*, el constructor de interfases gráficas de usuario (GUI) propia de *NetBeans* 5.0 que permite un diseño visual de las formas y su 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 una 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 4.11 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 Área de Trabajo, donde se organizan los iconos seleccionados y se construye un experimento de descubrimiento de conocimiento.

Figura 4.11: 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 escoge 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 4.12.

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 4.13, 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 4.12: Etiquetas dentro de *Chooser* para la selección de etapas

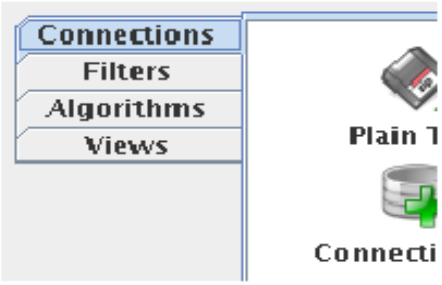


Figura 4.13: Paneles para cada etapa del proceso KDD

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 directamente del *Classpath* de la aplicación dentro del paquete *images*. Es importante aclarar que en este momento también se asígna 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 la figura 4.14

Figura 4.14: asignación de valores a Iconos

```
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);
```

Se puede notar en la primera línea como se asígna una imagen al *JLabel* utilizando el método *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 ilustra como es capturado un *JLabel* al ser presionado con el mouse.

Figura 4.15: Captura de *JLabel*

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

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 icono 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 4.16 se muestra la inclusión de 2 instancias de la clase *Icon* y la relación establecida entre ellos a través de sus conectores.

Figura 4.16: Implementación de la clase *Icon*

Se implementa dentro de esta clase la funcionalidad un menú desplegable donde se pueda incluir funciones propias para cada icono y la inclusión de una animación que se activa cuando el icono 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 *JPopupmenú* y *JmenúItem*. 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 *JmenúItem* y se setéa su propiedades de name a "*Delete*", posteriormente se procede a adicionar esta instancia de *JmenúItem* al *JPopupmenú* 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 icono con otros iconos y borrar el componente del área de trabajo. Nuevas adiciones al menú desplegable pueden ser hechas al instanciar objetos *JmenúItem* e incluirlos dentro del *JPopupmenú* 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 donde se ilustra el dibujado de esta clase al sobrecargar el método *paint* de la clase *Jcomponent*.

Figura 4.17 Dibujado de la clase conector

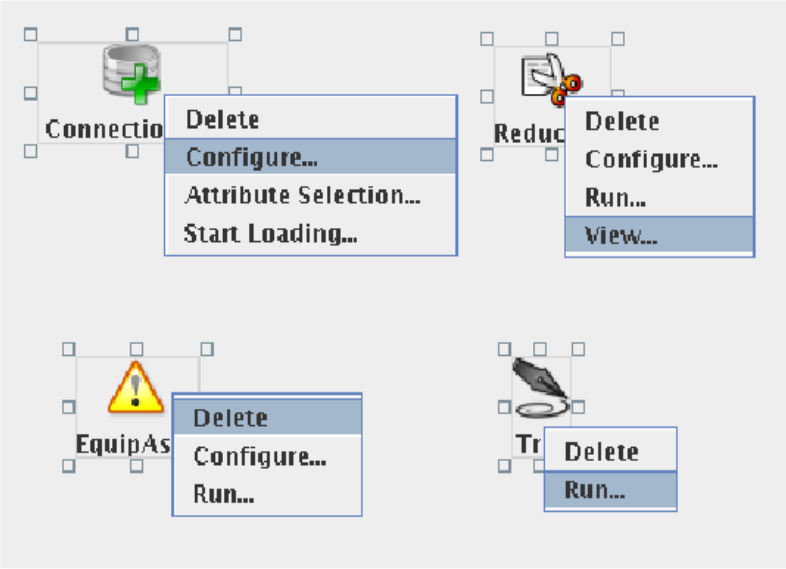
```
public synchronized void paint(Graphics g){
    g.setColor(Color.Blue);
    g.drawRect(0, 0, 6, 6);
    if(selected){
        g.fillRect(2, 2, 3, 3);
    }
}
```

Se observa 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 se puede escoger colores para el conector y dibujar un rectángulo de 7x7 pixels que representará el área del conector que puede ser pulsada por el usuario para su selección y dibujaría 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 *JPopupmenú* 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 4.18.

Figura 4.18: Clases heredadas de la clase *Icon*



La clase *MyCanvas* será la encargada de gestionar las conexiones entre los icono y su movimiento sobre el área de trabajo. Esta clase extiende un *JPanel* y monitoréa 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 icono 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 para explicar el método *MousePressed*.

Figura 4.19: Click sobre un icono

```
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();
}
```

Para el evento *MouseDragged* se identifica cual fue el componente seleccionado en el evento *MousePressed*, si se trata de un icono 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 mueve. Estas dos acciones se realizan en el método *paint(Graphics g)* de la clase *MyCanvas*. Se muestra el siguiente fragmento de código4.2.27 para ilustrar lo hecho en este método.

Para el evento *MouseRelease* se debe identificar igualmente cual componente esta seleccionado si se trata de un icono, 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 icono que este disponible se establecerá una relación entre ambos iconos 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 el fragmento de código figura 4.20 y fragmento de código 4.2.1.

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 icono, desplegar el menú emergente. El siguiente fragmento de código 4.22 ilustra este método.

Figura 4.20: Evento Arrastrar y soltar

```
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();
}
```

Figura 4.21: Liberación del click del mouse

```
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();
}
```

Figura 4.22: Evento *Mouse Clicked*

```
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);
    }
  }
}
```

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 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 están 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 iconos 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 iconos a los que pertenecen trazando líneas que relacionan de manera visual un icono 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. Recuerde que la Clase Conector hereda a la clase *JComponent*, es decir que hereda los métodos *getX()* y *getY()* para calcular su posición dentro del la clase que los contiene, que en este caso es *MyCanvas*. De esta manera, se puede calcular las coordenadas de los conectores y trazar las líneas que representan la relación. A continuación se presenta un fragmento de código que ilustra lo explicado anteriormente.

Figura 4.23: Conectores y trazos de líneas

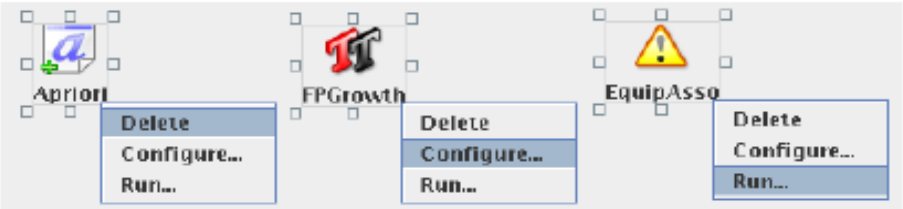
```
public synchronized void paint(Graphics g){
    int  xd, yd, xh, yh;
    super.paint(g);
    Iterator it = connections.iterator();
    while(it.hasNext()){
        Connection aux = (Connection)(it.next());
        xd = aux.from.getX();
        yd = aux.from.getY();
        xh = aux.to.get_X();
 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

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 menú contextual de cada icono.

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 *DBConnection*, *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 y que puede contener tres estados al hacer alusión al algoritmo Apriori, FPGrowth o EquipAsso. Estos estados se muestran en la figura 4.24.

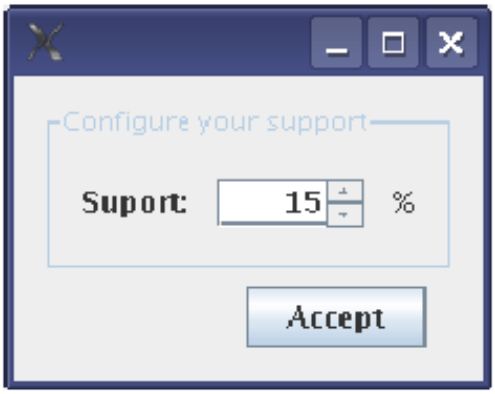
Figura 4.24: Estados de la Clase *AssociationIcon*



En la figura se puede apreciar el contenido del menú desplegable de cada icono: *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 4.25 ilustra el contenido de esta ventana.

Figura 4.25: Captura del soporte del sistema



La última 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 *DBConnectionIcon* 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 explica lo ejecutado por este método.

Figura 4.26: Ejecución de un comando con el mouse

```
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();
    }
}
```

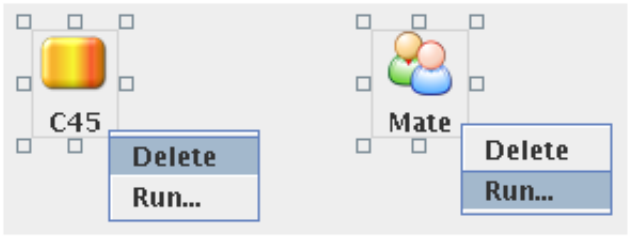
El resultado de ejecutar cualquiera de las clases de asociación que se han implementado devolverá a la Clase *AssociationIcon* un Vector el cual contiene un arreglo de los árboles Avl (instancias de la Clase *AvlTree*) que organizan el conjunto de itemset frecuentes obtenidos al ejecutar un determinado algoritmo. Este vector es

guardado en la variable *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 4.27.

Figura 4.27: Estados de la Clase *ClassificationIcon*



La única opción implementada en el menú es *Run* ya que este tipo de algoritmos 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 explica lo ejecutado por este método.

Figura 4.28: Ejecución de un algoritmo

```
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();
    }
}
```

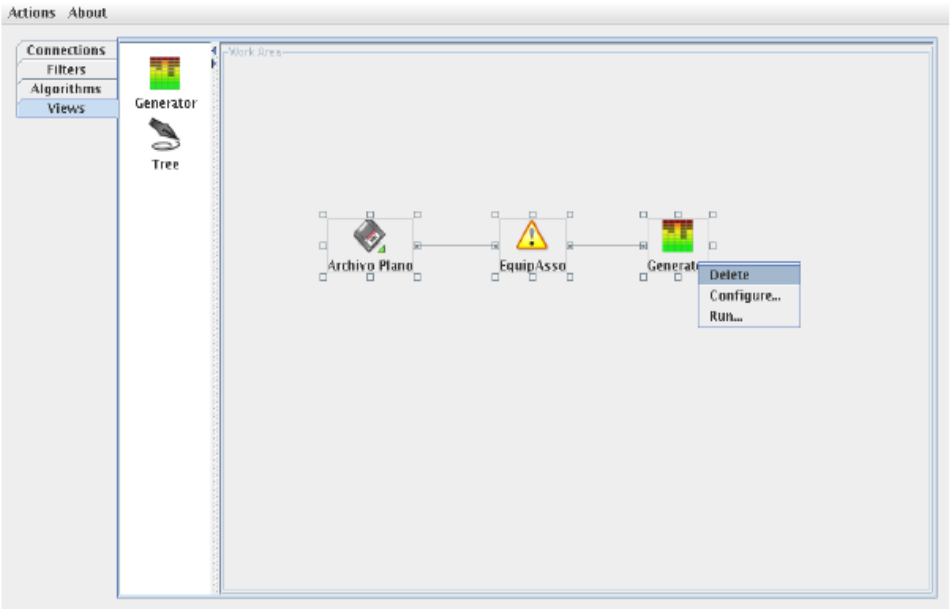
El resultado de ejecutar cualquiera de los algoritmos de clasificación será dos objetos de las Clases *JPanel y ArrayList*. El primero contendrá una extensión de *JPanel* donde se despliega un objeto de tipo *JTree* que contiene gráficamente el árbol de decisiones que resulta del análisis y el *ArrayList* 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.

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 figura 4.29 se pueden ver las funciones que esta clase implementa.

Figura 4.29: Funciones de *RulesIcon*



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 introduce una entrada invalida, en cuanto se pierda el foco del *JSpinner* este tomara el ultimo 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 Tabla 4.4.

Tabla 4.4: Diccionario de datos

Dato códifi-cado	**Item**
1	Jabón
2	Arroz
3	Champú
4	Desodorante

De acuerdo con lo anterior, una posible regla podrá ser: 1 & 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 consecuente.

A continuación, las reglas se decodifican, o sea que la regla 1 & 2 => 3 se almacenará en el array de la clase *AssocRules*, pero de la siguiente manera: Jabón & Arroz => 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.

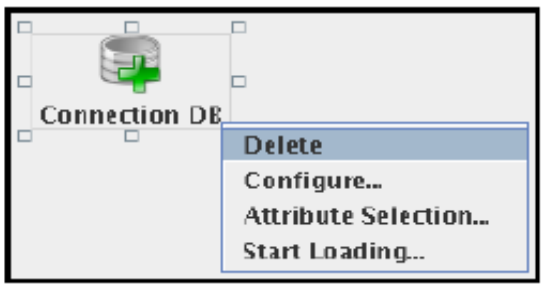
La tabla que muestra los datos se encuentra en la clase *showRules*, e implementa un evento del mouse para que cuando el usuario haga click en el encabezado Rules de la tabla, las reglas se ordenen por un criterio que ha sido determinado ”Pepita de oro”, o sea que en las primeras posiciones se indicaran las reglas cuyo antecedente sea menor que su consecuente, 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 reglas 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:

4.3.4 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 4.30 muestra la implementación de la clase *DBConnectionIcon* con sus opciones de menú .

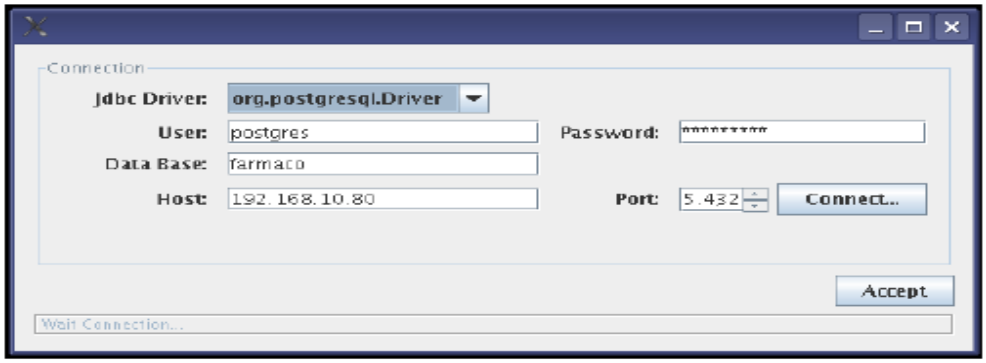
Figura 4.30: Implementación de la Clase *DBConnectionIcon*



Al heredar de la clase *Icon*, *DBConnectionIcon* posee por defecto la opción *Delete* en

su menú para eliminar este icono 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 4.31.

Figura 4.31: Implementación de la Clase *DBConnectionIcon2*



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 interfases involucradas en la conexión a bases de datos son:

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 estética 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 será 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);
```

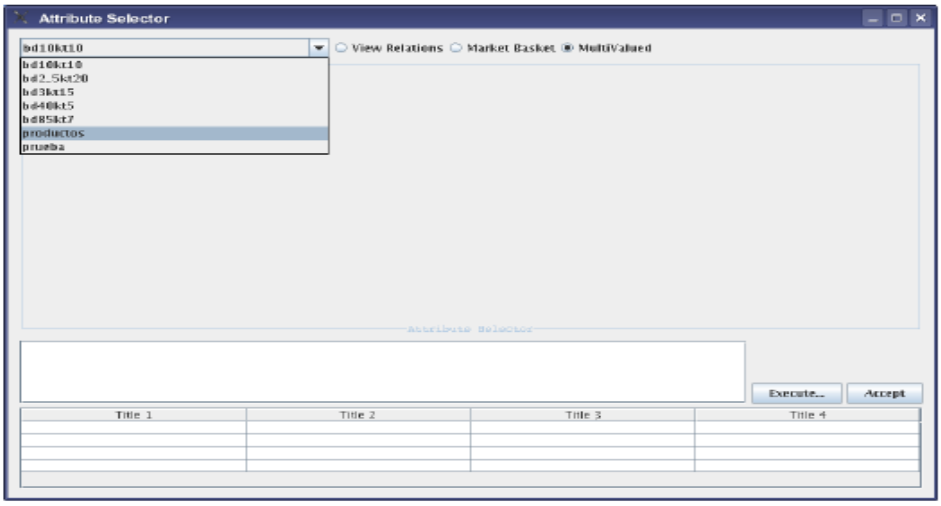
Un ejemplo de la construcción de una conexión a una base de datos PostgreSQL llamada minería a través del usuario ”nceron” con los valores por defecto para el Servidor y el Puerto será la siguiente:

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

La *interfase Connection* obtenida al pulsar el botón Accept de la Clase *connectionWizard* devuelve una instancia de esta conexión al *DBConnectionIcon* que se almacena en 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 4.32 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 una consulta de manera visual usando la metodología *Drag’n Drop*.

Figura 4.32: Implementación de la Clase *DBConnectionIcon3*



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 *interfase 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.

Figura 4.33: Función *getTables*

```
  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;
  }
}
```

A partir de las tablas que despliegue el *JComboBox* se puede escoger de esa lista la(s) tabla(s) que queremos relacionar en el análisis. La figura 4.34 muestra esta acción. Estas aparecerán dentro del área del *Attribute* Selector, una instancia de la Clase *MyCanvasTable*, donde se puede interactuar con las tablas de forma gráfica a y establecer relaciones al tiempo que se construye 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 4.35.

Figura 4.34: Tablas de la conexión organizadas en un *JComboBox*

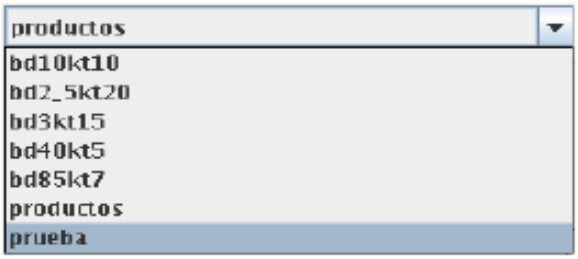
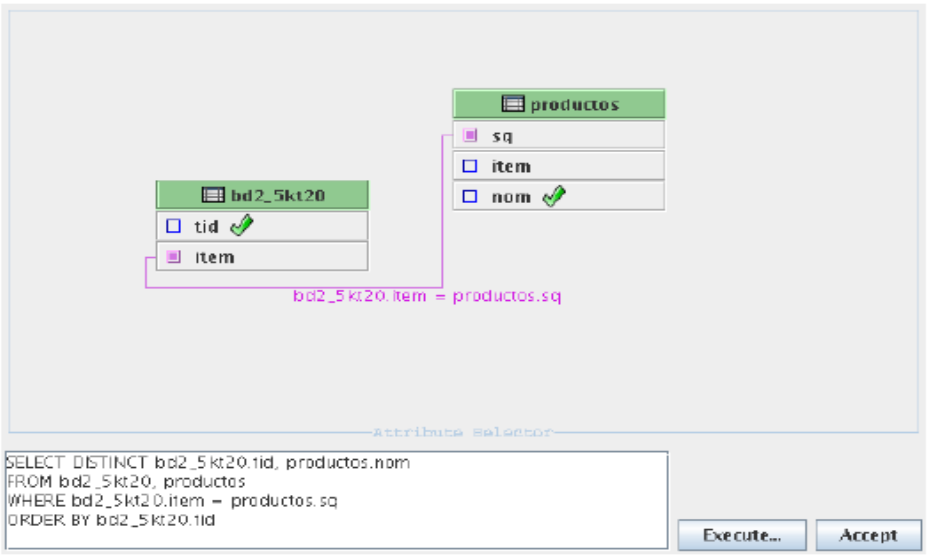


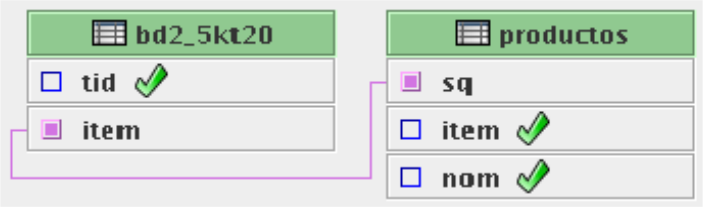
Figura 4.35: Implementación de la Clase *MyCanvasTable*



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 4.37.

El conjunto de objetos *Attribute* conforman un objeto *Table* que puede estar conformado por un número *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 4.36.

Figura 4.36: 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 iconos y sus relaciones. La figura 4.37 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 4.37.

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 interfases *Statament y ResulSet*. La *interfase Statement* se crea a partir de la interfase *Connection* que llega como parámetro en el constructor de la clase. Esta interfase, 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 construyó como parámetro al constructor de esta clase. El fragmento de código ilustra este procedimiento:

Figura 4.37: Construcción de la Sentencia SQL

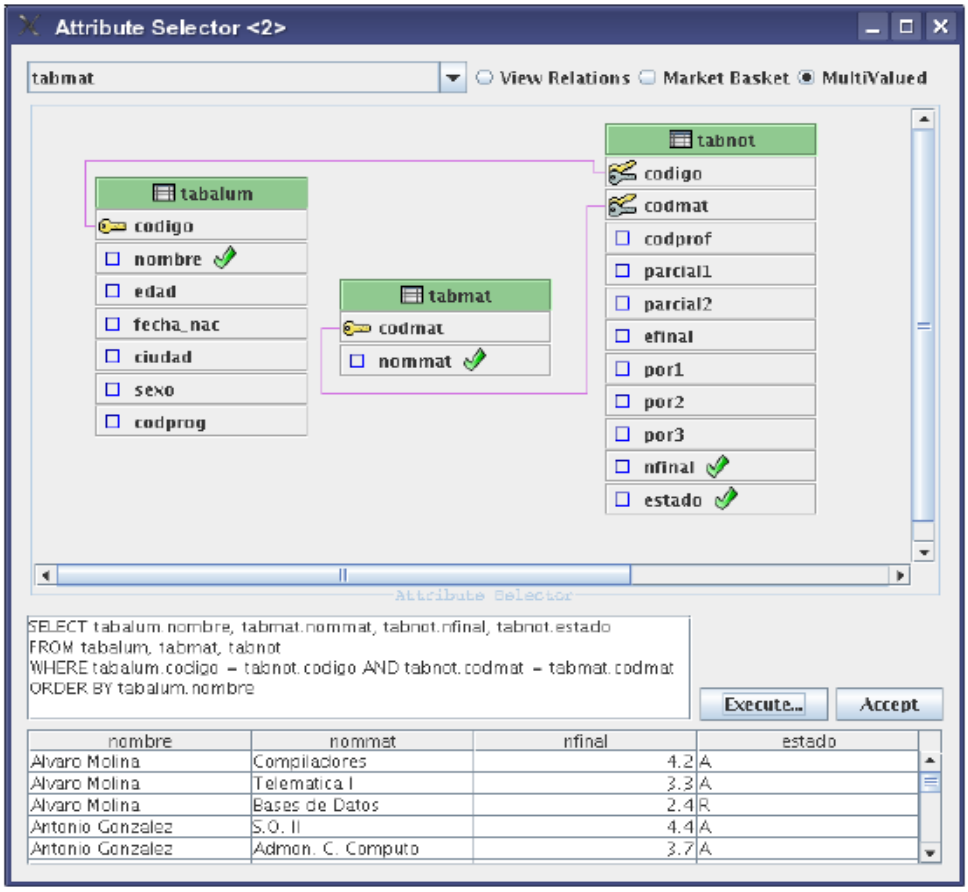
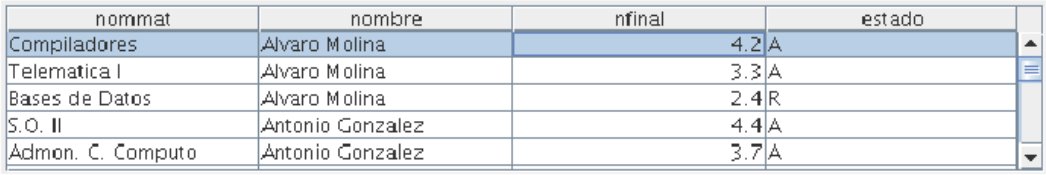


Figura 4.38: Función *executeQuery(String query)*

```
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);
}
```

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 4.39.

Figura 4.39: Construcción de la previsualización de datos en un JTable



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

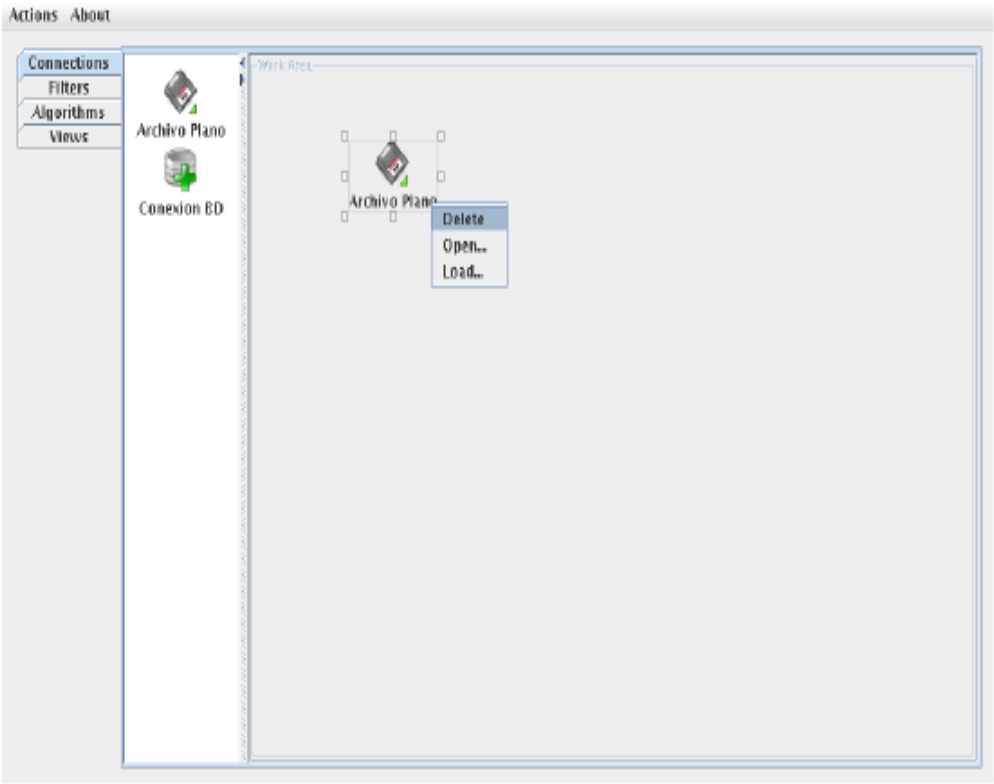
Una vez validados los datos que se quiere minar se confirma la aceptación al pulsar el botón *Accept*. En ese momento devolverá la interfaz principal y el ultimo paso para construir el conjunto de datos se realiza a través de la opción *Start Loading* del menú contextual de la clase *DBConnectionIcon* donde, según sea el caso, se invocaría lo métodos *loadMarketBasketDataSet*(), para cargar una instancia del *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 conforman este paquete son: *fileIcon, Openfile y fileTableModel.*

En la figura 4.40 se pueden ver las funciones que esta clase implementa.

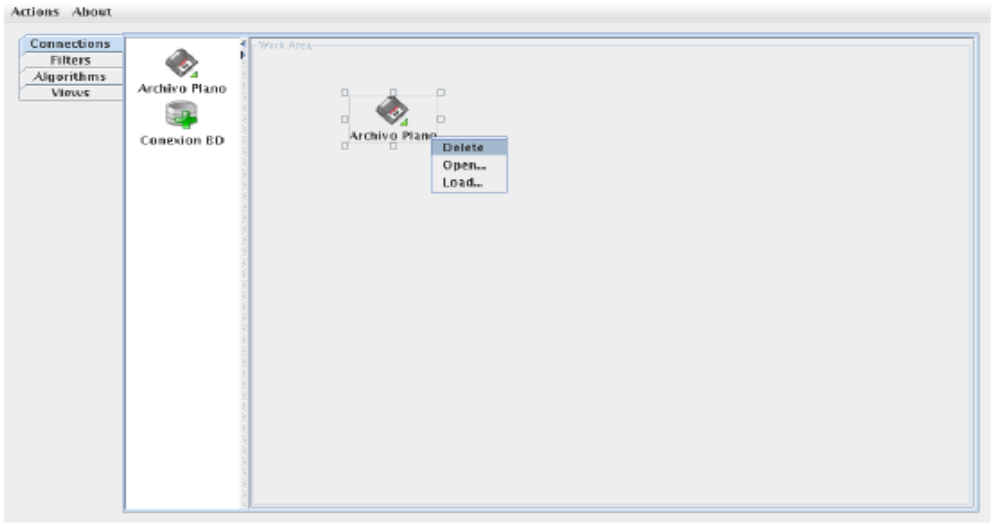
fileIcon extiende a la clase *Icon* que hace parte del paquete *KnowledgeFlow* y a su vez del paquete GUI, por tanto automáticamente hereda su menú con la opción por defecto *delete*, la cual permite al usuario eliminar el icono de conexión al Archivo Plano. Por otra parte, la clase *fileIcon* añade 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 como una cadena de texto la ruta completa en donde se encuentra el archivo.

Figura 4.40: Funciones de *fileIcon*



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 4.41.

Figura 4.41: Visualización del conjunto de datos



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*, quien 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 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 el código fuente, la clase *fileTableModel*.

Figura 4.42: 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];
  }
}
```

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.

4.3.5 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, vacíos, 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, se obtiene 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 mismo filtro *RemoveMissing* formato.

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

Filtro RemoveMissing El objetivo específico de este filtro, es eliminar todas las transacciones que contengan campos vacíos. 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 vacíos, 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 la figura 4.43, de la función encargada de hacer la transformación.

Figura 4.43: Pseudo código del filtro *RemoveMissing*

```
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
        }
    }
  }
```

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 vacíos, en las transacciones 2,4,6,7,8,9,10,11,12,14, como se muestra en la figura 4.44.

Figura 4.44: Datos de entrada antes de aplicar *RemoveMissing*

	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

Al aplicar el filtro *RemoveMissing*, se eliminaran las transacciones que contengan

atributos vacíos, las cuales son las siguientes: 2,4,6,7,8,9,10,11,12,14, dejando solo las transacciones 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 figura 4.45.

Figura 4.45: Datos de Salida después de aplicar *RemoveMissing*

	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

Filtro UpdateMissing El objetivo específico de este filtro, es remplazar los campos vacíos 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 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 vacíos de un atributo, con un valor otorgado por el analista, creando de este modo un nuevo conjunto de datos, libre de campos vacíos y con el mismo numero de transacciones originales. Se presenta la figura 4.46 de la función encargada de hacer esta transformación.

Figura 4.46: Pseudo código del filtro *UpdateMissing*

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

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 vacíos en las transacciones 7,8,11,12 como se muestra en la figura 4.47.

Al aplicar el filtro *UpdateMissing*, se remplaza en el atributo ”ALERGIA ANTIBIOTICO”, todos los campos vacíos, 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 4.48.

Figura 4.47: 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		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 4.48: Datos de Salida después 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

Filtro Selection El objetivo específico 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 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 la figura 4.49, de la función encargada de hacer la selección.

Figura 4.49: Pseudo código del filtro *Selection*

```
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]);
  }
}
```

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 módulo también brinda información sobre el numero de atributos que se eliminaron y el número 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, antes de aplicar el filtro *Selection*, en la figura 4.50. 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 se hizo la selección, los cuales son: AZUCAR SANGRE, ALERGIA ANTIBIOTICO y al final de la tabla como atributo objetivo ADMINISTRAR FARMACO.

Figura 4.50: Datos de Entrada antes de aplicar *Selection*

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 4.51: Datos de Salida después de aplicar *Selection*

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

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.*
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 pseudocódigo y las técnicas encargadas de hacer el muestreo.

Técnica 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 función se presenta en la figura 4.52.

Técnica de 1 en n Esta técnica se encarga de seleccionar una muestra, que tomará

cada transacción en saltos de *n* en *n, n* es el valor de salto otorgado por el analista.

Técnica de Primeros n Esta técnica se encarga de seleccionar una muestra, a partir de las primeras **n** transacciones del conjunto de datos de entrada.

La estructura de esta función se presenta en la figura 4.53.

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 módulo también brinda información sobre el número de Transacciones que se eliminaron y el número de transacciones que permanecen en el conjunto de datos final.

Figura 4.52: Pseudo código de la Técnica de Aleatorios

```
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
  }
}
```

Figura 4.53: Pseudo código de la Técnica de Primeros *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
    }
  }
```

Ejemplo de funcionamiento del filtro *Range*: Se presenta la tabla con los datos de entrada en la figura 4.54, antes de aplicar el filtro *Range*. En la tabla original se puede observar 14 Transacciones.

Figura 4.54: Datos de Entrada antes de aplicar las Técnicas de Muestreo

	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

En primera instancia, se aplica la técnica de Aleatorios, escogiendo un tamaño de muestra de 5 transacciones. Se puede observar que las transacciones elegidas por este método son 5, reduciendo el conjunto original en 9 transacciones. Como se muestra en la figura 4.55.

Figura 4.55: Datos de Salida después de aplicar la Técnica de Aleatorios

	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

Ahora se aplica la técnica de 1 en n, escogiendo como salto de muestra el valor 3. Se puede observar que las transacciones elegidas por este método se eligen a partir de la primera transacción en saltos de 3 en 3, reduciendo el conjunto original.
Como se muestra en la figura 4.56.

Figura 4.56: Datos de Salida después de aplicar la Técnica de 1 en *n*

	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

Ahora se aplica la técnica de Primeros n, escogiendo como Tamaño de muestra el valor 7. Se puede 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 4.57.

Figura 4.57: Datos de Salida después de aplicar la Técnica de primeros *n*

	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

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.

Técnica de Reducción 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 la figura 4.58.

Figura 4.58: Pseudo código de la Técnica de Reducción por Rango

```
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
    }
  }
}
```

Técnica de Reducción 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 la figura 4.59.

Figura 4.59: Pseudo código de la Técnica de Reducción por Atributo

```
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
     }
   }//////////
}
```

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 u de transacciones original. también se encarga de mostrar el nuevo conjunto de datos, con las transacciones que permanecieron después de aplicar la reducción. Este módulo también brinda información sobre el número de Transacciones que se

eliminaron y el número de Transacciones que permanecen en el conjunto de datos final. Ejemplo de funcionamiento del filtro *Reduction*: Se presenta la tabla con los datos de entrada, ates de aplicar el filtro *Reduction*, en la figura 4.60. En la tabla original se puede observar 14 transacciones.

Figura 4.60: Datos de Entrada antes de aplicar las Técnicas de Reducción

	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

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 puede 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 4.61.

Figura 4.61: Reducción por rango, Manteniendo los datos

	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

Ahora se aplica la misma técnica de Reducción por rango, pero como parámetro de selección, se remueve dicho rango. Se escoge la transacción 5 como límite 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 4.62.

Figura 4.62: Reducción por rango, Removiendo los datos

	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

Se aplica ahora la técnica de Reducción de Transacciones por Atributo, con valores alfabéticos, y se escoge como atributo de reducción a ”INDICE COLESTEROL”, y como valor de este atributo a ”ALTO”. también se escoge como parámetro de selección, Mantener el conjunto. Esto significa que el filtro buscará cualquier valor diferente a ”ALTO” en este atributo, y eliminará su transacción correspondiente. Se puede 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 4.63.

Figura 4.63: Reducción por atributo Alfabético, Manteniendo los datos

	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

Ahora se aplica la misma técnica de Reducción de Transacciones por Atributo, con valores alfabéticos, y se escoge como atributo de reducción a ”INDICE COLESTEROL”, y como valor de este atributo a ”ALTO”. Pero ahora se escoge como parámetro de selección, Remover el conjunto. Esto significa que el filtro buscará los valores del atributo iguales a ”ALTO” y eliminará su transacción correspondiente. Se puede 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 4.64.

Figura 4.64: Reducción por atributo Alfabético Removiendo los datos

	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

Se aplica ahora la técnica de Reducción de Transacciones por Atributo, con valores numéricos, y se escoge como atributo de reducción a ”ADMINISTRAR FARMACO”, y como limite restrictivo superior, al valor 5. También se escoge como parámetro de selección, Mantener el conjunto. Esto significa que el filtro eliminará todas las transacciones, con valores superiores o iguales a 5 en este atributo. Se puede 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 4.65.

Figura 4.65: Reducción por atributo Numérico Manteniendo los datos

	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

Ahora se aplica la misma técnica de Reducción de Transacciones por Atributo, con valores numéricos, y se escoge como atributo de reducción a ”ADMINISTRAR FARMACO”, y como limite restrictivo superior, al valor 5. Pero ahora, se escoge como parámetro de selección, Remover el conjunto. Esto significa que el filtro eliminará todas las transacciones, con valores inferiores a 5 en este atributo. Se puede 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 4.66.

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 *ShowCodification* y el núcleo principal la clase *Codification*.

Figura 4.66: Reducción por atributo Numérico Removiendo los datos

	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

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 la figura 4.67.

Figura 4.67: Pseudo código de Codification

```
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);
 }
```

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 4.68.

Figura 4.68: Reducción por atributo Numérico Removiendo los datos

	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

Después de aplicar el filtro *codification* al conjunto de entrada, la tabla codificada queda como se muestra en la figura 4.69.

Figura 4.69: Datos de Salida, después de aplicar la Codificación

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

Paralelamente se crea el diccionario de datos, para la posterior decodificación, como se muestra en la figura 4.70

Figura 4.70: Diccionario de datos

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

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.*

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.

ReplaceValue *ReplaceValue* 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 la figura 4.71.

Figura 4.71: Pseudo código de *ReplaceValue*

```
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;
  }
}
```

ShowReplaceValue *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 figura 4.72.

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 4.73.

Figura 4.72: Datos de Entrada, antes de aplicar el filtro *RemplaceValue*

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 4.73: Datos de Salida, después 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

Filtro NumericRange El objetivo específico de este filtro, es eliminar los valores de un atributo numérico, que están 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 están por fuera de un rango determinado, reemplazándolos con valores nulos. Se presenta la estructura de la función encargada de hacer la transformación, en la figura 4.74.

Figura 4.74: Pseudo código de *NumericRange*

```
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
  }
}
```

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 4.75

Figura 4.75: Datos de Entrada, antes de aplicar el filtro 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

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 en la figura 4.76.

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.*

Figura 4.76: Datos de Salida, después de aplicar el filtro *NumericRange*

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	

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 cules 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 discretización, 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 aplicación de sus técnicas en la figura 4.77.

Técnica 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 la figura 4.78.

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 mínimo valor, y desde el máximo valor hasta el + infinito, como se muestra en la figura 4.79.

Figura 4.77: Argumentos de las variables

```
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);
   }
}
```

Figura 4.78: Pseudo código de discretización con número de rangos:

```
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;
      }
    }
}
```

Figura 4.79: Pseudo código de discretización con el tamaño del rango

```
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;
    }
  }
}
```

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

Ejemplo de funcionamiento del filtro *Discretize:*
Se presenta la tabla con los datos de entrada, ates de aplicar el filtro *Discretize* en la figura 4.80.

En primera instancia se aplica la técnica de discretizar con número de rangos, seleccionamos al atributo numérico ”ADMINISTRAR FARMACO”, y se segmenta el 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 figura 4.81.

Figura 4.80: Datos de Entrada, antes de aplicar el filtro *Discretize*

	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 4.81: Datos de Salida, antes de aplicar el filtro *Discretize* con Numero de Rangos

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]

Ahora, se aplica la técnica de discretizar con el tamaño del rango, se selecciona el atributo numérico ”ADMINISTRAR FARMACO”, y se escoge como tamaño del rango el valor 3 para obtener 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 figura 4.82.

Figura 4.82: Datos de Salida, antes de aplicar el filtro Discretize con el tamaño del rango.

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]

5. PRUEBAS Y RESULTADOS

5.1 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 período determinado. El conjunto de datos contiene 10.757 diferentes productos. Los conjuntos de datos minados con la herramienta TariyKDD se muestran en el Tabla 5.1.

Tabla 5.1: 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

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.

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 5.1, 5.2 y 5.3, 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.

Tabla 5.2: 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 5.1: Rendimiento BD85KT7

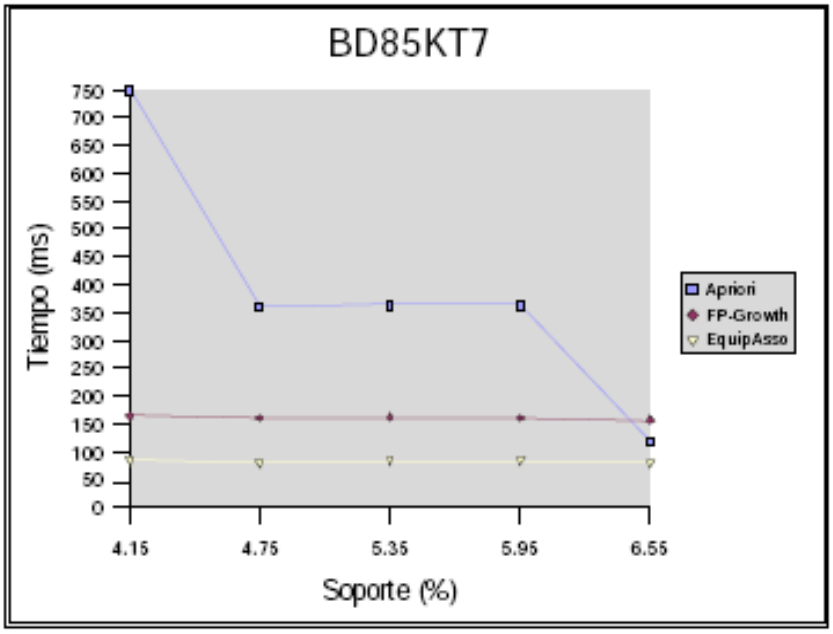


Tabla 5.3: Tiempos de ejecución tabla BD40KT5

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 5.2: Rendimiento BD40KT5

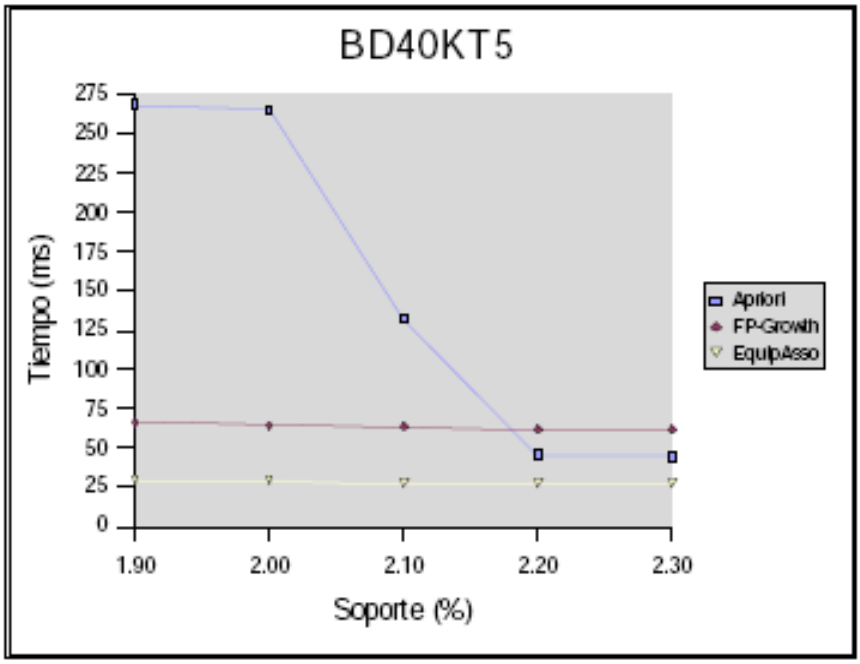
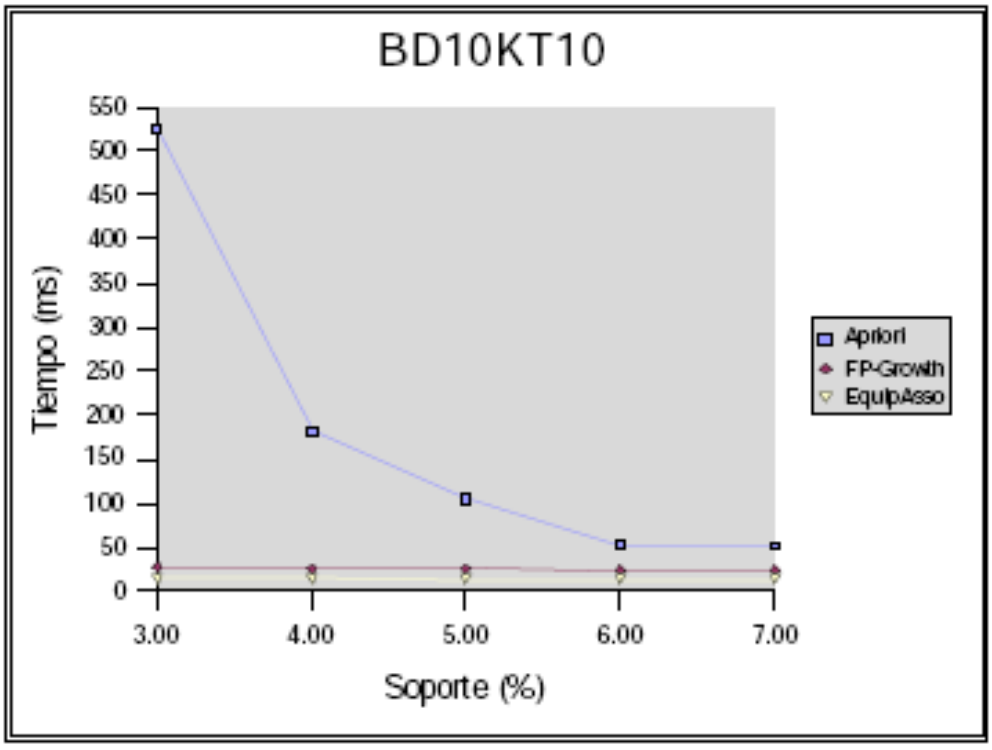


Tabla 5.4: Tiempos de ejecución tabla BD10KT10

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 5.3: Rendimiento BD10KT10



Analizando el tiempo de ejecución de únicamente los algoritmos *FP-Growth* y *EquipAsso* (figuras 5.4, 5.5 y 5.6) para los conjuntos de datos BD85KT7, BD40KT5 y BD10KT10. Con soportes más bajos, el comportamiento de estos algoritmos sigue siendo similar.

Tabla 5.5: 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 5.4: Rendimiento BD85KT7

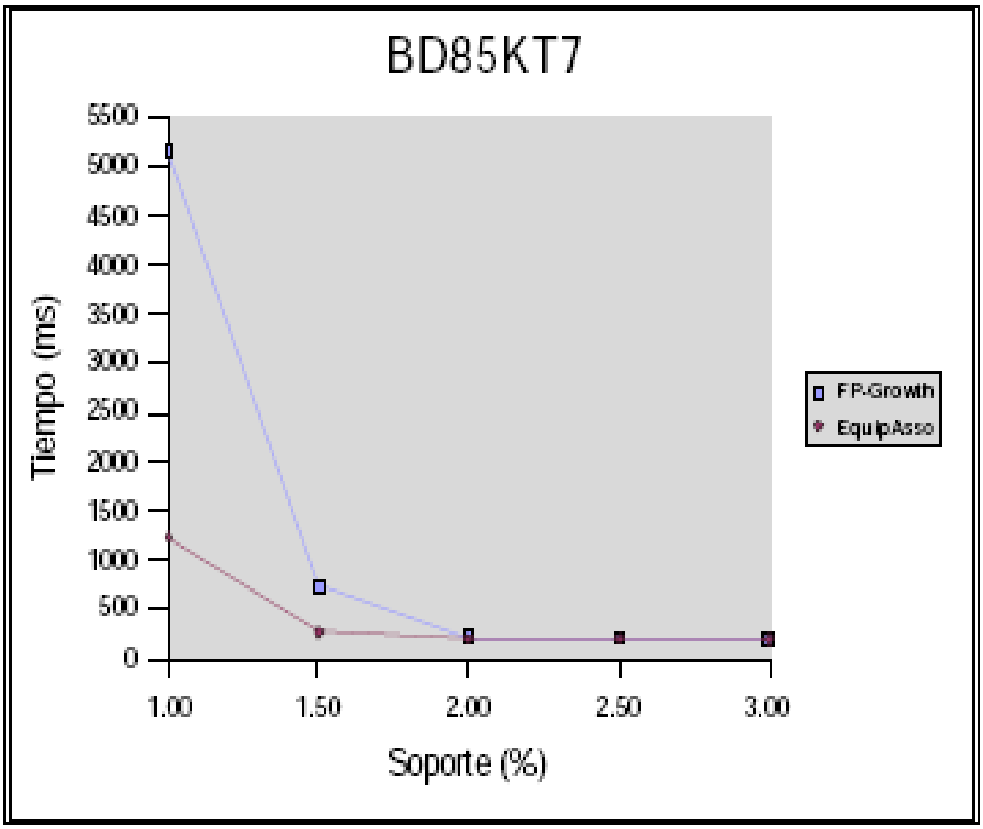


Tabla 5.6: Tiempos de ejecución tabla BD40KT5

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 5.5: Rendimiento BD40KT5

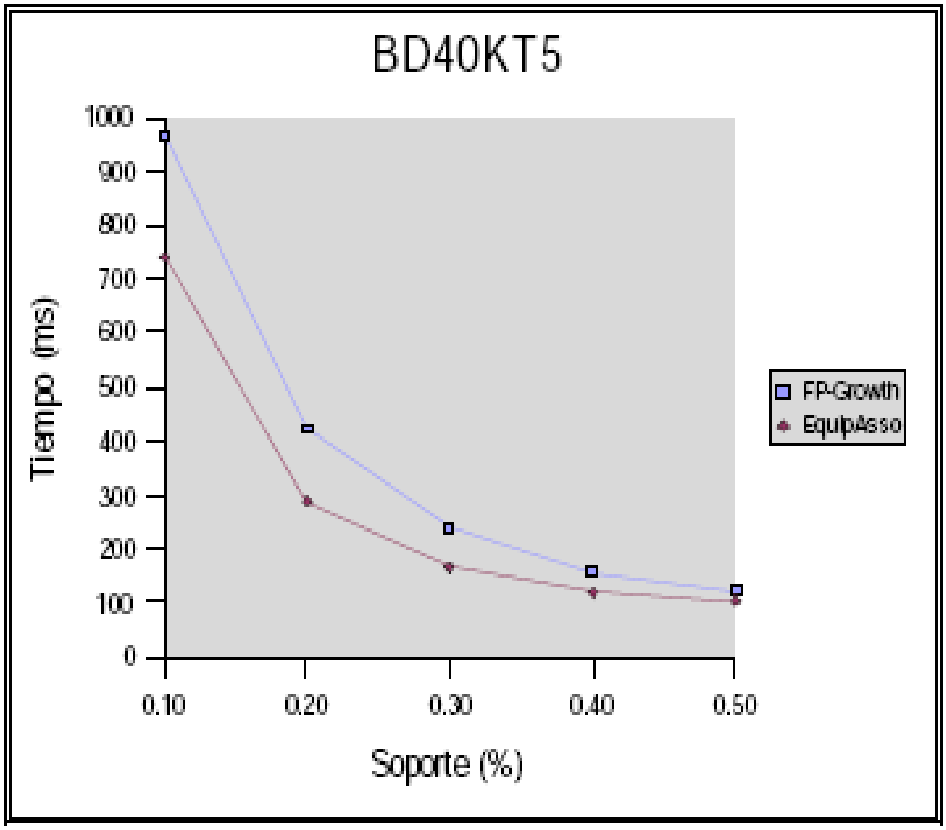
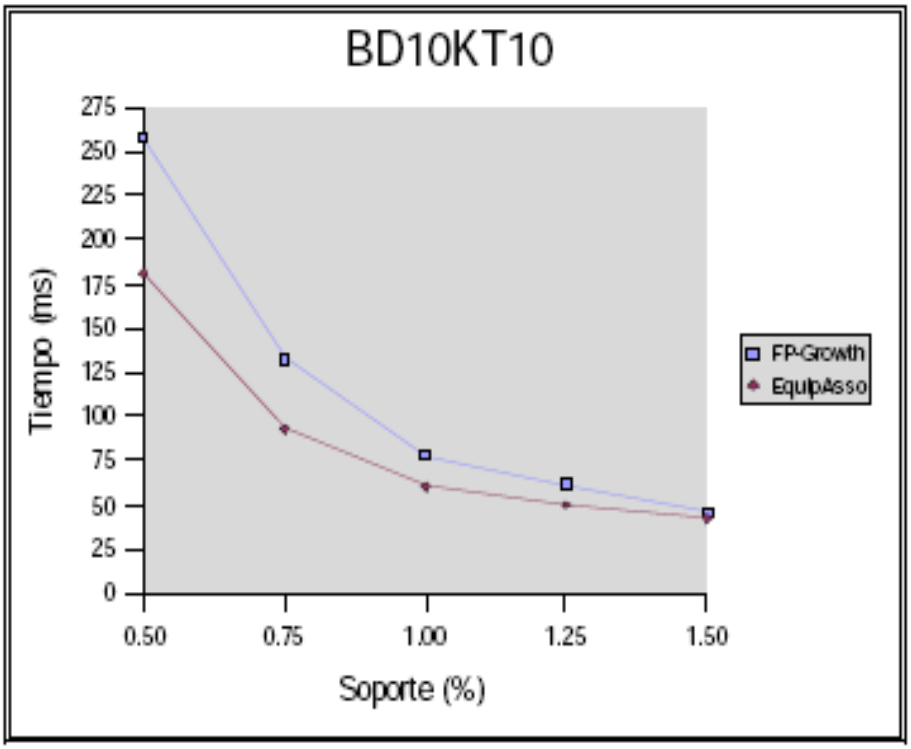


Tabla 5.7: Tiempos de ejecución tabla BD10KT10

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 5.6: Rendimiento BD10KT10



5.2 RENDIMIENTO ALGORITMOS DE CLASIFICACION

Para realizar las pruebas de clasificación se seleccionó la base de datos histórica de los estudiantes de la Universidad de Nariño, compuesta por información personal y académica de 20328 estudiantes. Este conjunto de datos fue construido por miembros del Grupo de Investigación GRiAS de la Universidad de Nariño. Mayores detalles de este conjunto pueden ser consultados en [27].

Después de los procesos de limpieza y transformación de datos se trabajó con una

única tabla o vista minable que contó con un total de 26 atributos que se mencionan en el Tabla 5.8. listando sus posibles valores.

Tabla 5.8. Campos en el conjunto UDENAR

Atributo	Valores
edad	A, B, C, D
edading	A, B, C, D
fecha_ing	A, B, C, D, E
ingresos	A, B, C, D, E
val_matricula	A, B, C, D, E
nestrato	0, 1, 2, 3, 4, 5, 6
naturaleza	Oficial, Privado
jornada	Completa, Manana, Tarde, Noche
calendario	A, B, Flexible
sexo	M, F
ponderado	A, B, C, D
ocu_madre	1, 2, 3, 4, 5
Mas uno a cargo	S, N
vive_con_familia	S, N
tiene_hermanos_u	S, N
tipo_residencia	0, 1, 2, 3
zona_nac	Norte, Norten, Occidente, Occidenten, Oriente, Orienten, Sur, Surn
semestred	1, 2, 3, 4, 5
cod_facultad2	1, 2, 3, 4, 5, 6, 7, 8, 18, 22, 32
claseal	1, 2, 3

Atributo	Valores
claserend	1, 2, 3, 4, 5
clasepromedio	A, B, C, D, E
especial mas_una_a_cargo	S, N
padre_vive	S, N
estado_civil	0, 1, 2, 3, 4
madre_vive	S, N

Como se ve en la tabla varios atributos fueron discretizados. Los respectivos valores y detalle de este proceso se explican en los Tablas 5.9. a 5.24 respectivamente.

Tabla 5.9. Discretización atributo Edad

EDAD	DISCRETIZACIÓN	CANTIDAD
Menores e iguales a 18	A	827
Mayores de 18 y menores que 22	B	3634
Mayores e iguales que 22 y Menores de 26	C	4856
Mayores e iguales que 26	D	11012

Tabla 5.10. Discretización atributo Edad_ingreso

EDAD DE INGRESO	DISCRETIZACIÓN	CANTIDAD
Menores e iguales a 18	A	9054
Mayores de 18 y menores que 22	B	7370
Mayores e iguales que 22 y Menores de 26	C	2594
Mayores e iguales que 26	D	1311

Tabla 5.11. Discretización atributo Fecha ingreso

FECHA DE INGRESO	DISCRETIZACIÓN	CANTIDAD
Antes de 1990	A	1022
Después o igual a 1990 y Menores de 1995	B	4852
Después o igual a 1995 y Menores de 2000	C	5978
Después o igual al 2000 y Menores de 2003	D	5046
Mayores o iguales de 2003	E	7621

Tabla 5.12. Discretización atributo Ingresos

INGRESOS	DISCRETIZACIÓN	CANTIDAD
Menores que 3	A	6101
Mayores o iguales a 3 y menores que 6	B	6350
Mayores que 6 y menores que 9	C	3325
Mayores que 9 y menores que 12	D	1828
Mayores de 12	E	2725

Tabla 5.13. Discretización atributo Valor Matricula

VALOR DE MATRICULA	DISCRETIZACIÓN	CANTIDAD
Menores de 50000	A	3529
Mayores o iguales que 50000 y menores que 100000	B	5348
Mayores o iguales que 100000 y menores que 200000	C	6762
Mayores o iguales que 200000 y menores que 500000	D	3636
Mayores o iguales que 500000	E	1054

Tabla 5.14. Discretización atributo Naturaleza

NATURALEZA	DISCRETIZACIÓN	CANTIDAD
Oficial	Oficial	13571
No Oficial	Privado	6757

Tabla 5.15. Discretización atributo Ocupación Madre

OCUPACION MADRE	DISCRETIZACIÓN	CANTIDAD
Ama de Casa	1	9049
Profesional	2	572
Empleada	3	313
Independiente	4	200
Otro	5	10194

Tabla 5.16. Discretización atributo Ponderado

PONDERADO	DISCRETIZACIÓN	CANTIDAD
Menores que 30	A	43
Mayores o iguales que 30 y menores que 50	B	2689
Mayores o iguales que 50 y menores que 70	C	16601
Mayores o iguales que 70 y menores o iguales que100	D	1623

Tabla 5.17. Discretización atributo Estado Civil

ESTADO CIVIL	DISCRETIZACIÓN	CANTIDAD
Soltero	0	20651
Casado	1	5
Separado	2	251
Divorciado	3	24
Unión Libre	4	25

Tabla 5.18. Discretización atributo Tipo Residencia

TIPO_RESIDENCIA	DISCRETIZACIÓN	CANTIDAD
No propia	0	5181
Propia	1	14995
Propia pagando cuotas	2	308
Arrendando o Anticresis	3	472

Tabla 5.19. Discretización atributo Semestre

SEMESTE	DISCRETIZACIÓN	CANTIDAD
Semestre 1 hasta semestre 4	1	8859
Semestre 5 hasta semestre 8	2	3539
Semestre 9 hasta semestre 10	3	1390
Egresados	4	1383
Graduados	5	5157

Tabla 5.20. Discretización atributo Facultad

FACULTAD	DISCRETIZACIÓN	CANTIDAD
Artes	1	2097
Ciencias Agrícolas	2	1532
Derecho	3	1208
Ciencias Económicas y Administrativas	4	2424
Ingeniería	5	2549
Ciencias Pecuarias	6	1715
Ciencias Naturales y Matemáticas	7	3701
Ciencias Humanas	8	4009
Educación	18	793
Ingeniería Agroindustrial	22	537
Ciencias de la Salud	32	390

Tabla 5.21. Discretización atributo Zona_nac

ZONA_NAC	DISCRETIZACIÓN	CANTIDAD
Norte del departamento de Nariño	Norten	66
Sur del departamento de Nariño	Surn	16931
Oriente del departamento de Nariño	Orienten	1512
Occidente del departamento de Nariño	Occidentenn	595
Norte de Colombia	Norte	62
Sur de Colombia	Sur	410
Oriente de Colombia	Oriente	23
Occidente de Colombia	Occidente	729

Tabla 5.22. Discretización atributo Clase_al

CLASE_AL	DISCRETIZACIÓN	CANTIDAD
Normal	1	18641
Reingreso	2	846
Retirado	3	1469

Tabla 5.23. Discretización atributo Clase_rend

CLASE_REND	DISCRETIZACIÓN	CANTIDAD
No ha perdido materias	1	7852
Ha perdido 1 materia	2	3339
Ha perdido 2 materia	3	2576
Ha perdido 3 materia	4	2047
Ha perdido más de 3 materias	5	5142

Tabla 5.24. Discretización atributo Clase_promedio

CLASE_PROMEDIO	DISCRETIZACIÓN	CANTIDAD
Menor a 2	A	2391
Mayor o igual a 2 hasta 3	B	2934
Mayor o igual a 3 hasta 3.5	C	5166
Mayor o igual a 3.5 hasta 4.0	D	6850
Mayor o igual a 4.0 hasta 5.0	E	3615

Se hicieron dos tipos de pruebas sobre el conjunto UDENAR evaluando los tiempos de ejecución de los algoritmos de clasificación implementados en TariyKDD (C4.5 y Mate). Las primeras pruebas variaban el número de atributos con el que se trabajaba y en las segundas se varió el número de registros con el que se construyó el modelo.

En la primera prueba se construyeron modelos seleccionando en primera instancia 22 atributos y el total de registros (20328), la clase objetivo con la que se evaluó el modelo fue *clase_rendimiento*. Posteriormente, se eliminó uno a uno los atributos hasta un limite de 4 columnas.

Objetivo de la prueba, evaluar el rendimiento de los clasificadores al disminuir el número de columnas del conjunto de datos. Los resultados de esta prueba se pueden ver en la Figura 5.7.

La ultima prueba sobre los atributos de la tabla consistió en eliminar en cada prueba el atributo ganador en cada experimento para determinar los atributos más relevantes. Objetivo de la prueba, determinar los atributos con mayor fuerza clasificatoria, lo cual significa determinar los atributos con mayor ganancia. Los resultados de esta prueba se resumen en la Figura 5.8.

Tabla 5.25. Tiempos de ejecución en el conjunto UDENAR al reducir atributos.

Número de Atributos	Tiempo de Ejecución (ms)	
	C4.5	Mate
22	1.495,67	7.157,33
21	1.670,33	6.931,33
20	1.701,67	7.700,33
19	1.441,33	7.143,33
18	1.445,67	6.461,67
17	1.327,33	6.990,00
16	1.195,67	6.211,67
15	1.139,00	5.969,33
14	1.083,33	5.900,67
13	1.080,67	5.986,67
12	916,00	5.827,67
11	854,00	5.044,67
10	696,33	4.530,33
9	609,00	4.786,33
8	494,33	4.931,67
7	429,67	3.922,00
6	353,00	4.091,33
5	237,33	3.713,00
4	157,67	2.338,67

Figura 5.7. Rendimiento de algoritmos en el conjunto UDENAR al reducir atributos.

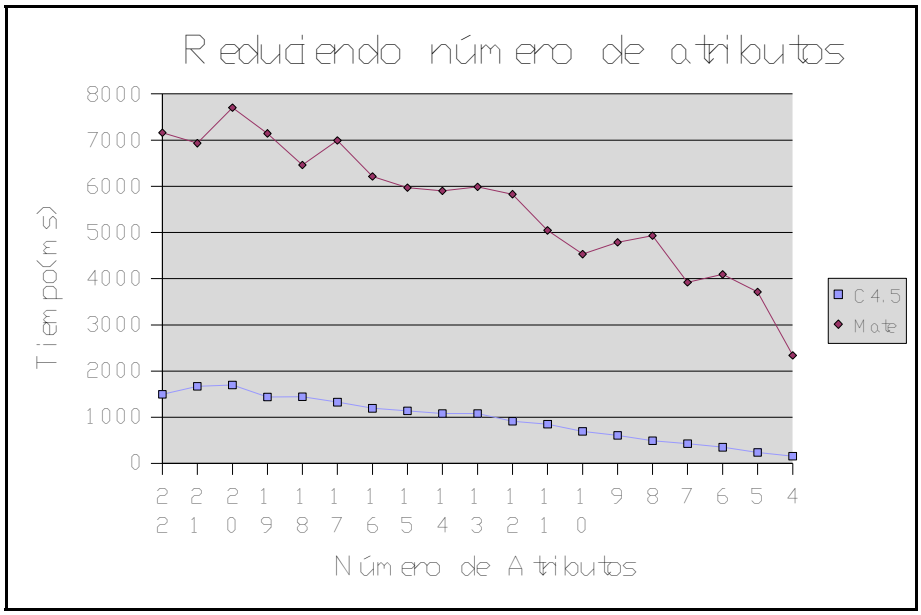
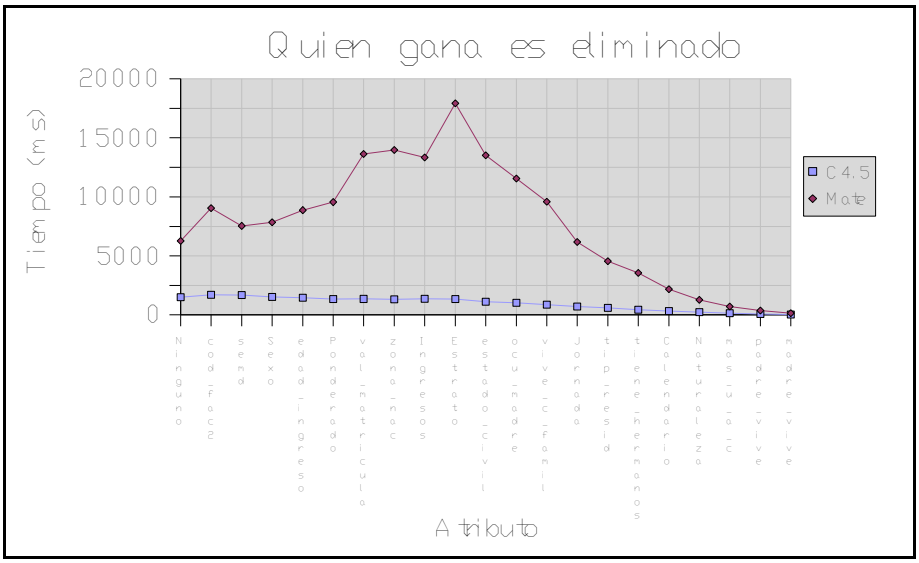


Tabla 5.26. Tiempos de ejecución en el conjunto UDENAR al eliminar el atributo ganador.

Atributo	Tiempo de Ejecución (ms)		Número de Reglas
	C4.5	Mate	
Ninguno	1.495,00	6.276,00	429
cod_fac2	1.697,00	9.040,00	398
semd	1.688,33	7.541,67	417
Sexo	1.515,67	7.851,67	358
edad_ingreso	1.448,67	8.875,67	410
Ponderado	1.337,33	9.555,33	389
val_matricula	1.356,67	13.634,33	381
zona_nac	1.320,67	13.965,67	384
Ingresos	1.372,00	13.327,00	293
Estrato	1.350,33	17.925,67	219
estado_civil	1.121,33	13.524,67	209
ocu_madre	1.026,67	11.557,00	158
vive_c_famil	867,33	9.587,67	121
Jornada	722,67	6.181,00	59
tip_resid	598,67	4.552,67	34
tiene_hermanos	432,00	3.559,67	26
Calendario	331,00	2.186,00	16
Naturaleza	227,00	1.272,00	8
mas_u_a_c	137,33	700,00	5
padre_vive	79,33	366,33	3
madre_vive	31,33	153,33	2

Figura 5.8. Rendimiento de algoritmos en el conjunto UDENAR al eliminar el atributo ganador.

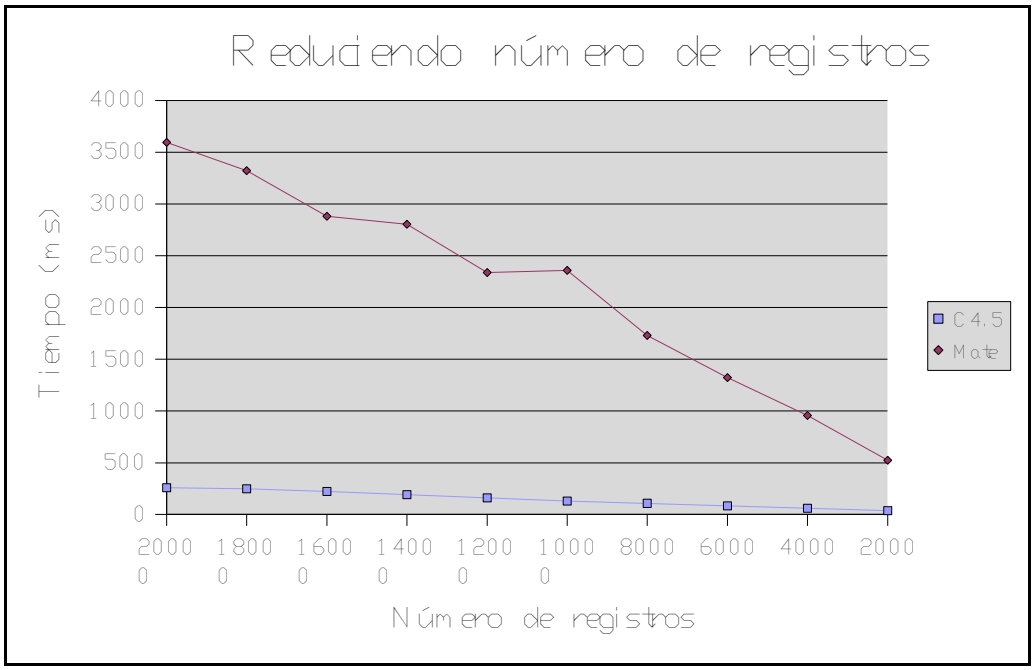


A partir de la última prueba se construyó una vista minable con los 5 atributos más relevantes (*cod_facultad, semestred, sexo, edad_ingreso, ponderado*) más la clase (*clase_rendimiento*) y se tomaron muestras desde 20000 hasta 2000 registros disminuyendo el tamaño del conjunto en 2000 registros para cada medición. Un resumen de la prueba se puede ver a continuacion en la Figura 5.9.

Tabla 5.27. Tiempos de ejecución en el conjunto UDENAR al reducir el número de registros.

Registros	Tiempo de Ejecución (ms)		Número de Reglas
	C4.5	Mate	
20000	258,33	3.595,67	226
18000	249,00	3.322,67	212
16000	222,33	2.880,67	216
14000	192,67	2.804,00	224
12000	160,33	2.338,33	228
10000	130,00	2.358,33	225
8000	108,33	1.730,67	213
6000	83,00	1.323,67	216
4000	60,00	957,00	200
2000	38,33	523,67	204

Figura 5.9. Rendimiento de algoritmos en el conjunto UDENAR al reducir el número de registros.



En el gráfico 5.7 se puede observar como al reducir el número de atributos bajan los tiempos de ejecución de los algoritmos siendo más notable para el algoritmo Mate.

Se puede observar también que la ausencia de ciertos atributos hace aumentar los tiempos de ejecución en ambos algoritmos determinando la relevancia de dicho atributo dentro del conjunto.

Analizando la figura 5.8 se encuentra un comportamiento especial que afecta en mayor medida al algoritmo Mate, a medida que se eliminan atributos relevantes esto afecta los tiempos de ejecución y resulta más difícil para los algoritmos construir modelos que definan a la clase con los atributos con los que dispone.

En la figura 5.9 se ve claramente como el número de registros afecta en mayor medida al algoritmo Mate y conforme se disminuye el tamaño del conjunto, se mejoran los tiempos de ejecución.

De todas maneras y en forma general se ve claramente un mejor comportamiento del algoritmo C4.5 con respecto a Mate.

6. CONCLUSIONES Y TRABAJOS FUTUROS

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 [42,43,44,45].

Para el desarrollo del proyecto se hizo un análisis de varias herramientas de software libre que abordan tareas similares a las que se pretendió 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.

Se incluyó en TariyKDD mejoras en la conexión a bases de datos implementando un interfaz que permite la selección visual de atributos y el establecimiento de relaciones entre diferentes tablas pertenecientes a una misma base de datos. Se implementó nuevos algoritmos de asociación y clasificación ausentes en otras herramientas como FPGrowth, EquipAsso y Mate. Igualmente se construyó visores que permiten la interacción directa con los resultados obtenidos a través de tablas ordenadas, arboles jerarquizados y gráficos de distribución.

Se establecieron patrones de diseño que hicieron 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, así como la implementación de un sistema de control de versiones que facilita el trabajo colaborativo y permite el acceso a la herramienta a través de Internet.

La construcción de TariyKDD comprendió el desarrollo de cuatro módulos que cubrieron:

El módulo de conexión a datos, tanto a archivos planos como a bases de datos relacionales.

El módulo de utilidades, que contiene una colección de clases y librerías comunes usadas en toda la aplicación.

El módulo de kernel, que incluye la etapas 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 Mate* para clasificación y el proceso de visualización de resultados, utilizando tablas y árboles para generar reportes de los resultados y reglas obtenidas.

Por último, el módulo de interfaz gráfica de usuario que provee una interacción amigable sobre los demás componentes por parte del usuario.

Se incluyó un módulo para la predicción de nuevos registros a partir de los modelos construidos con los algoritmos de clasificación.

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.

El desarrollo de TariyKDD fue logrado usando en su totalidad herramientas de código abierto y software libre.

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 las ventas de un supermercado de la Caja de Compensación Familiar de Nariño compuesta por 85.692 transacciones. Para Clasificación, se trabajó con la base de datos histórica de los estudiantes de la Universidad de Nariño, compuesta por información personal y académica de 20.328 estudiantes.

Como resultado de las pruebas para la tarea de clasificación se concluye que es recomendable el uso del algoritmo *C4.5* para conjuntos de datos extensos donde el número de atributos a analizar sea grande. Solo bajo conjuntos de datos pequeños y con reducido número de atributos resulta viable aplicar el algoritmo *Mate* bajo una arquitectura débilmente acoplada.

Como resultado de las pruebas para Asociación se concluye que los algoritmos *FPGrowth* y *EquipAsso* obtienen buenos tiempos de respuesta al ser aplicados en conjuntos grandes pero a medida que se disminuye el criterio de soporte se ve un mejor comportamiento por parte del algoritmo *EquipAsso*. No es viable aplicar el algoritmo *Apriori* bajo conjuntos grandes limitando su uso a muestras pequeñas de datos.

Este proyecto complementa el trabajo "*Proyecto TARIY – Un Análisis de Rendimiento de Algoritmos para Reglas de Asociación*" [36] que fue ganador de la VII Convocatoria Alberto Quijano Guerrero de proyectos de investigación estudiantil en la categoría de Investigación Cuantitativa presentado por los mismos autores.

Este proyecto hizo parte de los proyectos financiados por el Sistema de Investigaciones de la Universidad de Nariño dentro del marco del Concurso de Tesis de Pregrado organizado en el año 2006.

Durante el desarrollo del proyecto se participó activamente en eventos regionales como el Día Internacional del Software Libre y la Semana de Ingeniería.

Como resultado de este proyecto se publicó y presentó un artículo internacional en el marco del XXXII Congreso Latinoamericano de Estudios Informáticos - CLEI 2006 realizado en la ciudad de Santiago de Chile [40].

Se cuenta con una versión estable 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 realización de este proyecto, se sugiere una serie de recomendaciones como punto de partida para trabajos futuros:

- Realizar mayores pruebas de rendimiento de esta arquitectura con otros repositorios de datos reales.

- Implementar otras primitivas que Timaran propone para tareas de Asociación y Clasificación.

- Implementar otras tareas y algoritmos de minería de datos, como clustering y patrones secuenciales.

- Implementar nuevos filtros e interfaces de visualización que permitan el mejoramiento continuo de TariyKDD.

- Acoplar gráficos estadísticos a los conjuntos de datos cargados para obtener una información inicial de sus características.

- Acoplar TariyKDD fuertemente con PostgreKDD.

- Disponer de la herramienta como material de apoyo a las electivas de base de datos dentro del programa de Ingeniería de Sistemas.

- Liberar, compartir y difundir una versión estable de TariyKDD con la capacidad de descubrir conocimiento en bases de datos.

- Asegurar la continuidad del proyecto.

Finalmente, este trabajo permitió aplicar los conocimientos adquiridos en el programa de Ingeniería de Sistemas y en especial los de la electiva de bases de datos, así como el trabajo y aprendizaje dentro del Grupo de Investigación GRiAS Línea KDD.

BIBLIOGRAFIA

[1] 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.

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

[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 2Ao Conference KDD y Data Mining, Portland, Oregon, 1996.

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

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

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

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

[9] IBMCorporation. Intelligent miner. http://www4.ibm.com/software/data/iminer, 2001.

[10] 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.

[11] 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.

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

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

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

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

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

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

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

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

[27] Lombana C, Narvaez R, Viteri G, Dulce E. Detección de patrones de bajo rendimiento y/o deserción de los estudiantes de la universidad de nariño con técnicas de minería de datos, VIII Convocatoria Alberto Quijano Guerrero, Sistema de Investigaciones Universidad de Nariño, 2006.

[28] 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.

[29] 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.

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

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

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

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

[34] R. Rakotomalala. Tanagra project. Http: //chirouble. Univ-lyon2 .fr/ ricco/ tanagra/ en/ tanagra. html, 2006.

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

[36] I.Ramirez, A. Calderón, J, Alvarado and F. Guevara. Análisis de desempeño de EquipAsso: Proyecto TARIY – Un Análisis de Rendimiento de Algoritmos para Reglas de Asociación, VII Convocatoria Alberto Quijano Guerrero, Sistema de Investigaciones, Universidad de Nariño, 2005.

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

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

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

[40] R. Timarán, A. Calderón, I. Ramirez, J. Alvarado and F. Guevara. Análisis de desempeño de EquipAsso: Un algoritmo para el cálculo de *Itemsets* frecuentes basado en operadores algebraicos relacionales, XXXII Conferencia Latinoamericana de Estudios Informáticos CLEI, Santiago, Chile, 2006.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

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

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

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

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

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

[52] Waikato ML Group. Collections of datasets.
http://www.cs.waikato.ac.nz/ml/weka/indexd atasets.html.

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

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

ANEXOS

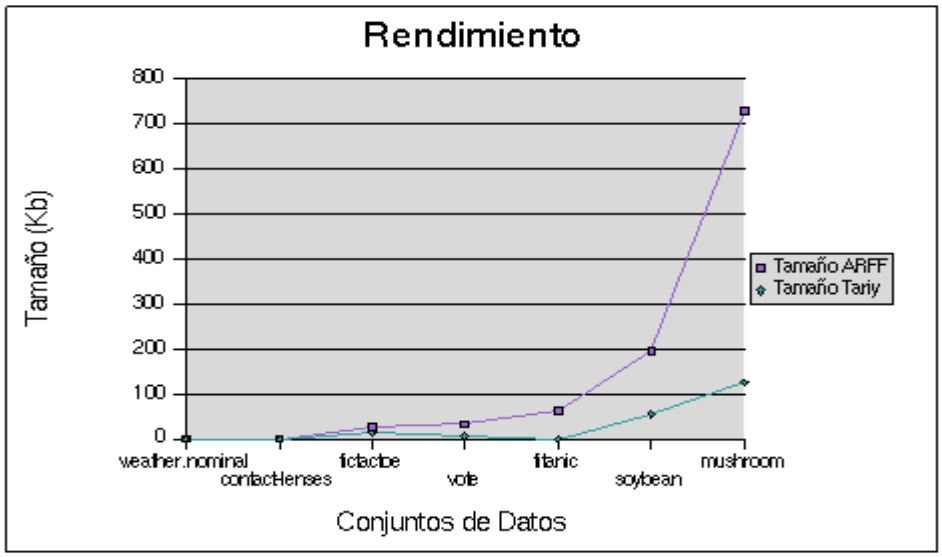
A. RENDIMIENTO FORMATO DE COMPRESIÓN TARIY vs FORMATO ARFF

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 Tabla, donde se registra el tamaño en disco de cada formato al almacenar conjuntos de datos con diferente número de transacciones y atributos.

Tabla 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



B. ARTICULO PUBLICADO EN LA XXXII CONFERENCIA LATINOAMERICANA DE ESTUDIOS INFORMÁTICOS CLEI 2006 EN LA CIUDAD DE SANTIAGO DE CHILE

Análisis de desempeño de EquipAsso: Un algoritmo para el cálculo de *Itemsets* frecuentes basado en operadores algebraicos relacionales.

Ricardo Timarán Pereira, Ph. D.*

ritimar@udenar.edu.co

Andrés O. Calderón Romero.*
and@mail.udenar.edu.co

Iván Ramírez Freyre.*
ivan.ramirezf@gmail.com

Juan Carlos Alvarado.*
endimeon777@yahoo.es

Fernando Guevara.*
drus3@latinmail.com

*Universidad de Nariño, Departamento de Ingeniería,

Ciudad Universitaria Torobajo

San Juan de Pasto, Colombia

2006

Abstract

The task of searching for interesting relationships among data has been always a researching focus in data mining. The overall performance of mining association rules is determined by discovering large itemsets, i.e., sets of itemsets that have their support above a pre-determined minimum support. The different algorithms proposed for association rules task show different approaches to generate all large itemsets: Apriori, AprioriTid, AprioriHybrid, DHP, DIC, Partition, FP-Growth and EquipAsso.

In this paper, the performance of EquipAsso, an algorithm for discovering large itemsets, based on two new operators of relational algebra, is evaluated in relation with Apriori and FP-Growth algorithms, on Tariy, a tool for the Association task loosely coupled with a DBMS.

Keywords: Data Mining, Algorithms for Association Task, Data Mining Tools, Performance Analisys.

Resumen

La tarea de generar asociaciones a partir de los datos ha sido siempre un foco de investigación en el área de Minería de Datos. El cálculo de conjuntos de ítems frecuentes es la etapa computacionalmente más costosa cuando se aborda la tarea de asociación. Los distintos algoritmos propuestos para implementarla se han concentrado en el cálculo de conjuntos frecuentes: Apriori, AprioriTid, Apriori Hibrido, DHP, DIC, Partition, FP-Growth y EquipAsso.

En este artículo, se evalúa el rendimiento del algoritmo EquipAsso, un algoritmo para el cálculo de conjuntos de ítems frecuentes, basado en dos nuevos operadores del álgebra relacional, con respecto a los algoritmos Apriori y FP-Growth en Tariy, una herramienta para la tarea de Asociación débilmente acoplada con un SGBD.

Palabras claves: Minería de Datos, Algoritmos para Asociación, Herramientas de Minería de Datos, Análisis de Rendimiento.

1. Introducción

El Descubrimiento de Conocimiento en Bases de Datos (DCBD) es básicamente un proceso automático en el que se combinan desarrollo 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 y presentar resultados [5] [8].

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 [9]. Las arquitecturas de estas herramientas se pueden ubicar en una de tres tipos: sistemas débilmente acoplados, medianamente acoplados y fuertemente acoplados con un Sistema Gestor de Bases de Datos (SGBD) [13]. 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 [11].

En el área de minería de datos, el problema de derivar asociaciones de los datos ha sido un centro de gran atención e importancia. El problema fue formulado por Agrawal et al [1] [2] y a menudo se referencia como el problema de canasta de mercado (*market-basket*). En este problema se da un conjunto de ítems y una colección de transacciones que son subconjuntos (canastas) de estos ítems. La tarea es encontrar relaciones entre la presencia de varios ítems con esas canastas. La formalización de este problema es encontrar reglas de asociación que cumplan unas especificaciones mínimas dadas por el usuario, expresadas en forma de *soporte* y *confianza* [1]. Un conjunto de ítems, se denomina frecuente si su soporte excede un umbral dado.

El cálculo de conjuntos de ítems frecuentes es la etapa computacionalmente más costosa cuando se aborda la tarea de asociación. Los distintos algoritmos propuestos para implementarla se han concentrado en el cálculo de conjuntos frecuentes: Apriori, AprioriTid, Apriori Hibrido [2], DHP (Standing for Direct Hashing and Pruning) [10], DIC (Dynamic Itemset Counting) [3], Partition [12], Sampling [17], FP-Growth [6] [7] y EquipAsso [15] [16].

EquipAsso [15] [16], es un algoritmo para el cálculo de los conjuntos de ítems frecuentes basado en dos nuevos operadores del álgebra Relacional para Asociación: *Associator* y *EquiKeep* [14] e implementado en el lenguaje SQL mediante las nuevas primitivas *Associator Range* y *EquiKeep On* [14].

En este artículo, se evalúa el rendimiento del algoritmo EquipAsso [15] [16] con relación a los algoritmos Apriori [2] y FP-Growth [6] [7] en Tariy, una herramienta de tarea sencilla para el descubrimiento de reglas de asociación débilmente acoplada con un SGBD.

El resto del artículo se organiza en secciones. En la sección 2, se dan algunos conceptos sobre Asociación y sobre los nuevos operadores del álgebra relacional que soportan esta tarea. En la sección 3, se describe la arquitectura de la herramienta Tariy, así como también algunos aspectos de la implementación de los diferentes módulos. En la sección 4, se hace el análisis del rendimiento de los algoritmos Apriori, FP-Growth y EquipAsso, implementados en Tariy. Finalmente, en la sección 5, se presentan las conclusiones.

2. Conceptos Preliminares

2.1 Reglas de Asociación

El problema de encontrar reglas de asociación fue formulado por Agrawal et al [1] [5] y a menudo se referencia como el problema de canasta de mercado (market-basket). En este problema se da un conjunto de ítems y una colección de transacciones que son subconjuntos (canastas) de estos ítems. La tarea es encontrar relaciones entre la presencia de varios ítems en esas canastas.

Formalmente, sea $I= \{i_1, i_2,..., i_m\}$ un conjunto de literales, llamados ítems. Sea D un conjunto de transacciones, donde cada transacción T es un conjunto de ítems tal que $T \subseteq I$. Cada transacción se asocia con un identificador, llamado TID. Sea X un conjunto de ítems. Se dice que una transacción T contiene a X si y solo si $X \subseteq T$. Una regla de asociación es una implicación de la forma $X \Rightarrow Y$, donde X y Y son conjuntos de ítems tal que $X \subset I$, $Y \subset I$ y $X \cap Y = \phi$. El significado intuitivo de tal regla es que las transacciones de la base de datos que contienen X tienden a contener Y. La regla $X \Rightarrow Y$ se cumple en el conjunto de transacciones D con una confianza **c** si el c% de las transacciones en D que contienen X también contienen Y. La regla $X \Rightarrow Y$ tiene un soporte **s** en el conjunto de transacciones D si el s% de las transacciones en D contienen $X \cup Y$.

Un ejemplo de una regla de asociación es: "el 30% de las transacciones que contienen cerveza también contienen pañales; el 2% de todas las transacciones contienen ambos ítems" [2]. Aquí el 30% es la confianza de la regla y el 2%, el soporte de la regla.
La confianza denota la fuerza de la implicación y el soporte indica la frecuencia de ocurrencia de los patrones en la regla. Las reglas con una confianza alta y soporte fuerte son referidas como reglas fuertes (strong rules) [1]. El problema de encontrar reglas de asociación se descompone en los siguientes pasos:

- Descubrir los itemsets frecuentes, i.e., el conjunto de itemsets que tienen el soporte de transacciones por encima de un predeterminado soporte ***s*** mínimo.

- Usar los itemsets frecuente para generar las reglas de asociación.

Después de que los itemsets frecuentes son identificados, las correspondientes reglas de asociación se pueden derivar de una manera directa.

2.2. Operadores algebraicos relacionales para Asociación

Timarán [14] extiende el álgebra relacional [10] con los siguientes operadores algebraicos:

Associator (α). Este operador genera, por cada tupla de la relación *R*, todos sus posibles subconjuntos (itemsets) de diferente tamaño, en una sola pasada.

EquiKeep (χ). Este operador 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.

2.3 Primitivas SQL para Asociación

Timarán [14] extiende el lenguaje SQL con las siguientes primitivas:

- *Primitiva Associator Range*. Esta primitiva implementa el operador algebraico *Associator* [14] en la cláusula SQL SELECT. Associator permite obtener por cada tupla de una tabla, todos los posibles subconjuntos desde un tamaño inicial hasta un tamaño final determinado por la cláusula RANGE. Dentro de la cláusula SELECT, *Associator* tiene la siguiente sintaxis:

```
SELECT <ListaAtributosTablaDatos> [INTO <NombreTablaAssociator>]
FROM <NombreTablaDatos>
WHERE <CláusulaWhere>
ASSOCIATOR RANGE <numero1> UNTIL <numero2>
GROUP BY <ListaAtributosTablaDatos>
```

- *Primitiva Equikeep On*. Esta primitiva implementa el operador algebraico *EquiKeep* [14] en la cláusula SQL SELECT. *EquiKeep on* conserva en cada registro de una tabla los valores de los atributos que cumplen una condición determinada. El resto de valores de los atributos se hacen nulos. Dentro de la cláusula SELECT, *EquiKeep on* tiene la siguiente sintaxis:

```
SELECT <ListaAtributosTablaDatos> [INTO <NombreTablaEquiKeep>]
    FROM <NombreTablaDatos>
    WHERE <CláusulaWhere>
    EQUIKEEP ON <CondiciónValoresAtributos >
```

3. Algoritmo Equipasso

Se han diseñado diferentes algoritmos para obtener los conjuntos de ítems frecuentes en la tarea de Asociación, dentro de los cuales están: Apriori, AprioriTid, Apriori Hibrido [2], DHP (Standing for Direct Hashing and Pruning) [9], DIC (Dynamic Itemset Counting) [3], Partition [15] y FP-Growth [6] [7]. Sin embargo, todos ellos se han implementado aisladamente y por fuera de los motores de bases de datos.

EquipAsso es un algoritmo que se basa en los nuevos operadores del álgebra relacional *Associator* y *EquiKeep* para el cálculo de conjuntos de ítems frecuentes. Este hecho facilita su integración al interior de cualquier SGBD, acoplando la tarea de Asociación de una manera fuerte y favorece la aplicación de técnicas de optimización de consultas para mejorar su rendimiento. A continuación se describe este algoritmo.

3.1 Descripción Algoritmo Equipasso

En el primer paso del algoritmo se cuenta el número de ocurrencias de cada ítem para determinar los 1-conjuntos de ítems frecuentes *L1*. En el subsiguiente paso, se aplica el operador *EquiKeep para* extraer de todas las transacciones en *D*, los conjuntos de ítems frecuentes tamaño 1, haciendo nulos el resto de valores. Luego, a la relación resultante *R*, se aplica el operador *Associator* para generar todos los conjuntos de items tamaño 2 (Is = 2) hasta máximo tamaño *n*, donde *n* es el grado de *R*. Finalmente, se calculan todos los conjuntos de ítems frecuentes *L*, contando el soporte de las diferentes combinaciones generadas por *Associator* en la relación *R'*. En la figura 1

se muestra el algoritmo *Equipasso*.

3.2 Algoritmo SQL-Equipasso

Utilizando las nuevas primitivas SQL para minar reglas de Asociación, el algoritmo *EquipAsso* de la figura 1, se implementa con la cláusula SELECT de la siguiente manera:

```
Leer g // tamaño de la regla
L1 = (1 – conjuntos de ítems frecuentes);
Forall transacciones t ∈ D do begin
  // se aplica el operador EquiKeep
  R = χL1(D)
  k = 2
  // genera todos los conjuntos de ítems posibles
  R' = αk,q (R) = { ∪all Xi | Xi ⊆ ti }
End
// conjuntos de ítems frecuentes
L = { count(R') | count ≤ minsup }
```

Figura 1. Algoritmo EquipAsso

SELECT <ListaAtributosTablaDatos>, count(*) *AS* soporte *INTO R'*

FROM D

EQUIKEEP ON L1

ASSOCIATOR RANGE k UNTIL g

GROUP BY <ListaAtributosTablaDatos> *HAVING* count(*) >= minsup

3.3 Ejemplo de Aplicación del Algoritmo Equipasso

Sea la tabla booleana *Transacción* (Tid, Item_1, Item_2, Item_3, Item_4), la cual se indica en la tabla 1.

Tid	Item_1	Item_2	Item_3	Item_4
100	1	1	0	0
200	0	1	1	1
300	1	1	1	0
400	0	1	1	1
500	0	0	1	1

Tabla 1. Relación Transacción

Suponiendo que ya se han calculado los conjuntos de ítems frecuentes de tamaño 1 con un soporte igual a 3, los cuales son {Item_2}, {Item_3}, {Item_4}. Encontrar los conjuntos de ítems frecuentes de tamaño 2 y 3 formados por lo atributos Item_1, Item_2, Item_3, Item_4 y almacenar los resultados en la tabla *Frecuentes,* utilizando el algoritmo *EquipAsso*:

La sentencia SQL que permite obtener esta consulta es la siguiente:

SELECT Item_1, Item_2, Item_3, Item_4, count(*) AS soporte *INTO* Frecuentes

FROM Transaccion

EQUIKEEP ON Item_2 =1 *OR* Item_3=1 *OR* Item_4=1

ASSOCIATOR RANGE 2 *UNTIL* 3

GROUP BY Item_1, Item_2, Item_3, Item_4 *HAVING* count(*)>=3

Inicialmente se ejecuta la cláusula *SELECT,* en este caso se seleccionan los atributos Item_1, Item_2, Item_3, Item_4 de la tabla *Transacción*. Luego sobre la tabla resultante, se ejecuta la primitiva *EQUIKEEP ON* que conserva los valores de los atributos que cumplan la condición I*tem_2 =1 OR Item_3=1 OR Item_4=1* asi:

Item_1	Item_2	Item_3	Item_4
null	1	null	null
null	1	1	1
null	1	1	null
null	1	1	1
null	null	1	1

Tabla 2. Resultado primitiva SQL EquiKeep

Posteriormente, se generan los conjuntos de ítems de tamaño 2 y 3 con la primitiva *ASSOCIATOR RANGE* 2 *UNTIL* 3 así:

Item_1	Item_2	Item_3	Item_4
null	1	1	null
null	1	null	1
null	null	1	1
null	1	1	1
null	1	1	null
null	1	1	null
null	1	null	1
null	null	1	1
null	1	1	1
null	null	1	1

Tabla 3. Resultado primitiva SQL Associator Range

El resultado del agrupamiento y el conteo se presenta en la tabla 4.

En la tabla 5 se muestra la tabla *Frecuentes* donde se almacenan los conjuntos de items frecuentes cuyo soporte ≥ 3, que es el resultado final del algoritmo *SQL EquipAsso.*

Item_1	Item_2	Item_3	Item_4	soporte
null	1	1	null	3
null	1	null	1	2
null	null	1	1	3
null	1	1	1	2

Tabla 4. Resultado Agrupamiento y conteo

Item_1	Item_2	Item_3	Item_4	Soporte
null	1	1	null	3
null	null	1	1	3

Tabla 5. Resultado algoritmo SQL-EquipAsso

4. Aspectos de Implementación de EquipAsso en Tariy

El algoritmo EquipAsso se implementó conjuntamente con los algoritmos Apriori y FP-Growth en TARIY: una herramienta para minería de datos débilmente acoplada con el SGBD PostgreSQL, con el fin de evaluar el rendimiento de este algoritmo con respecto a los dos últimos.

4.1 Arquitectura de Tariy

Tariy se desarrolló bajo el sistema operativo Fedora Core 3 64 bits, lenguaje Java 1.5.0_06, con un computador con procesador AMD 64 bits a 2 Ghz, memoria RAM de 1Gb, con disco duro Serial ATA de 80Gb con una tasa de transferencia de 150 Mb/seg., entorno de desarrollo NetBeans 5.0 y SGBD utilizado para la conexión fue PostgreSQL 7.4.

La Herramienta Tariy esta compuesta por 4 módulos principales (figura 2):

- Módulo de conexión a SGBD, el cual permite la comunicación de la herramienta con el SGBD y los datos.
- Algoritmos, donde se ejecutan el cálculo de los conjuntos frecuentes con los tres algoritmos propuestos.
- Módulo de generación de reglas, encargado de generar las reglas que cumplan las restricciones de confianza.

- Interfaz Grafica, que permite la interacción del usuario con la herramienta.

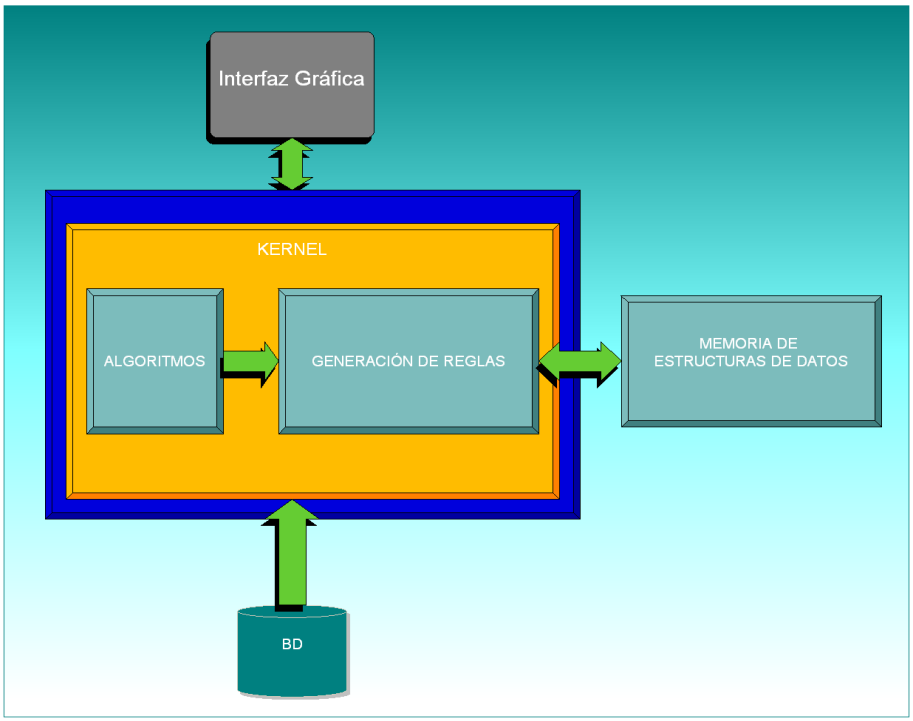


Figura 2. Arquitectura Herramienta Tariy.

4.2 Módulo de algoritmos

Los algoritmos EquipAsso, A-priori y FP-Growth se implementaron bajo el lenguaje de programación Java v.1.5.0_06.

Por otra parte para el almacenamiento de los datos provenientes del SGBD se utiliza un árbol n-ario el cual sirve para almacenarlos de forma compacta aprovechando que diferentes transacciones pueden compartir ítems repetidos, cada algoritmo obtiene las transacciones del árbol y se procede a la aplicación de los mismos. Los conjuntos de itemsets frecuentes obtenidos son almacenados en un árbol AVL balanceado.

5. Pruebas y Evaluación

Las pruebas y evaluación del rendimiento de los algoritmos Apriori, FP-Growth y EquipAsso, se realizaron en un computador con un procesador AMD 64 bits a 2 Ghz, con una memoria RAM de 1Gb, con disco duro Serial ATA de 80Gb con una tasa de transferencia de 150 Mb/seg.

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 Tariy se muestran en la tabla 6.

Para cada conjunto de datos se realizó preprocesamiento y transformación de datos con el fin de eliminar los productos repetidos en cada transacción y transformar las tablas objeto a un modelo simple (i.e. una tabla con esquema Tid, Item).

Nomenclatura	*Número de registros*	*Número de transacciones*	*Promedio ítems por Transacción*
BD85KT7	555.123	85.692	7
BD40KT5	194.337	40.256	5
BD10KT10	97.824	10.731	10

Tabla 6. Descripción de los conjuntos de datos minados.

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 3, 4 y 5 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.

BD85KT7			
Soporte (%)	Tiempo (ms)		
	Apriori	FP-Growth	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

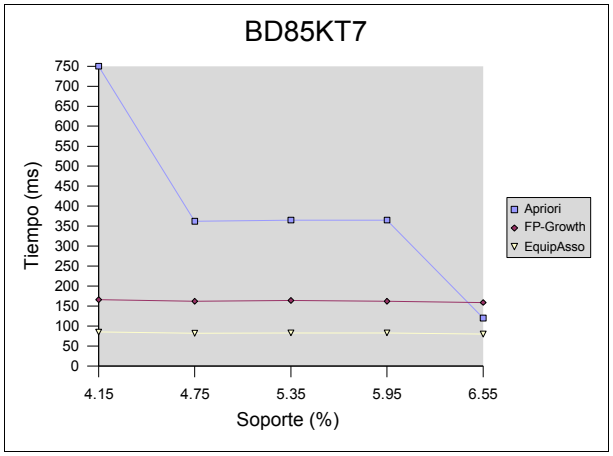


Figura 3. Resultados obtenidos para BD85KT7.

BD40KT5			
Soporte (%)	Tiempo (ms)		
	Apriori	FP-Growth	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

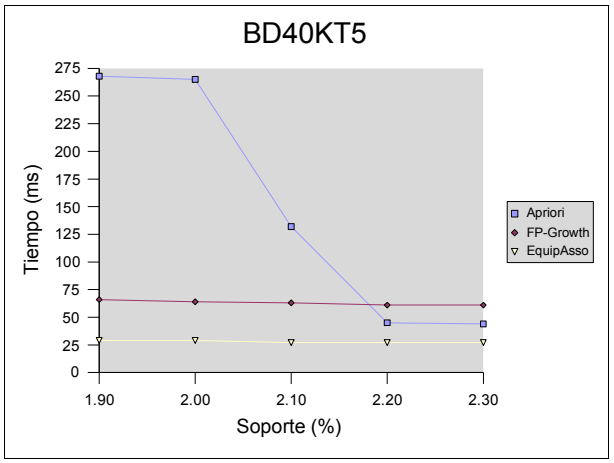


Figura 4. Resultados obtenidos para BD40KT5.

BD10KT10			
Soporte (%)	Tiempo (ms)		
	Apriori	FP-Growth	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

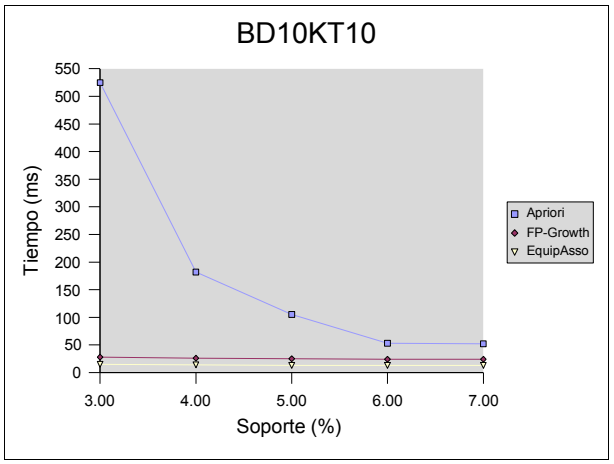


Figura 5. Resultados obtenidos para BD10KT10.

Analizando el tiempo de ejecución de únicamente los algoritmos FP-Growth y EquipAsso (figuras 6, 7 y 8) para los conjuntos de datos BD85KT7, BD40KT5 y BD10KT10 (figuras 6 y 7), con soportes mas bajos, el comportamiento de estos algoritmos sigue siendo similar.

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

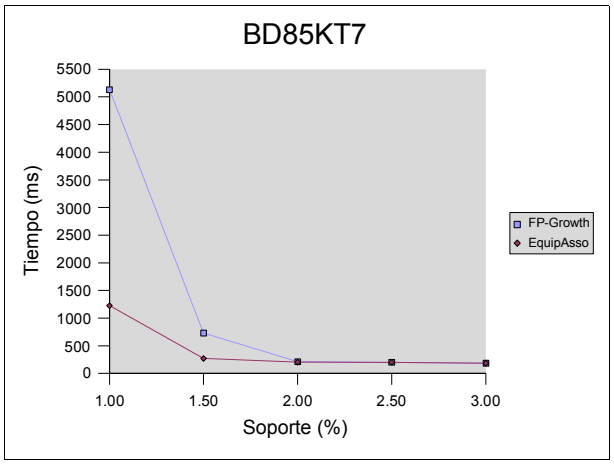


Figura 6. Resultados obtenidos para BD85KT7.

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

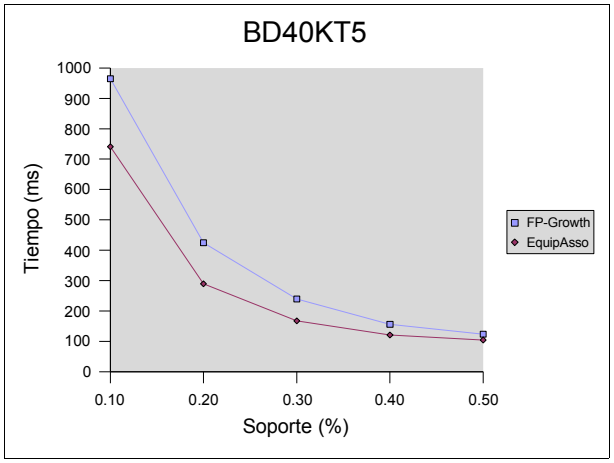


Figura 7. Resultados obtenidos para BD40KT5

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

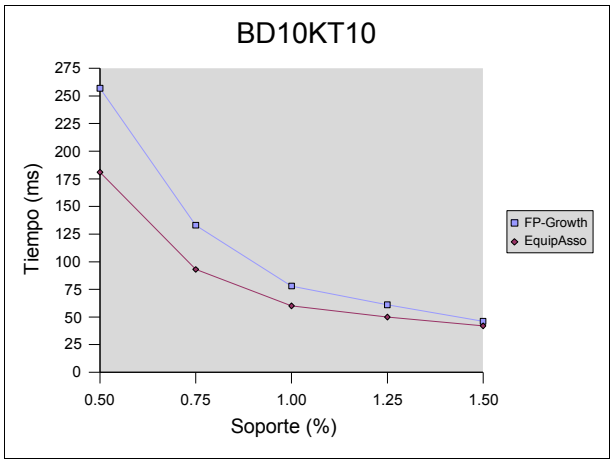


Figura 8. Resultados obtenidos para BD10KT10

6. Conclusiones

En este artículo se evaluó el desempeño del algoritmo EquipAsso [15] [16] con respecto a Apriori [2] y FP-Growth [7], para la tarea de minería de datos asociación, implementándolos en Tariy, una herramienta de minería de datos débilmente acoplada con PostgreSQL y desarrollada en el lenguaje JAVA 2. Para las pruebas, se utilizaron conjuntos de datos reales, pertenecientes a las ventas de un periodo determinado, de uno de los supermercados más importantes del departamento de Nariño (Colombia).

Como resultado de las pruebas realizadas, se puede observar que los algoritmos EquipAsso y FP-Growth tienen tiempos de ejecución semejantes. Con respecto a Apriori se puede apreciar que a medida que se disminuye el soporte sus tiempos de ejecución se incrementan hasta llegar a límites incomparables con respecto a EquipAsso y FP-Growth.

Para soportes bajos, donde la cantidad de itemsets frecuentes aumenta y lo mismo su tamaño, *Equipasso* tuvo un mejor desempeño que FP-Growth. En soportes altos, no existe una diferencia representativa entre el rendimiento de los algoritmos FP-Growth y Equipasso.

En la herramienta Tariy, el algoritmo Equipasso, aunque presenta mejores tiempos de ejecución, consume más recursos del sistema, en especial la memoria principal.

El rendimiento de cualquier algoritmo de minería de datos en una herramienta débilmente acoplada con un SGBD depende de la cantidad de memoria RAM disponible. Con grandes volúmenes de datos, el rendimiento de estos sistemas se ve afectado y es posible que se llegue a desbordar la memoria.

En el futuro, se complementará esta investigación realizando pruebas con diferentes conjuntos de datos reales y se continuará con el desarrollo de la herramienta Tariy con una interfaz gráfica y nuevos módulos que soporten otras tareas y algoritmos de minería de datos.

Referencias

[1] Agrawal R., Imielinski T., Swami A., Mining Association Rules between Sets of Items in Large Databases, ACM SIGMOD, Washington DC, USA, 1993.

[2] Agrawal R., Srikant R., Fast Algorithms for Mining Association Rules, VLDB Conference, Santiago, Chile, 1994.

[3] Brin S., Motwani R., Ullman J., Tsur S., Dynamic Itemset Counting and Implication Rules for Market Basket Data, ACM SIGMOD, USA, 1997.

[4] Codd, E., F., A Relational Model of Data for Large Shared Data Banks, CACM, Vol. 13, No. 6, June, 1970.

[5] Chen M., Han J., Yu P., Data Mining: An Overview from Database Perspective, IEEE Transactions on Knowledge and Data Engineering, 1996.

[6] Han, J., Pei, J., Yin, Y., Mining Frequent Patterns without candidate Generation, Proc. ACM SIGMOD, Dallas, TX, 2000.

[7] Han, J., Pei, J., Mining Frequent Patterns by Pattern-Growth: Methodology and Implications, SIGKDD Explorations, 2:14-20, 2000

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

[9] Imielnski T., Mannila, H., A Database Perspective on Knowledge Discovery, Communications of the ACM, Vol 39, No. 11, November, 1996.

[10] Park J.S., Chen M., Yu P., An Effective Hash-Based Algorithm for Mining Association Rules, ACM SIGMOD, San Jose, Ca USA, 1995.

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

[12] Savasere A., Omiecinski E., Navathe S., An Efficient Algorithm for Mining Association Rules in Large Databases, VLDB Conference, Zurich, Suiza,1995.

[13] Timarán, R., 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, Universidad del Valle, Volumen 3, No. 2, Cali, diciembre de 2001.

[14] Timarán R., Millán M., Machuca F., New AlgebraicOperators and SQL Primitives for Mining Association Rules, Proceedings of the IASTED International Conference Neural Networks and Computational Intelligence, Cancun, Mexico, 2003.

[15] Timarán, R., Millán, M., EquipAsso: un algoritmo para el descubrimiento de Reglas de Asociación basado en operadores algebraicos, en memorias de la 4ª Conferencia Iberoamericana en Sistemas, Cibernética e Informática CICI 2005, Orlando, Florida, EE.UU., julio 2005.

[16] Timarán, R., Millán, M., EquipAsso: an Algorithm based on New Relational Algebraic Operators for Association Rules Discovery, in proceedings of the Fourth IASTED International Conference on Computational Intelligence,

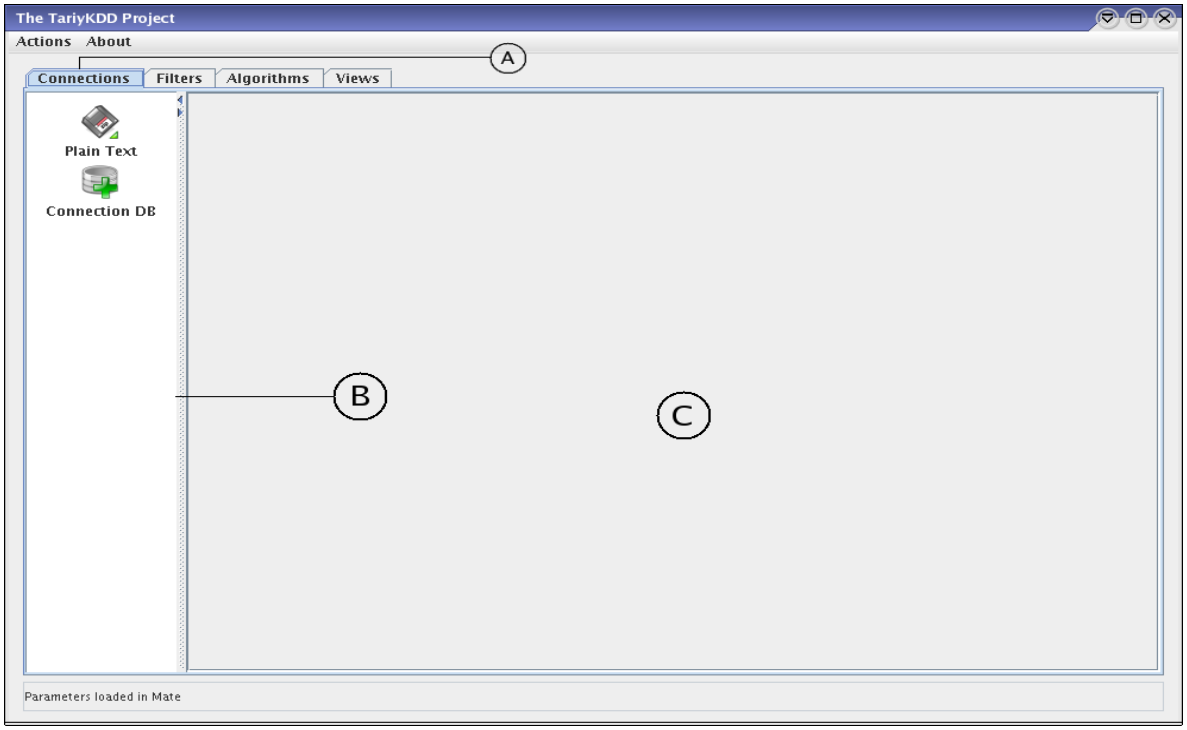
Calgary, Alberta, Canada, july 2005.

[17] Toivonen H., Sampling large databases for association rules, VLDB Conference, Bombay, India, 1996.

C. MANUAL DE USUARIO

C.1 Ingreso a la aplicación

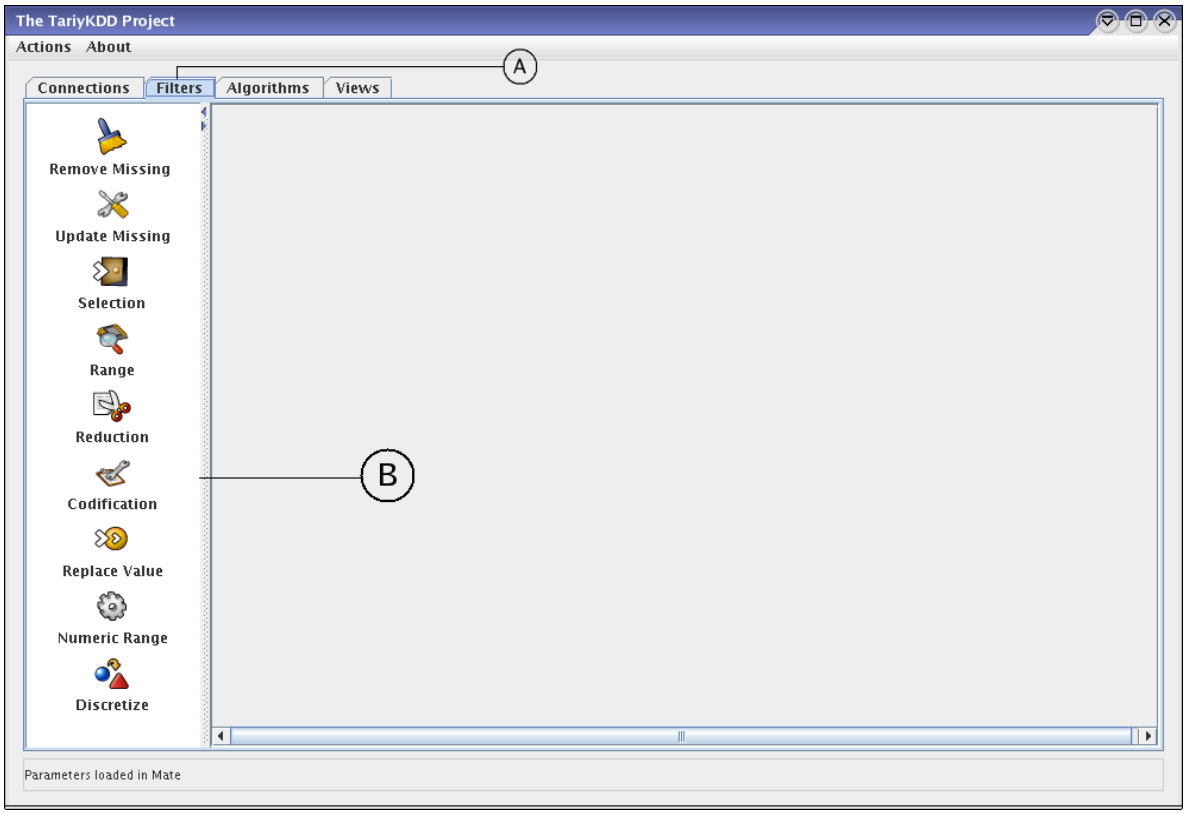
Figura C.1 Ingreso y conexión



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario ejecuta la aplicación	2. La interfaz gráfica de la aplicación aparece como se muestra en la figura. A: Area de pestañas a través de las cuales se puede acceder a los diferentes módulos de la aplicación. Ej, el módulo por defecto es ‘Connections‘. B: Area en la que aparecen las opciones de cada módulo. Ej, las opciones del módulo ’Connections’ son ’Archivo Plano’ y ’Conexión DB’. C: Area de trabajo sobre la que se arman los proyectos de Minería de Datos

Módulo filtros

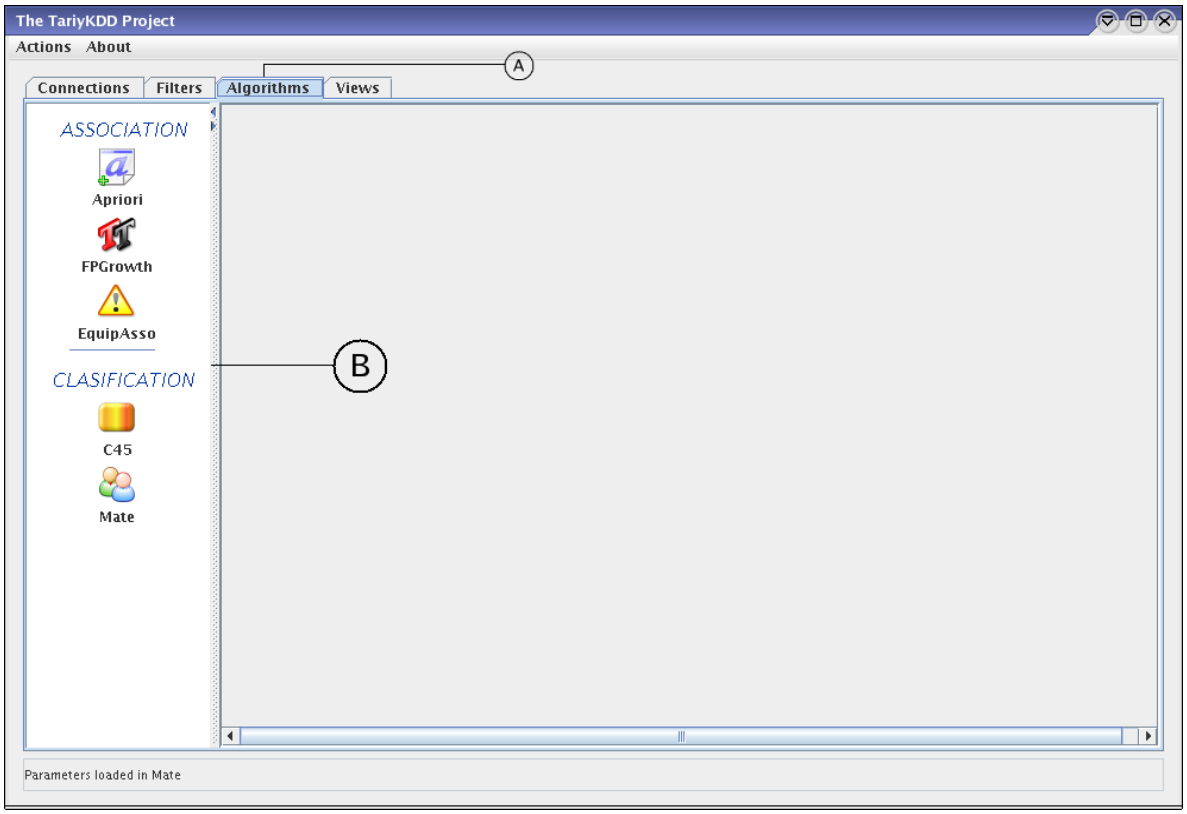
Figura C.2: Módulo filtros



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A ’Filtros’	2. Aparecen B Las opciones del módulo ’Filtros’

Módulo algoritmos

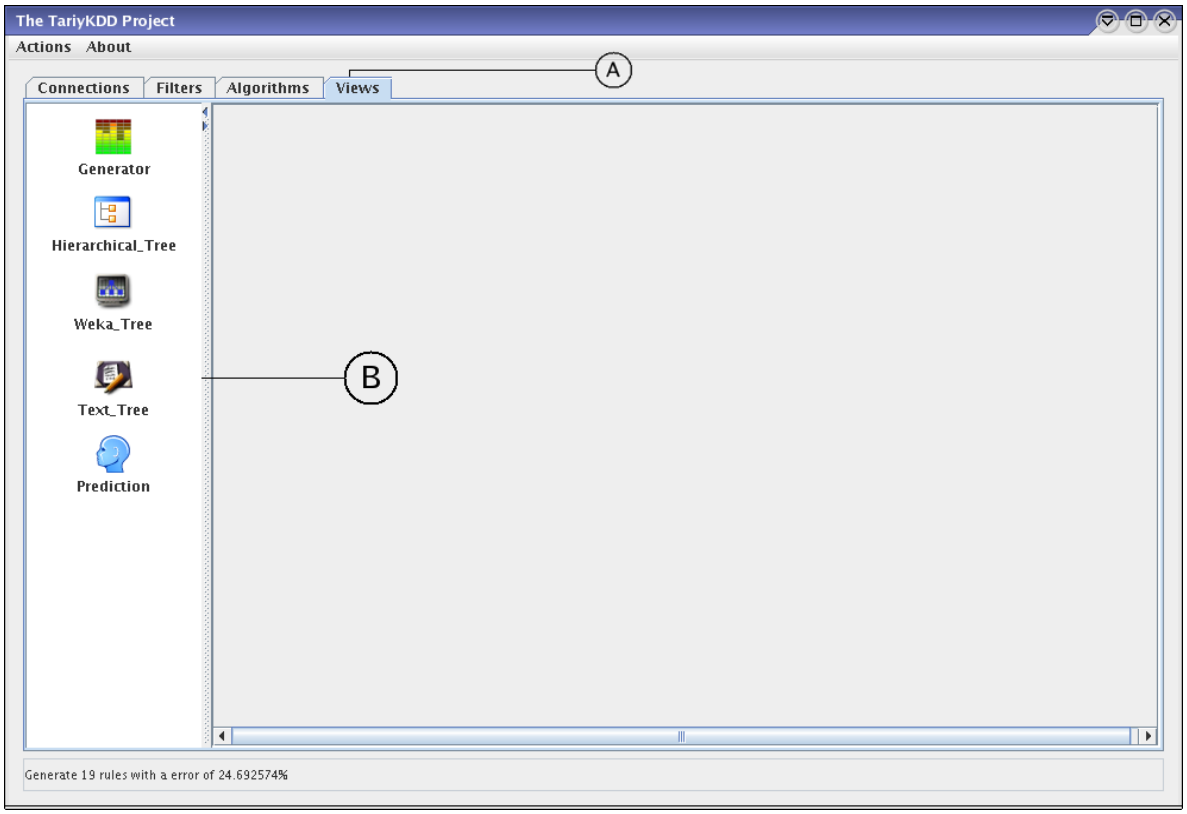
Figura C.3: Módulo algoritmos



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A ’Algoritmos’	2. Aparecen B Las opciones del módulo ’Algoritmos’

Módulo visualización

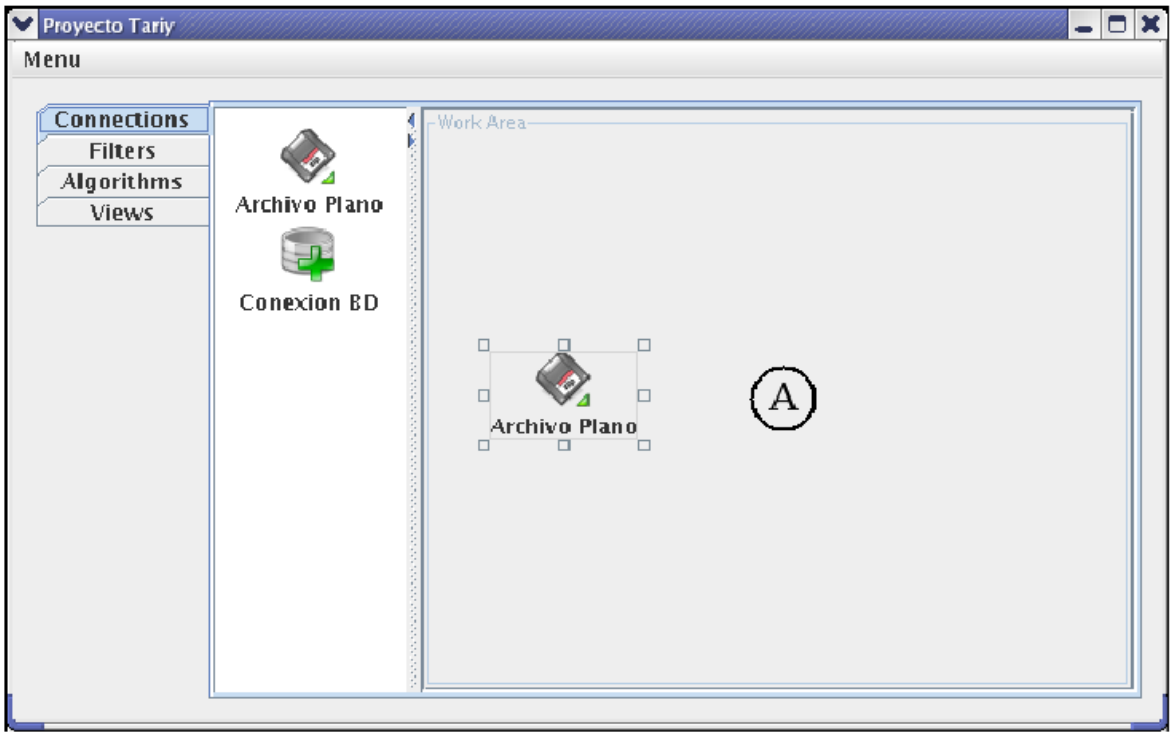
Figura C.4: Módulo visualización



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A ’Visualización"	2. Aparecen B Las opciones del módulo ’Visualización'

C.2 Conexión a un archivo plano

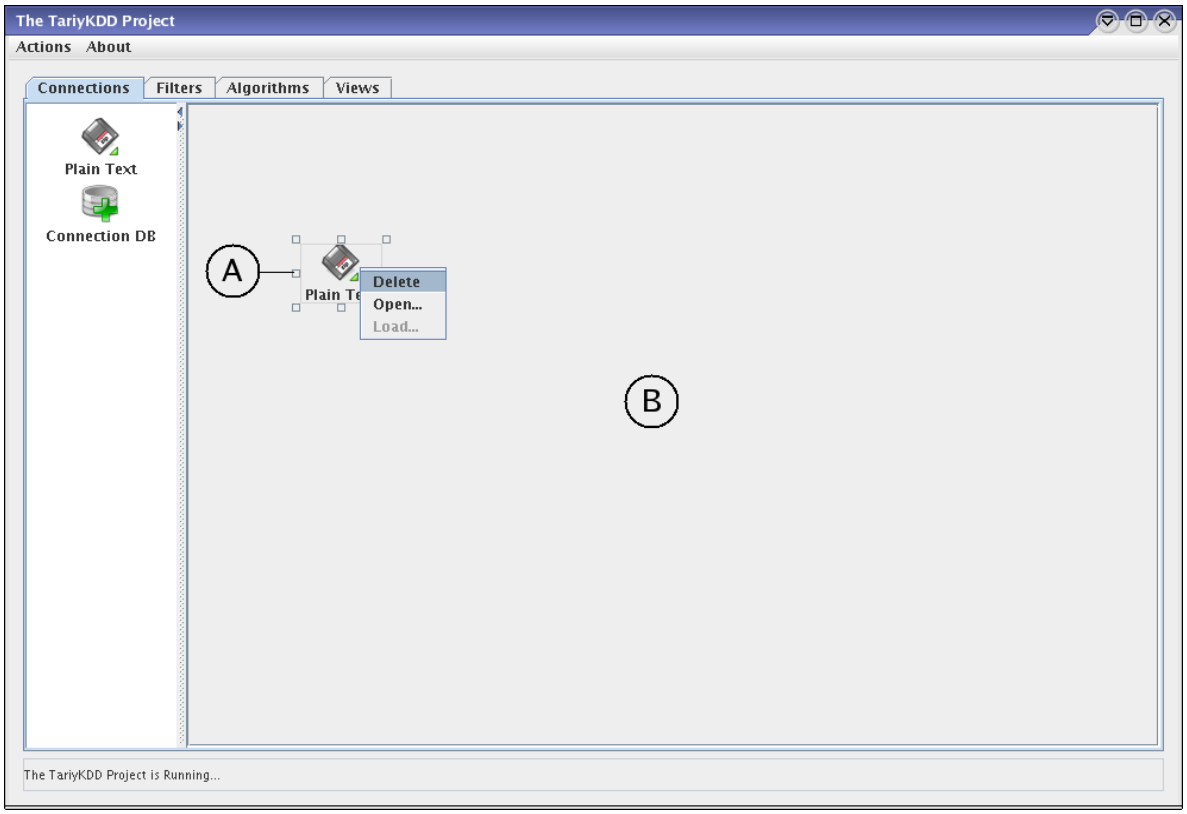
Figura C.5: Conexión a un archivo plano



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click sobre el ıcono ’Plain Text’	2. El ıcono ’Plain Text’ aparece sobre A: área de trabajo.

Opción “Delete” del archivo plano

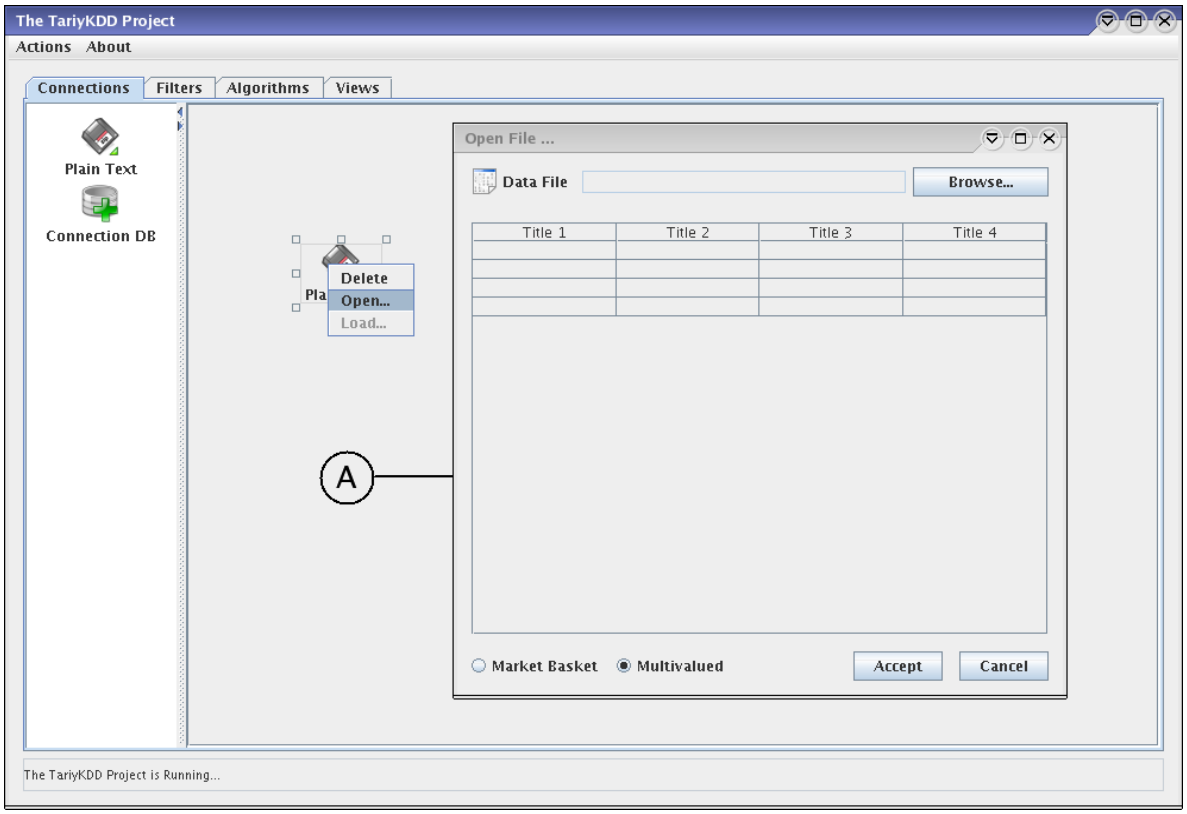
Figura C.5.1: Opción “Delete” del archivo plano



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Plain Text’ y selecciona la opción “Delete”.	2. El ıcono A ’Plain Text’ es borrado del área de trabajo B.

Opción “Open” del archivo plano

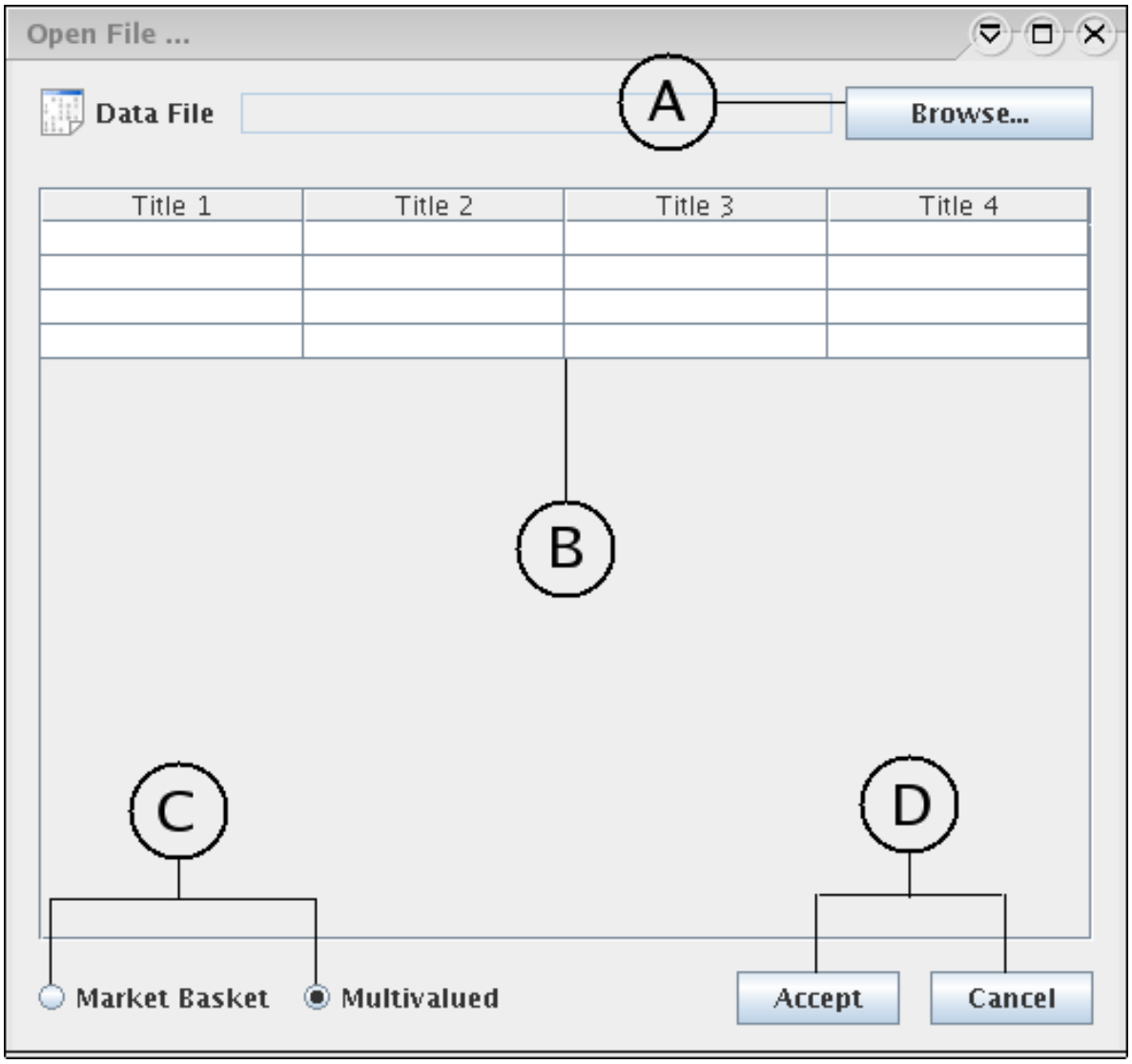
Figura C.5.2: Abrir un archivo plano.



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Plain Text’ y selecciona la opción “Open”.	2. Aparece la ventana de visualización de datos cargados desde archivo plano A explicada en el siguiente caso de uso.

Ventana de visualización de datos cargados desde archivo plano

Figura C.5.2.1: Interfaz de usuario para cargar datos desde archivo plano

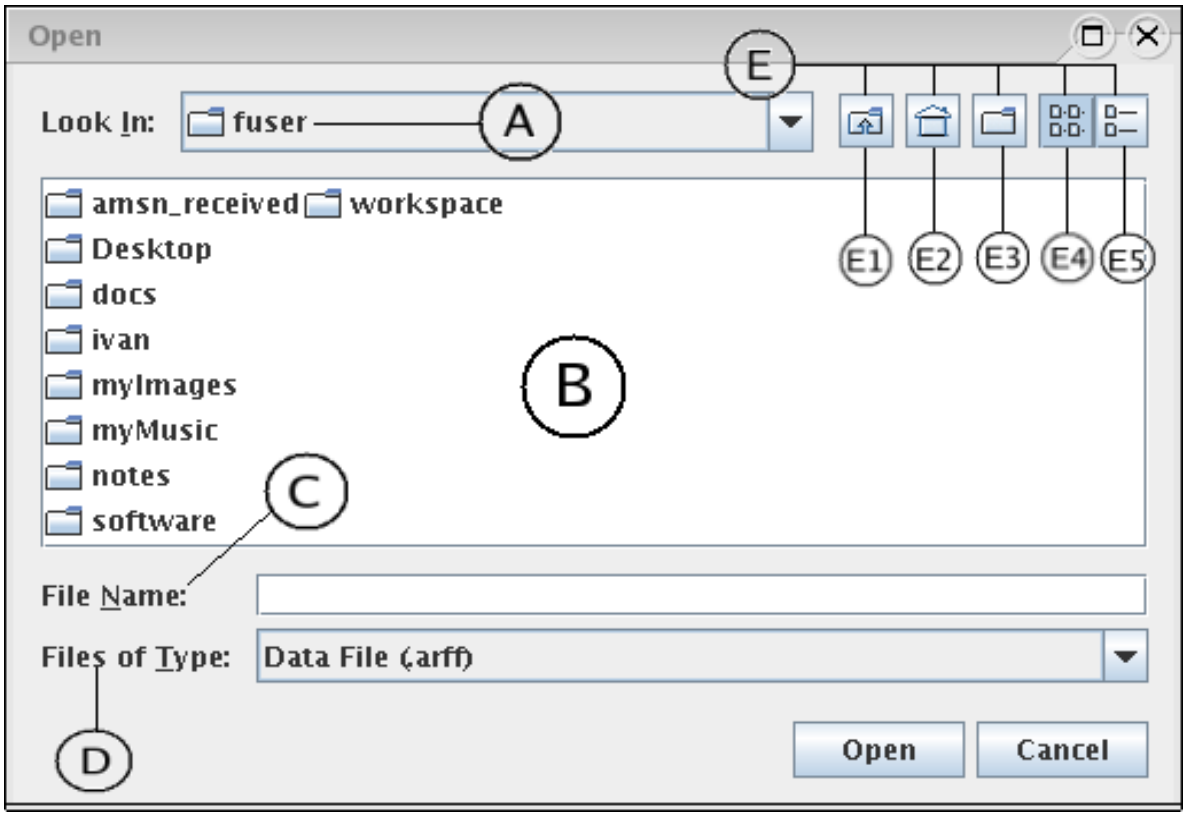


ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
	3. Aparece la ventana de visualización de datos cargados desde archivo plano. La cual esta compuesta por: A botón “Browse” que permite la navegación en el sistema de archivos, B la grilla en la que serán mostrados los datos cargados, C las opciones “Marquet Basket” y “Multivaluled” que el usuario puede

ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
	seleccionar según el caso y D los botones de aceptar o cancelar la operación ”Accept” y “Cancel”.
4. El usuario hace click en el botón “Browse”.	5. Aparece el navegador de archivos mostrado en el siguiente caso de uso real.

Navegador de archivos

Figura C.5.2.2: Interfaz de usuario para buscar archivos planos

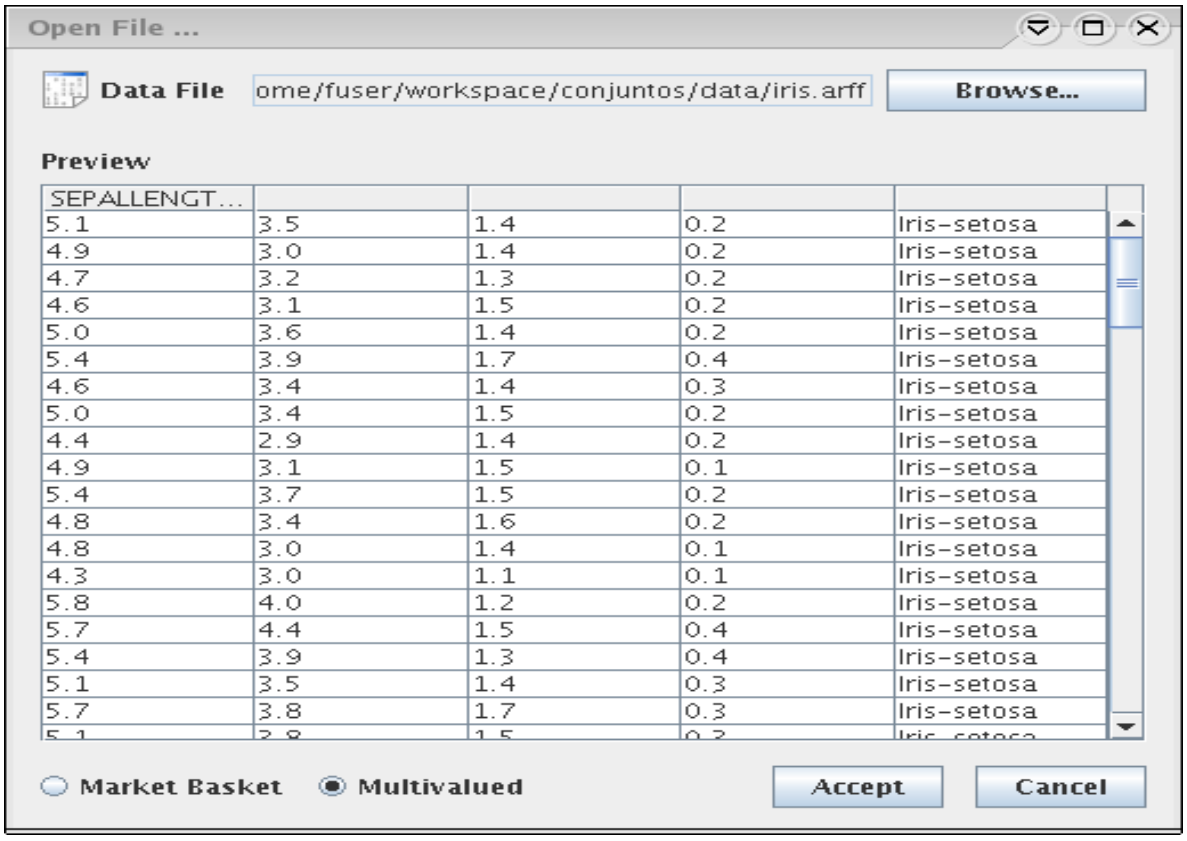


ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
	1. Aparece la interfaz de usuario para navegar, buscar y seleccionar un archivo plano. Las opciones disponibles son: A carpeta en la que se encuetra el usuario, B espacio en el que se muetran los archivos contenidos en la carpeta explorada, C nombre del archivo o carpeta, D extención de los archivos buscados, E grupo de botones de exploración: E1 subir un nivel, E2 ir al directorio configurado por defecto que por lo general es el directorio principal del usuario o “Home”, E3 crear carpeta nueva, E4 modo de visualización de archivos “Lista” y E5 modo de

ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
	visualización de archivos “Detalles”.

Datos cargados desde archivo plano

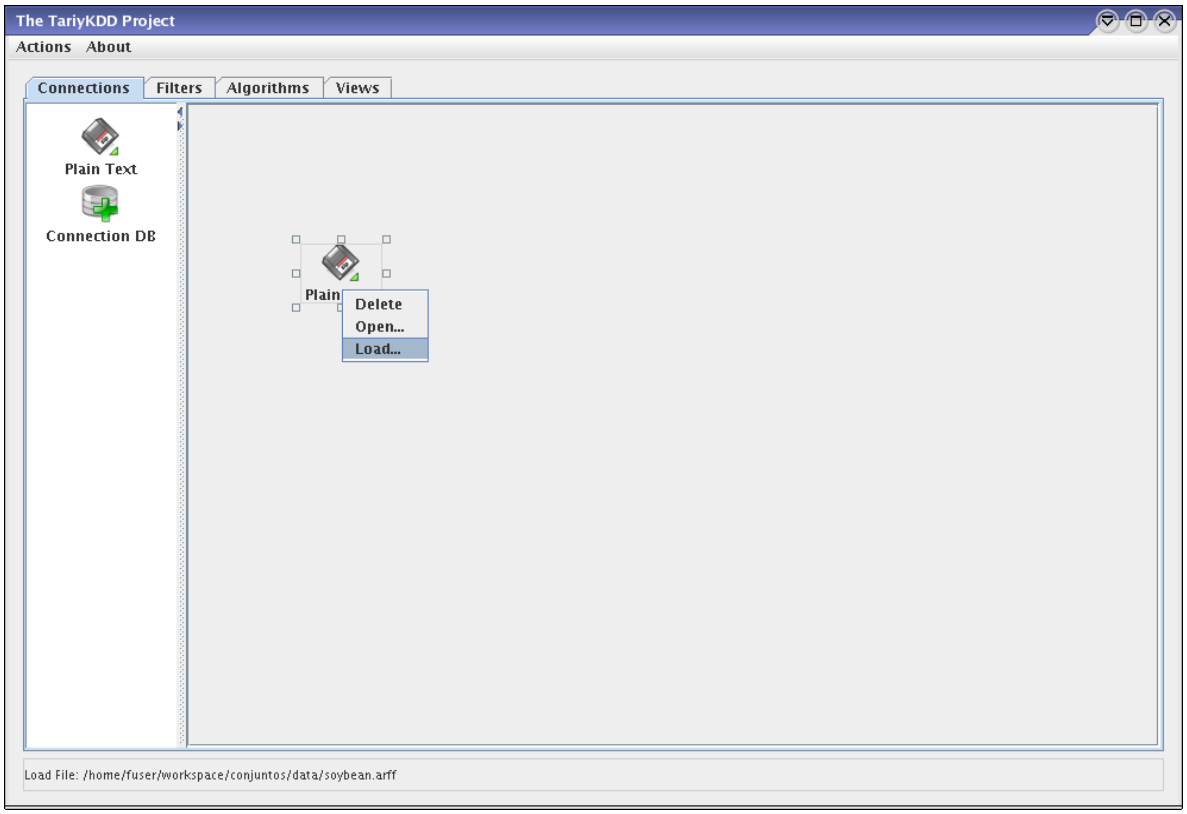
Figura C.5.3: Datos cargados desde archivo plano



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace uso del navegador de archivos y selecciona un archivo.	2. Los datos del archivo seleccionado son cargados en la grilla y son visualizados.

Opción “Load” del archivo plano

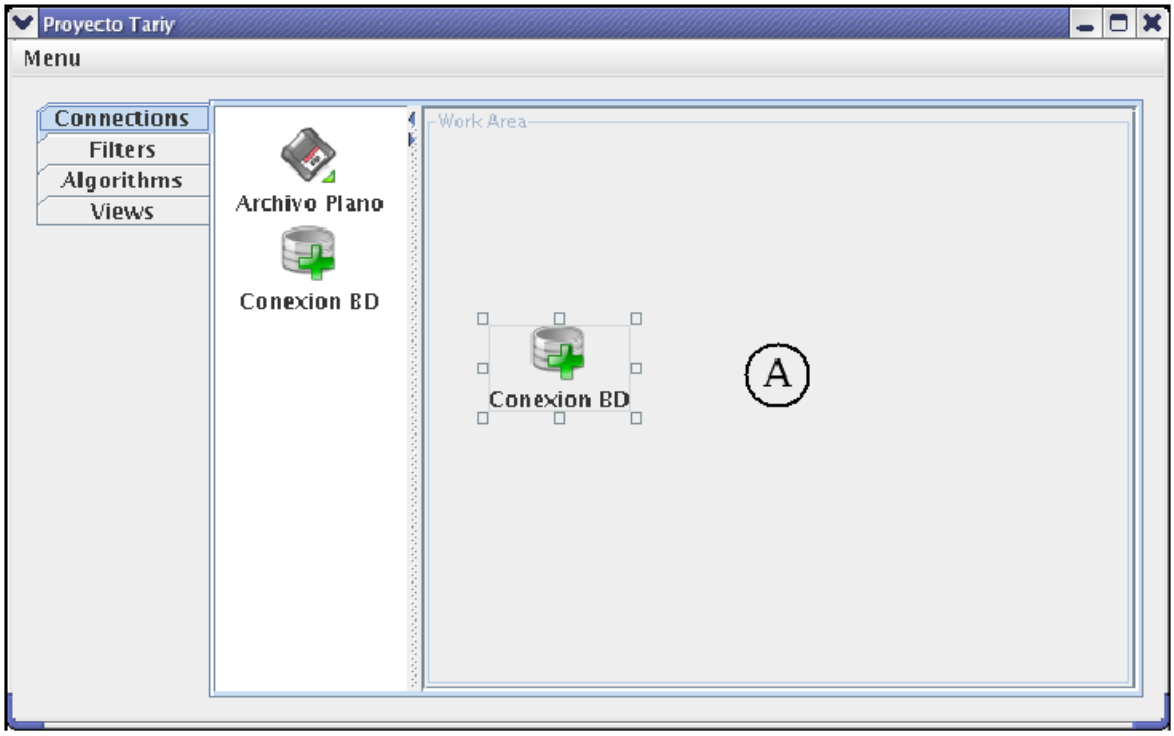
Figura C.5.4: Cargar datos del archivo plano seleccionado



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Plain Text’ y selecciona la opción “Load”.	2. El sistema carga los datos del archivo plano seleccionado y los deja listos para las siguientes operaciones de minería de datos, sea la aplicación de un filtro o algoritmo.

C.3 Conexión a una base de datos

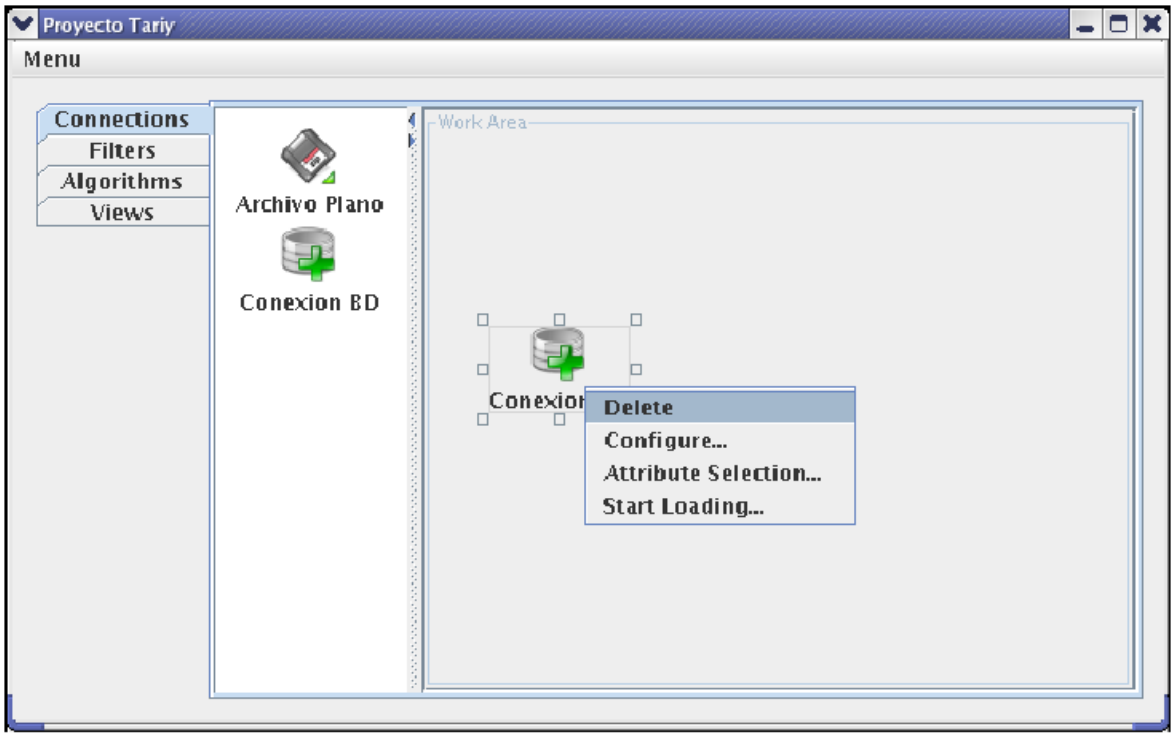
Figura C.6: Conexión a una base de datos



ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
1. El usuario hace click sobre el ıcono ’Conexión BD’	2. El ıcono ’Conexión BD’ aparece sobre A: área de trabajo.

Menú emergente conexión BD

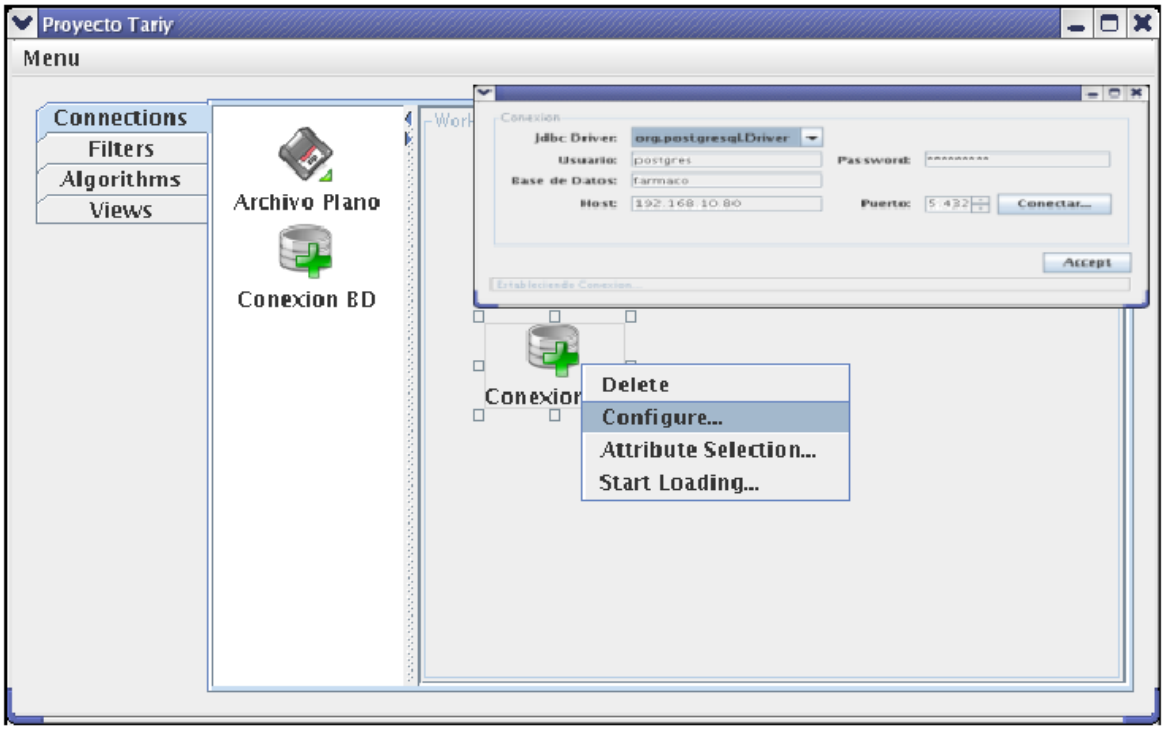
Figura C.6.1: Menú emergente conexión BD



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Conexión BD’	2. Se depliega A: menu del ıcono ’Conexión BD’. Las opciones son: ’Delete’: usada para eliminar el ıcono del área de trabajo. ’Configure’: usada para configurar la conexión a una base de datos. ’Selección de atributos’: usada para seleccionar de forma gráfica los datos que serán usados más adelante. ’Cargar’: ejecuta el query que se genera en la selección de atributos.

Configuración conexión BD

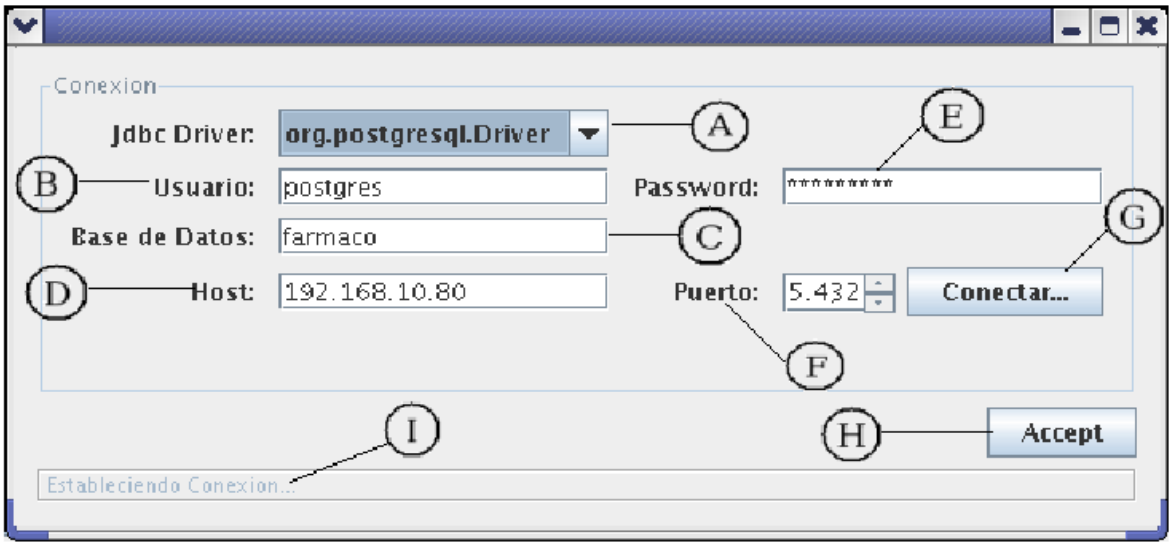
Figura C.6.2: Configuración conexión BD



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Conexión BD’ y selecciona la opción ’Configure’	2. Emerge una ventana de configuración de conexión a bases de datos

Ventana de conexión BD

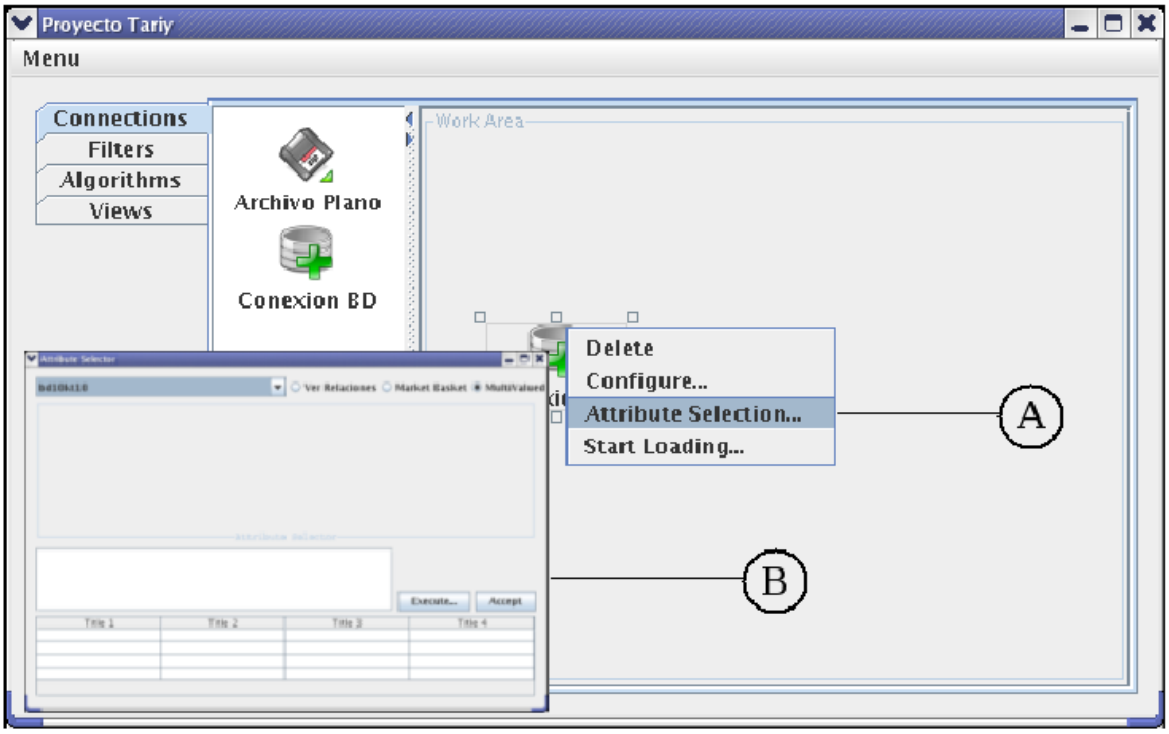
Figura C.6.2.1: Ventana de conexión BD



ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
1. El usuario desea configurar la conexión a una base de datos'	2. Las opciones de la ventana de configuración de conexión a bases de datos tiene los siguientes campos: A: Lista de controladores ODBC para varios tipos de bases de datos. B: Nombre del usuario de la base de datos. C: Nombre de la base de datos. D: Nombre del servidor. E: 'Password': clave de acceso a la base de datos. F: nmero del puerto utilizado para la comunicación con la base de datos. G: botón de conexión. H: botón para aceptar la conexión hecha. I: mensaje que indica el estado de la conexión.

Selección de atributos

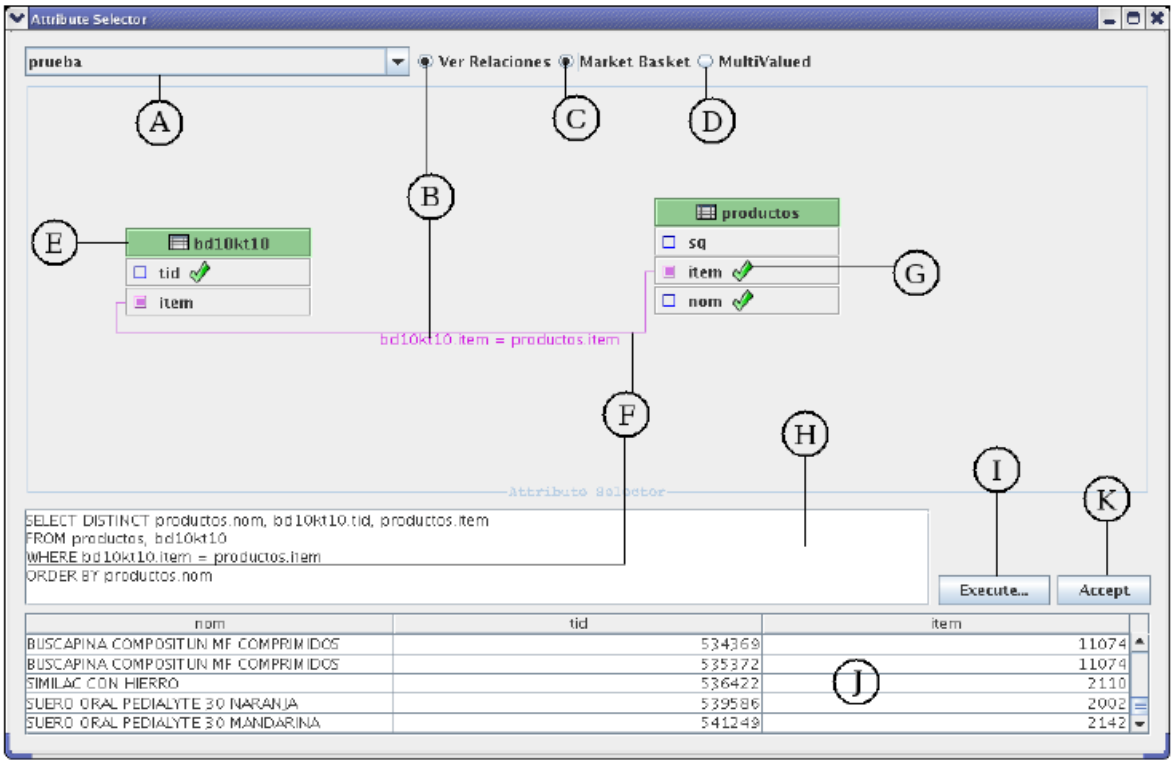
Figura C.6.3: Selección de atributos



ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
1. El usuario hace click derecho sobre el ıcono 'Conexión BD' para hacer la selección de atributos	2. Aparece el menú emergente del ıcono.y se ejecuta la ventana de selección de atributos A.
3. El usuario hace click en la opción B: 'Selección de Atributos'	4. Aparece la ventana de selección de atributos B.

Ventana selección de atributos

Figura C.6.3.1: Ventana selección de atributos



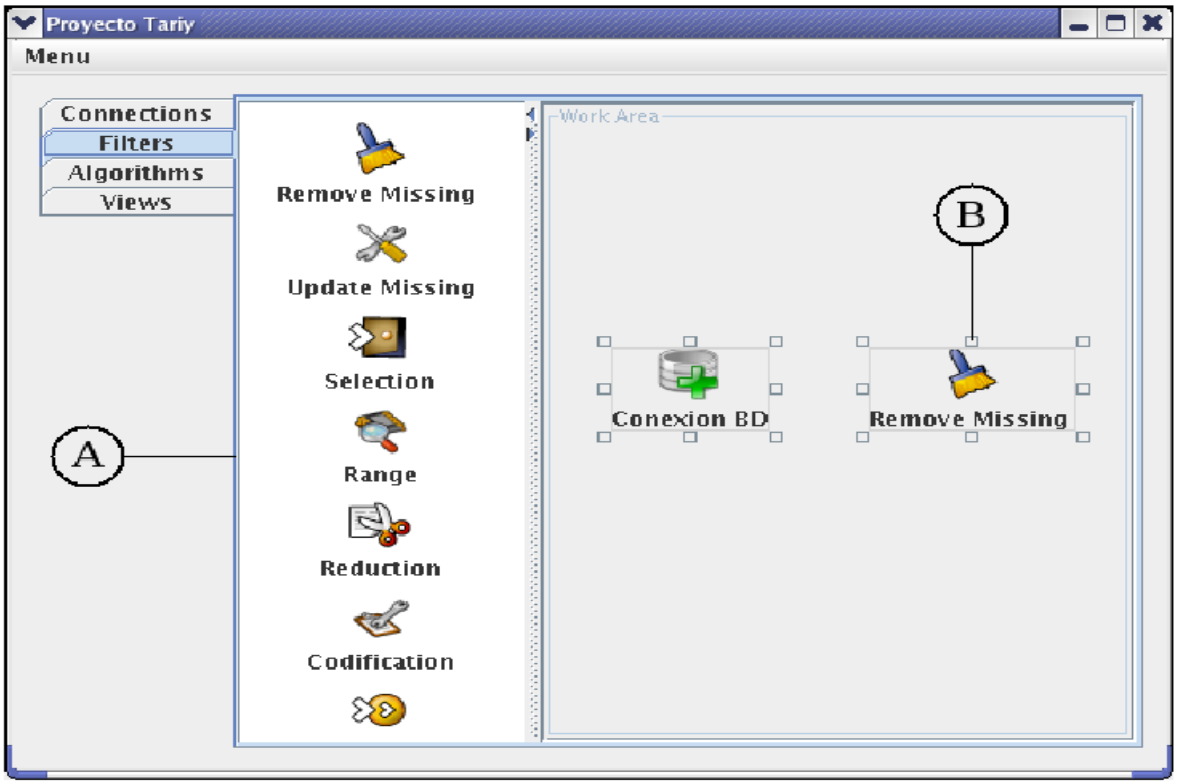
ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario desea hacer la selección de atributos	2. Aparece la ventana de selección'o de atributos. A: lista desplegable de las tablas de la base de datos a la que se ha conectado. Al seleccionar una de ellas su representación gráfica aparecera en el espacio de trabajo E. B: opción que permite ver las relaciones establecidas a través de la línea de conexión de atributos entre las tablas. C: esta opción es util cuando se trabajan problemas de canasta de mercado. D: opción para trabajar tablas multivaluadas. F: línea que permite realizar las relaciones entre atributos de dos tablas. El resultado de la relación establecida se refleja en el query. G:Si se hace click sobre uno de los atributos

ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
	aparece un ıcono de verificación que indica los campos que seránn mostrados al ejecutar el query. H: espacio en el que se crea el query. Es posible editarlo manualmente. I: botón de ejecución del query. J: tabla en la que se muestra el resultado de la ejecución del query. K: botón para aceptar las operaciones realizadas.

C.4 Filtros

Filtro Remove Missing

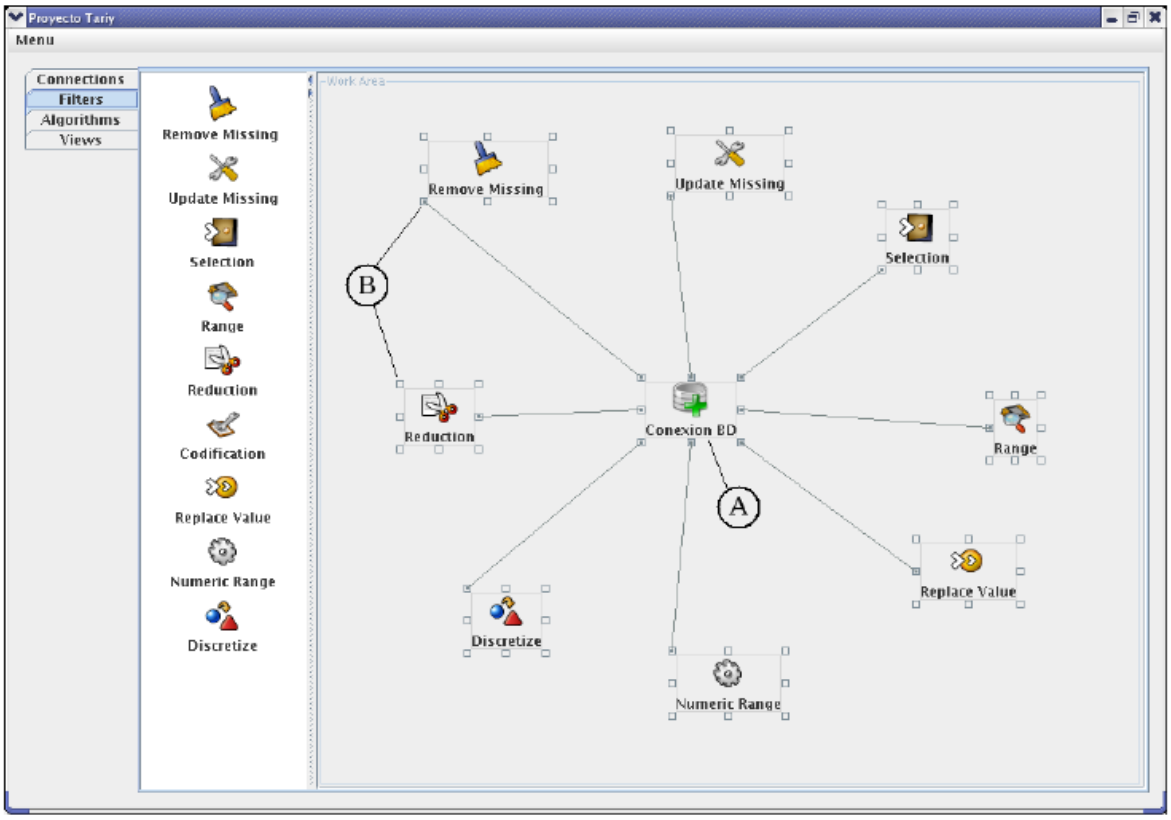
Figura C.7: Filtro Remove Missing



ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
1. El usuario hace click sobre uno de los ıconos del módulo A: ’Filtros’.	2. En el área de opciones del módulo aparecen los 9 ıconos correspondientes a los filtros.
3. El usuario hace click sobre uno de los ıconos correspondientes a los filtros.	4. El ıcono correspondiente aparece en el área de trabajo B.

Conexión filtros a BD

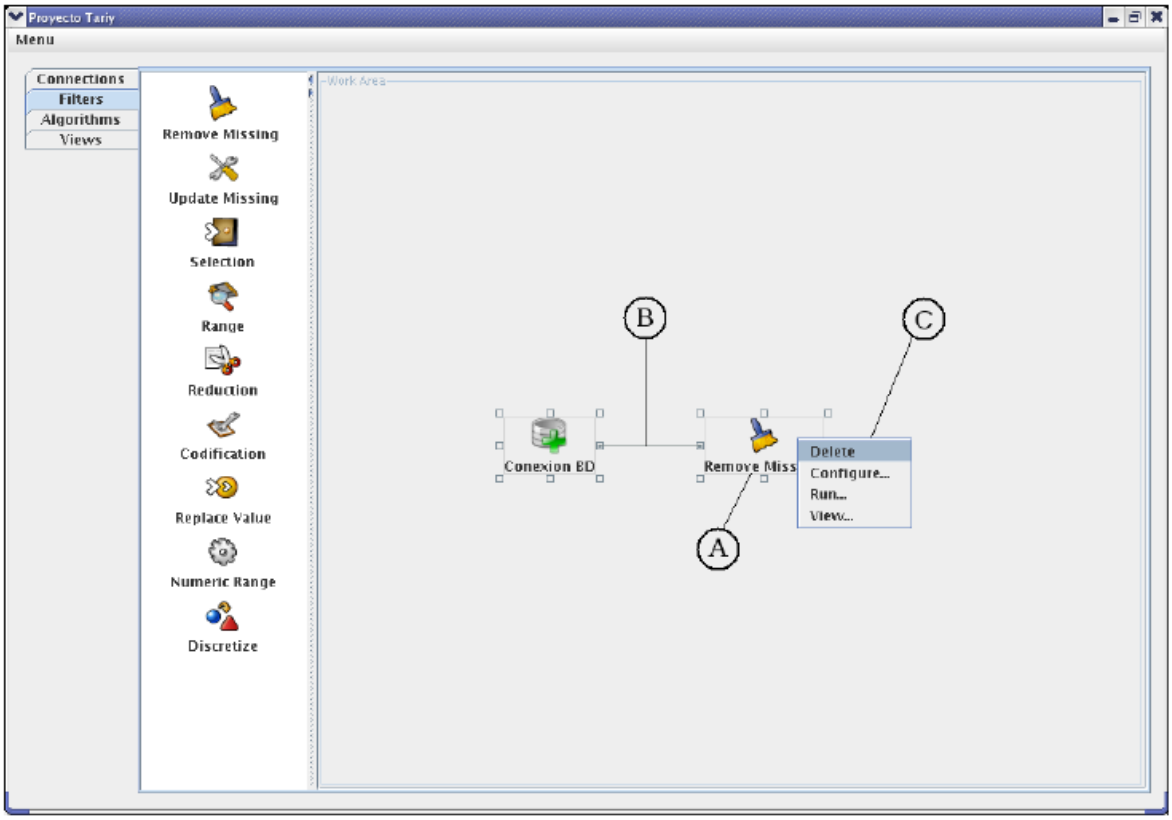
Figura C.7.1: Conexión filtros a BD



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario conecta una base de datos a alguno o varios de de los filtros A.	2. Los ıconos pueden ser conectados por medio de una línea B.

Menú emergente de filtros

Figura C.7.2: Menú emergente de filtros



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click sobre el ıcono ’Remove Missing’.	2. El ıcono aparece en el área de trabajo A.
3. El usuario conecta el filtro a la base de datos.	4. Aparece un hilo que conecta los ıconos B.
5. EL usuario hace click derecho sobre el filtro.	6. Aparece el menu emergente del ıcono C. La opción Delete, borra el filtro del área de trabajo. Este filtro no tiene ventana de configuración. La opción ’Run’ ejecuta la aplicación del filtro. La opción ’View’ muestra la ventana de vizualización de datos que será descrita en el siguiente caso de uso.

Visualización de datos filtrados

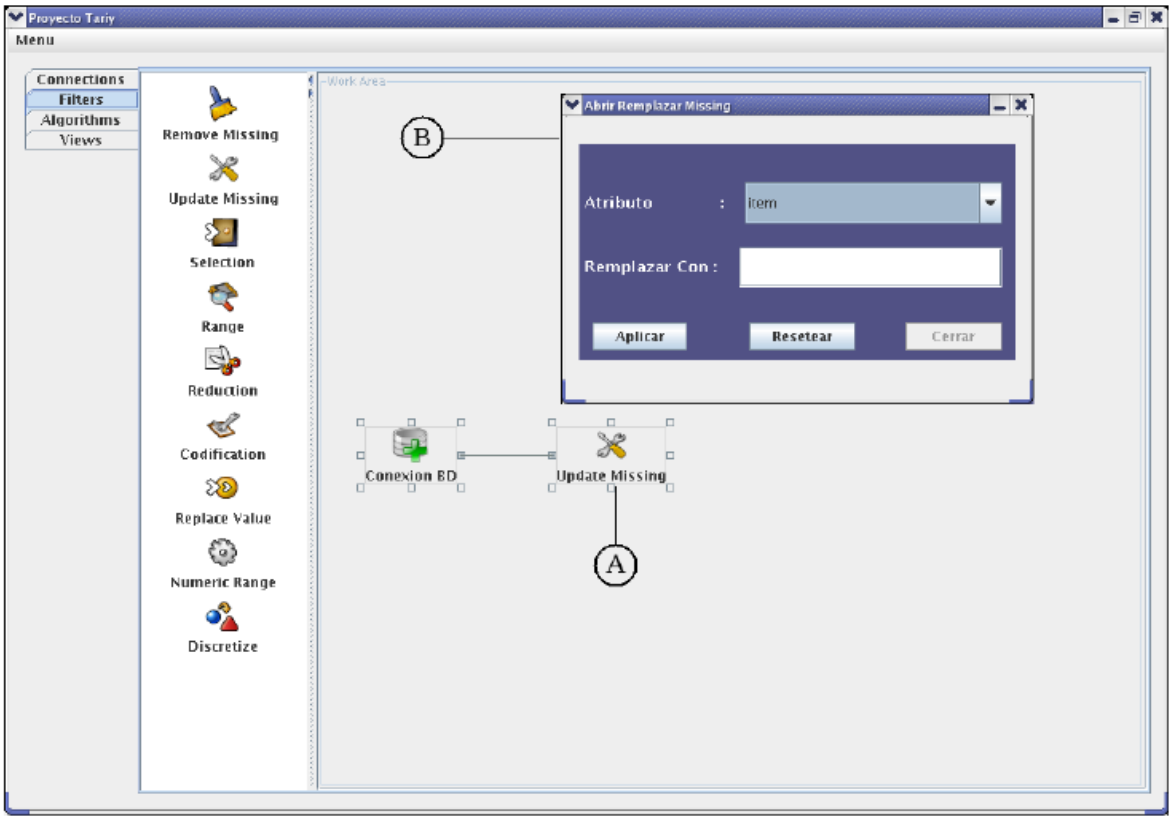
Figura C.7.3.1: Visualización de datos filtrados

Figura C.7.3.2: datos de entrada

ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
1. El usuario hace click sobre la opción ’View’ del menu desplegable filtro en el área de trabajo .	2. Aparece la ventana de vizualización de datos filtrados y no filtrados. Los campos son, A: Variables o nombres de los campos de la tabla. B: Datos de entrada que son los datos que llegaron al filtro inicialmente. C: Datos filtrados que son el resultado de haber aplicado el filtro. D: nmero de registros eliminados al aplicar el filtro. E: Número de registros después de aplicar el filtro. En la figura 16 se ve la grilla sobre la que se muestran los datos en el caso ’Datos de entrada’

Configuración filtro Update Missing

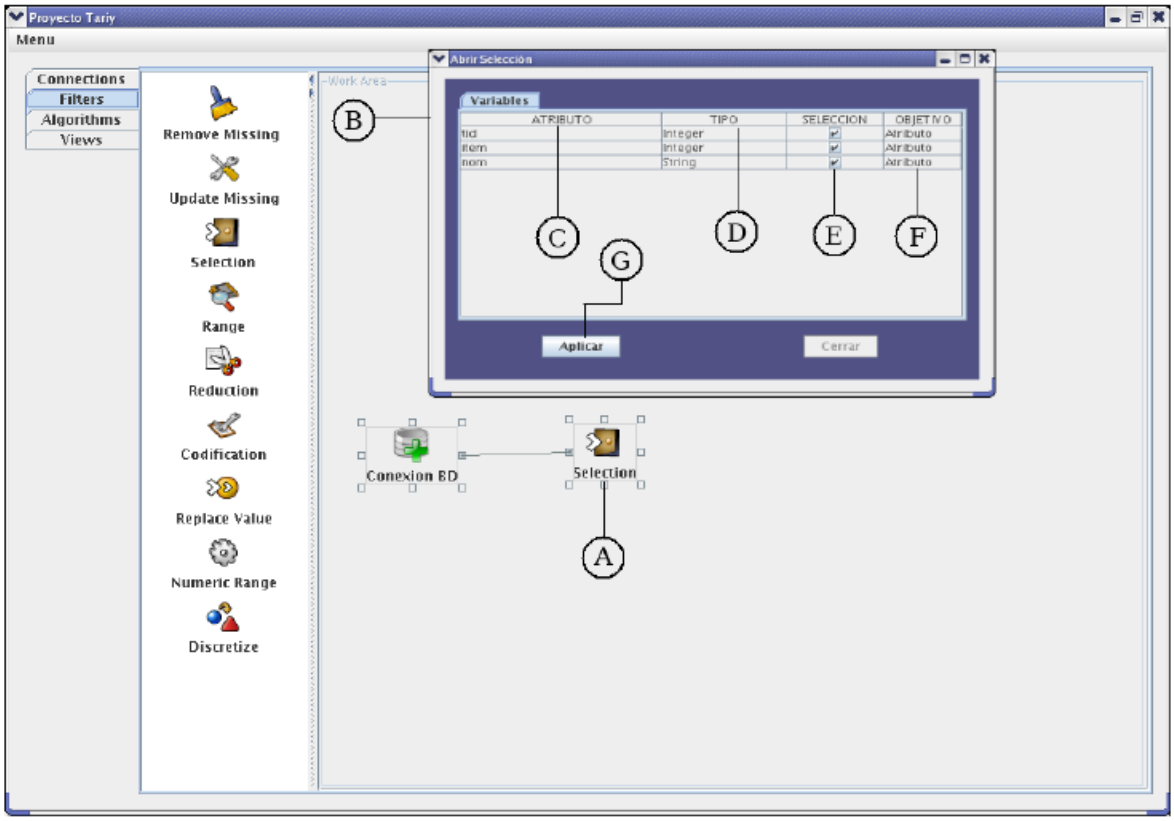
Figura C.7.4: Configuración filtro Update Missing



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra B la ventana de configuración correspondiente al filtro ’Update Missing’. Los campos son: Atributo, en el cual se escribe el nombre del atributo a buscar en el conjunto de datos. Reemplazar con, aqui se escribe el nuevo valor del atributo.

Configuración filtro Selection

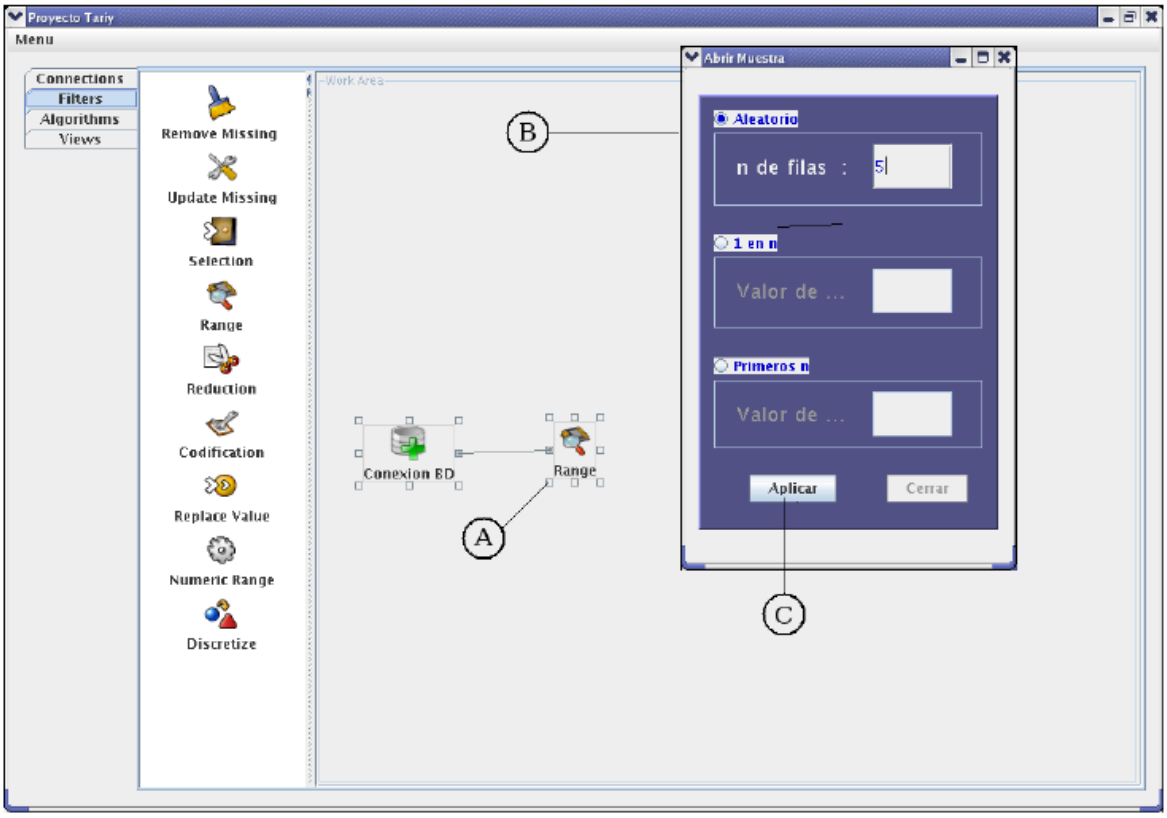
Figura C.7.5: Configuración filtro Selection



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra B la ventana de configuración correspondiente al filtro ’Selection’. Los campos son: C: Atributo, en esta grilla se muestran los nombres de los atributos seleccionados. D: Tipo, muestra el tipo de datos de los atributos. E: cajas de verificación para escoger los atributos a utilizar. F: es posible escoger un atributo clase haciendo click sobre estos campos. Esto es util en experimentos de clasificación. G: el botón ’Aplicar’ debe ser presionado para que el filtro sea aplicado.

Configuración filtro Range

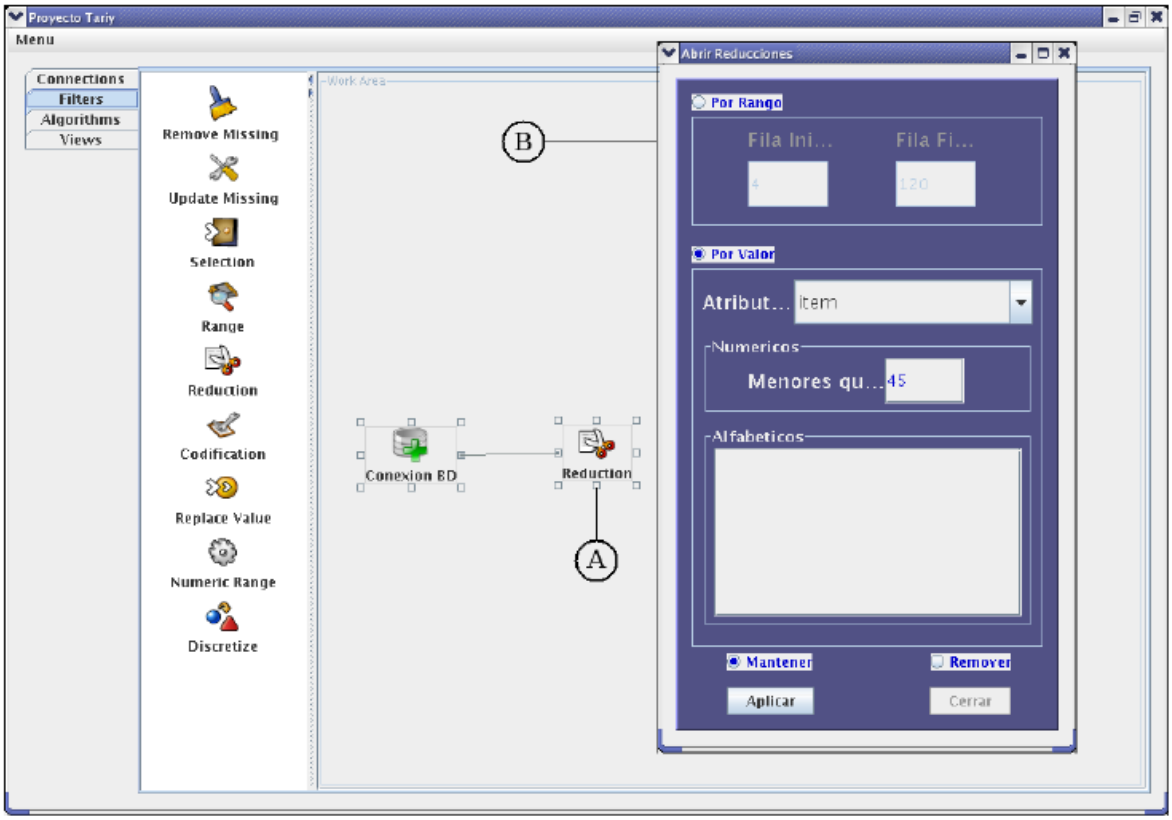
Figura C.7.6: Configuración filtro Range



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra B la ventana de configuración correspondiente al filtro ’Range’. Los campos son:Aleatorio, en donde se escribe el número n de filas que se desea sean escogidas aleatoriamente. 1 en n, donde n es el periodo utilizado para seleccionar los datos a utilizar. Primeros n, donde n es el número campos a incluir en la selección a partir del primero.

Configuración filtro Reduction

Figura C.7.7: Configuración filtro Reduction

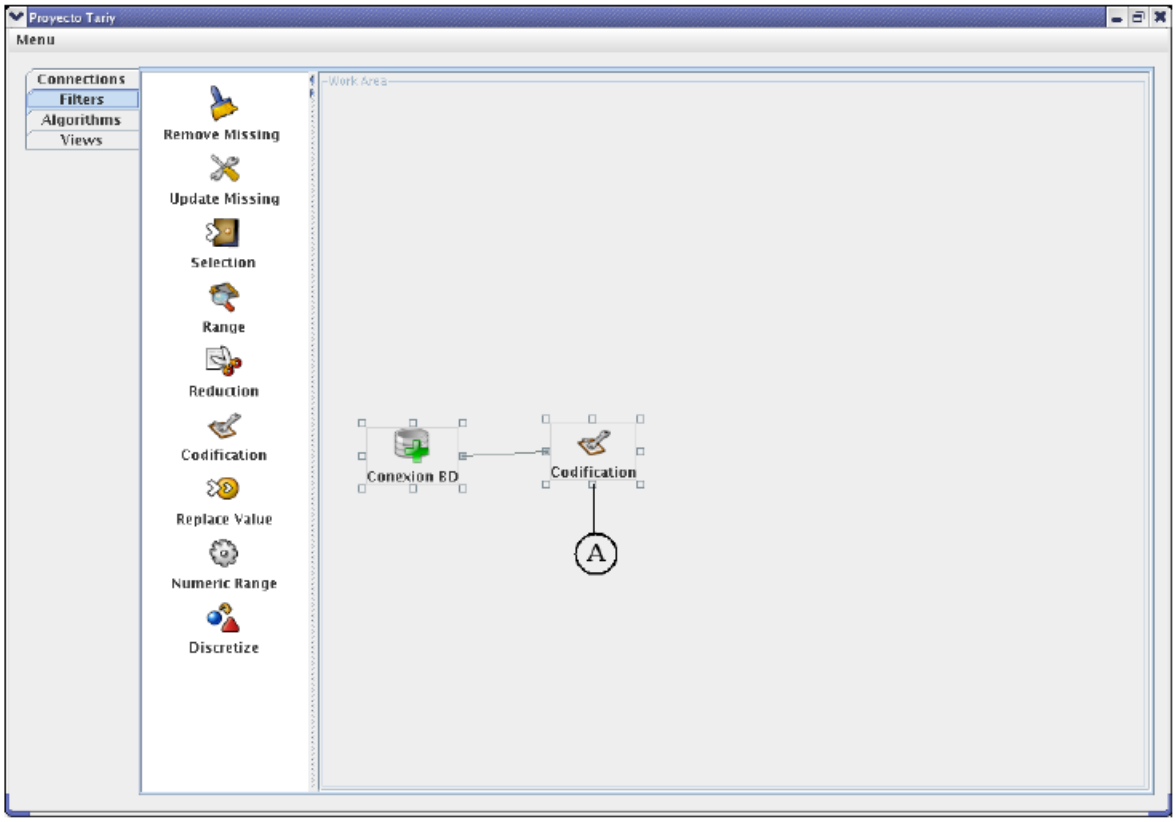


ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra la ventana B de configuración correspondiente al filtro ’Reduction’. Los campos son: Por rango, los campos son ’Fila inicial’ donde se escribe la fila a partir de la cual inicia el rango y ’Fila final’ que es el límite superior del rango. Por Valor: Se elige el nombre del atributo y luego en caso de que los valores a quitar sean numéricos en el campo ’Menores que’ se especifica el número a partir del cual se hace la reducción. Si el atributo es alfabético se escribe su valor en el área de texto y en las casillas de selección se especifica si ese valor se

ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
	desea ’Mantener’ o ’Remover’. Aplicar: ejecuta el filtro.

Configuración filtro Codification

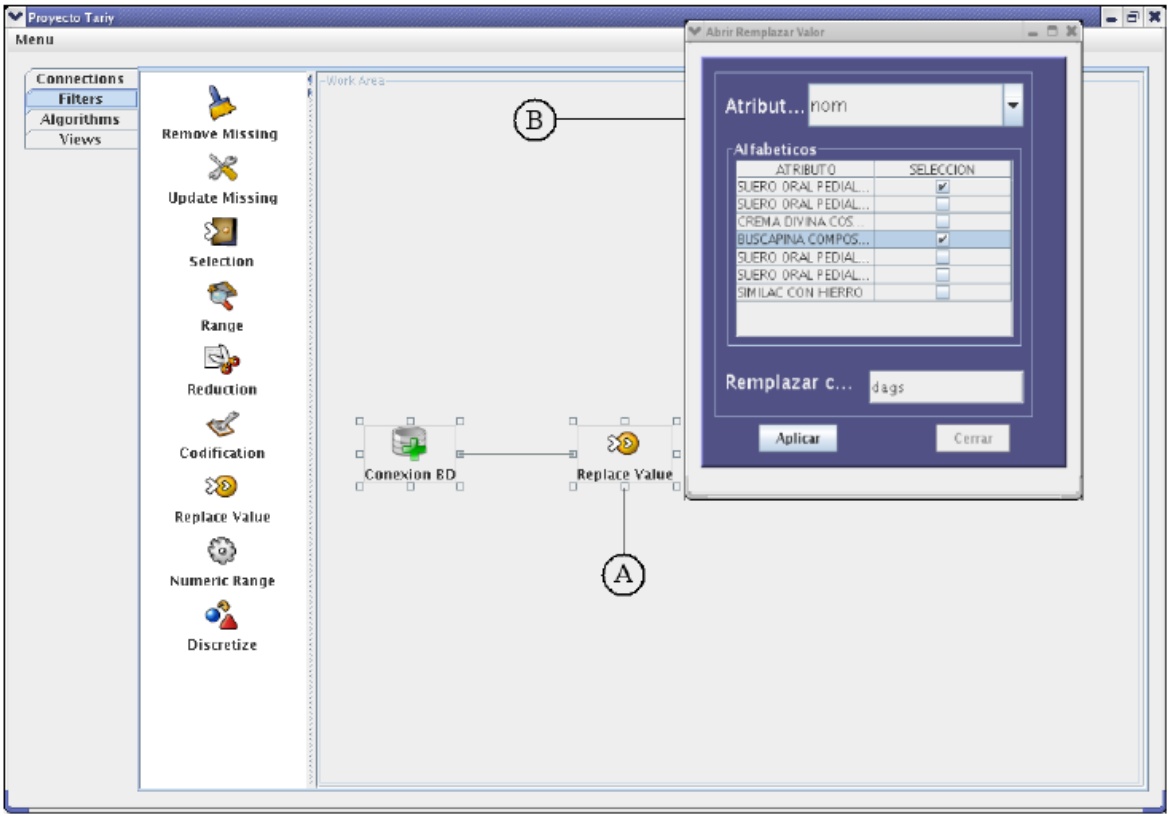
Figura C.7.8: Configuración filtro Codification



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra la ventana de configuración correspondiente al filtro ’Codifcation’. Este filtro no tiene ventana de configuración. Se aplica para asignar un número a valores alfabéticos.

Configuración filtro Replace Value

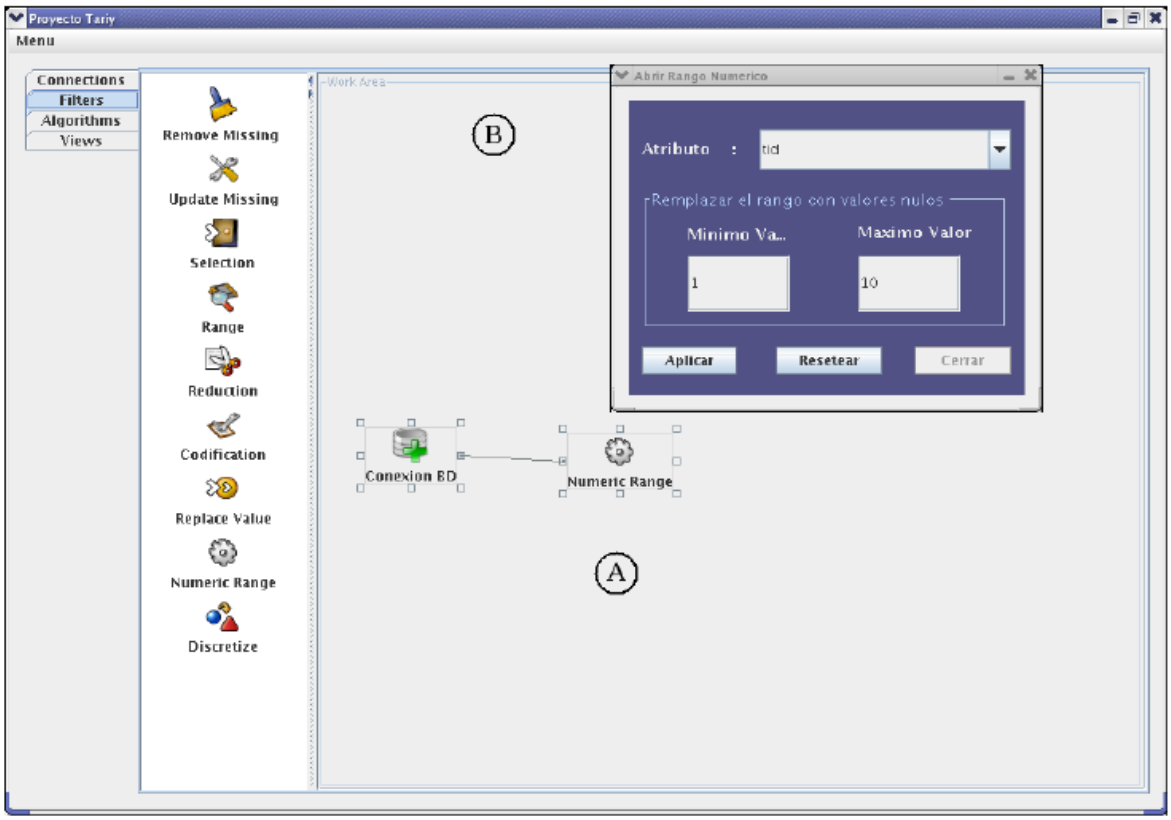
Figura C.7.9: Configuración filtro Replace Value



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra la ventana de configuración correspondiente al filtro ’Replace Value’. Los campos son: Atributo, en el cual se elige el nombre del atributo a buscar en el conjunto de datos. Reemplazar con, aqui se escribe el nuevo valor del atributo. Aplicar: ejecuta el filtro.

Configuración filtro Numeric Range

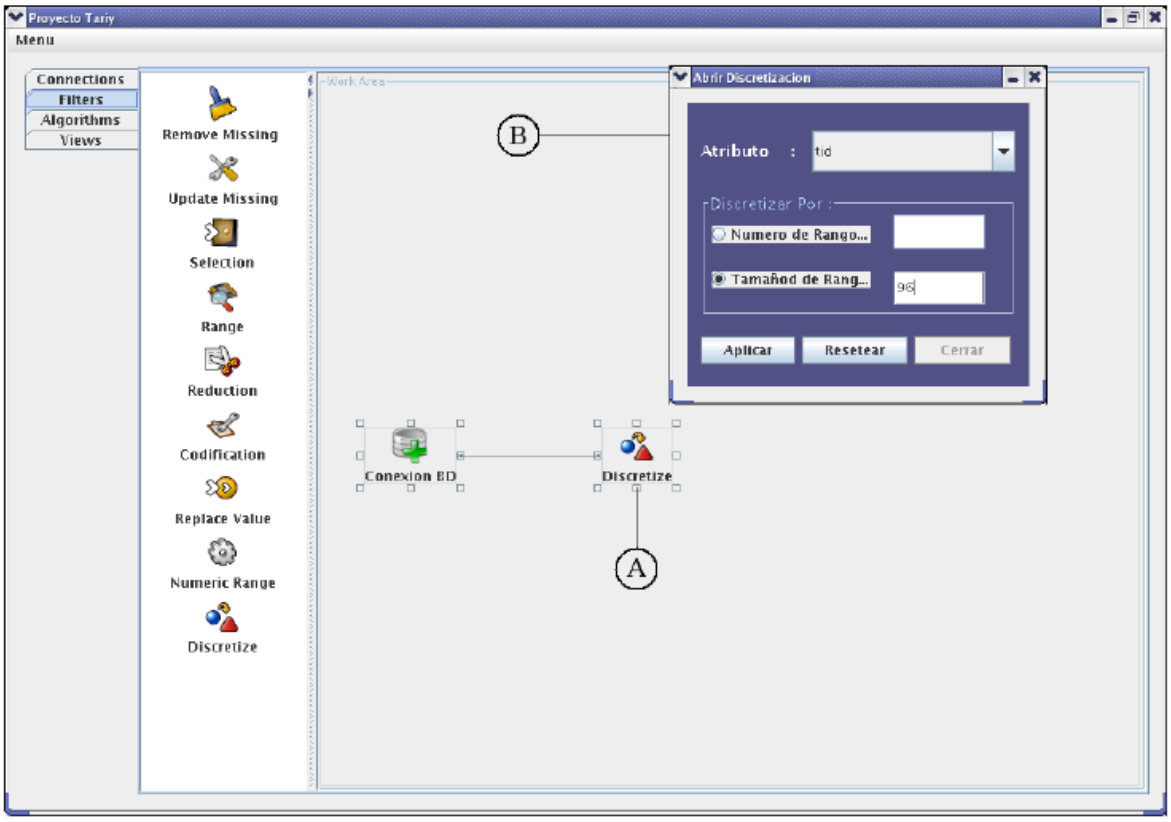
Figura C.7.10: Configuración filtro Numeric Range



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración'	2. Se muestra la ventana B de configuración correspondiente al filtro ’Numeric Range’. Los campos son: Atributo, en el cual se escribe el nombre del atributo a discretizar de tipo numérico. Reemplazar rango con valores nulos: aqui es posible especificar un rango de datos que seran convertidos a núlos. Mímimo valor: límite inferior del rango. Mínimo valor: límite superior del rango. Aplicar: ejecuta el filtro. Resetear: deja los campos en blanco.

Configuración filtro Discretize

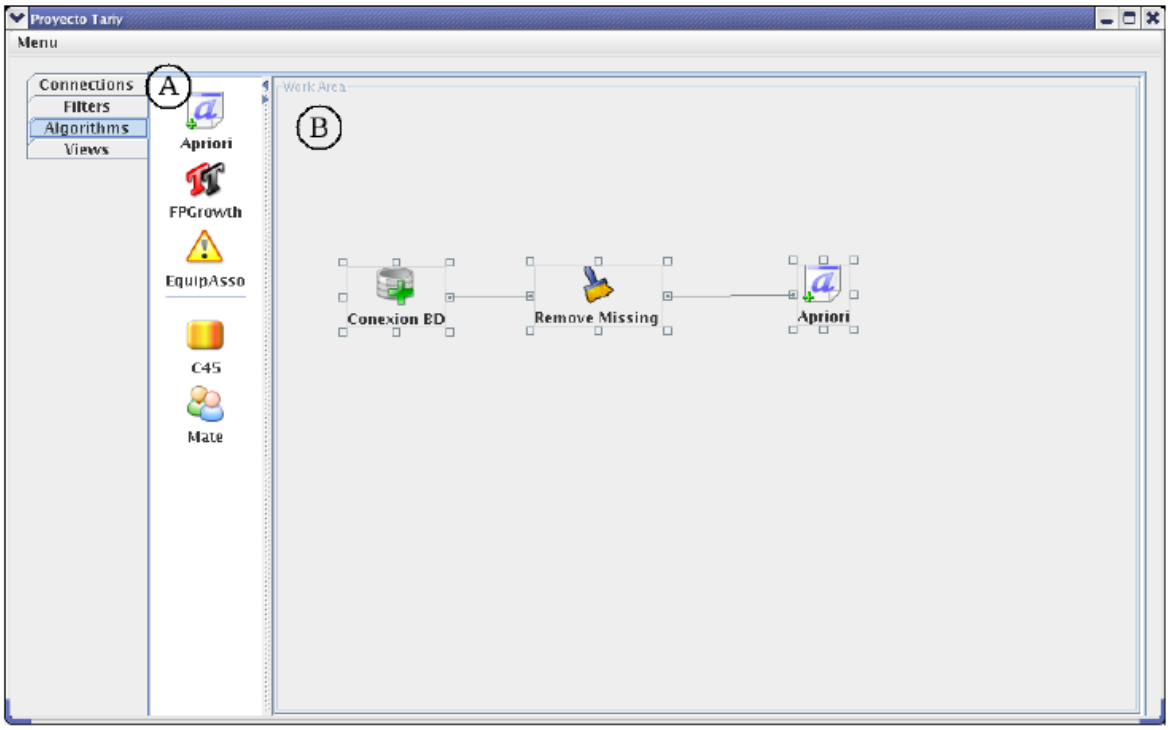
Figura C.7.11: Configuración filtro Discretize



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el filtro A y elige la opción ’Configuración’	2. Se muestra la ventana B de configuración correspondiente al filtro ’Discretize’. Los campos son: Atributo, en el cual se escribe el nombre del atributo a discretizar. Discretizar por: ’Número de rango’: se puede establecer el número de rangos a crear. ’Tamaño del rango’: se especifica el tamaño del rango Aplicar: ejecuta el filtro. Resetear: deja los campos en blanco

C.5 Algoritmos

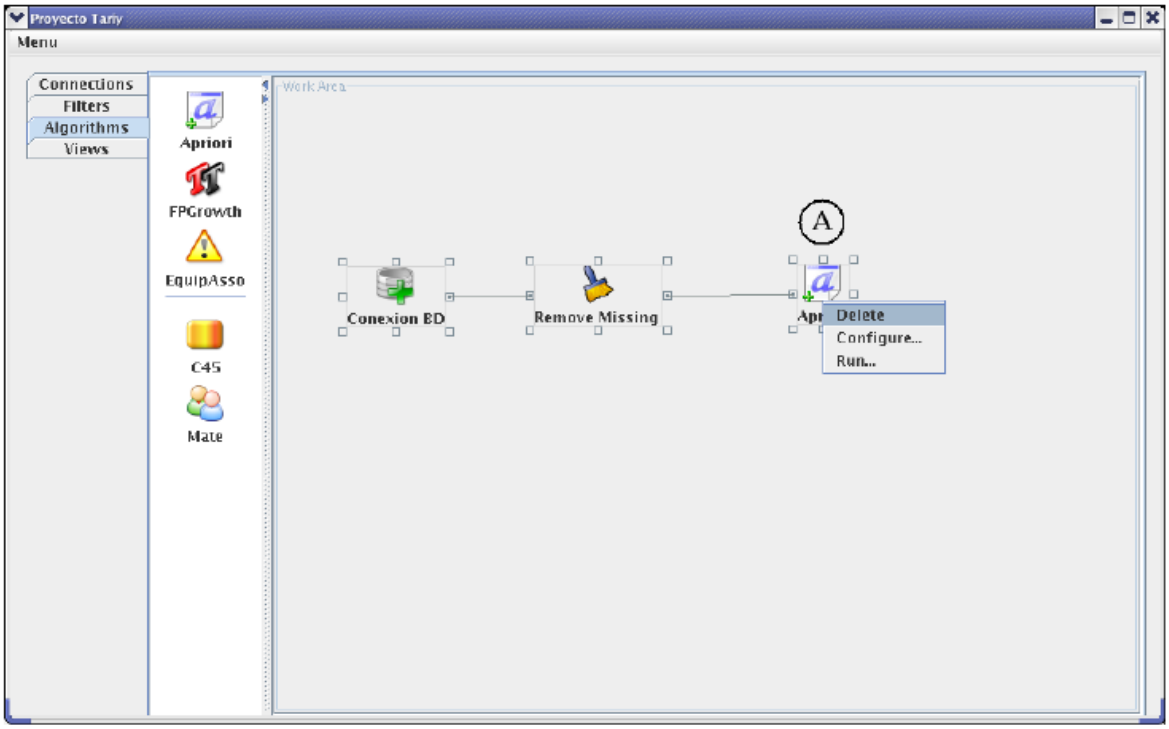
Figura C.8.1: Algoritmo Apriori



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. Si el usuario quiere minar los datos con Apriori y presiona sobre A (Area de opciones), en el icono respectivo.	2. En B (Area de trabajo) aparece el icono del algoritmo Apriori.

Opción Delete

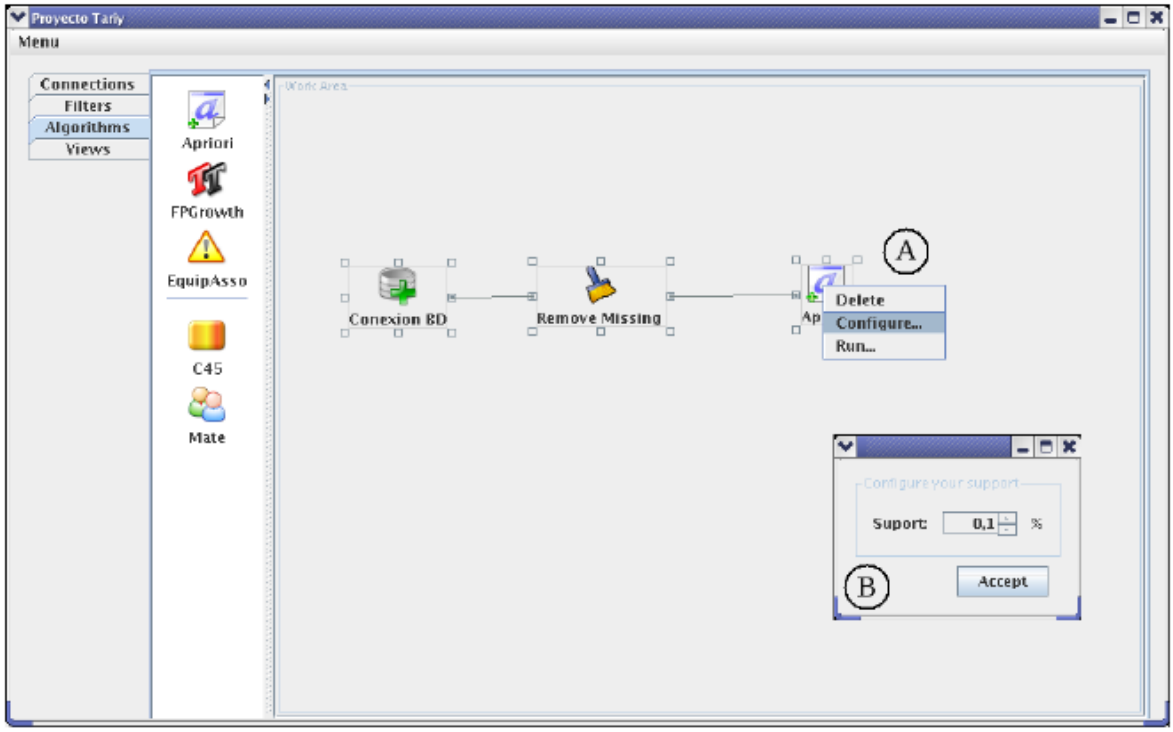
Figura C.8.1.1: Opción Delete



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre A: el icono del algoritmo (cualquiera que este sea, Apriori, EquipAsso, FPGrowth, MateBy o C4.5) y elige la opción delete del menú de configuración.	2. El icono del algoritmo es borrado del área de trabajo.

Opción Configure

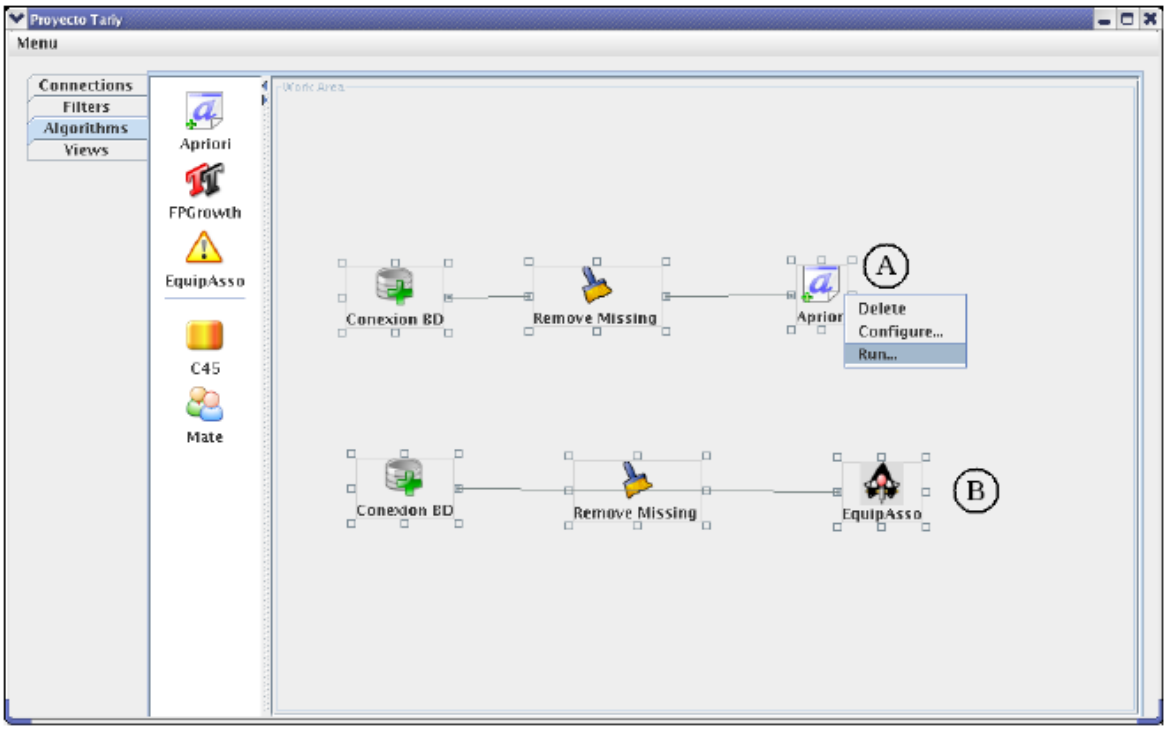
Figura C.8.1.2: Opción Configure



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre A: el icono del algoritmo (cualquiera que este sea, Apriori, EquipAsso, FPGrowth, MateBy o C4.5) y elige configurar sus parametros.	2. Sobre el área de trabajo aparece una ventana B, para que el usuario configure el soporte del algoritmo.

Opción Run

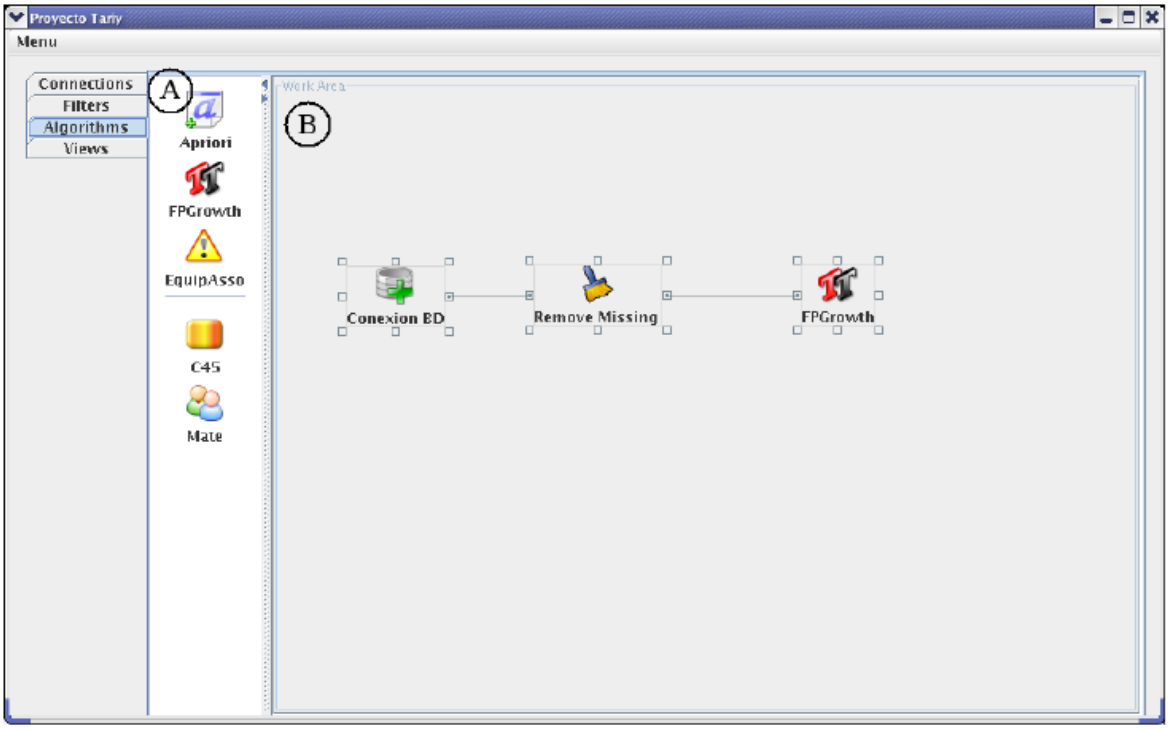
Figura C.8.1.3: Opción Run



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre A: el icono del algoritmo (cualquiera que este sea, Apriori, EquipAsso, FPGrowth, MateBy o C4.5) y elige la opción run.	2. El icono del algoritmo cambia por una animación, así como se muestra en B.

Algoritmo FPGrowth

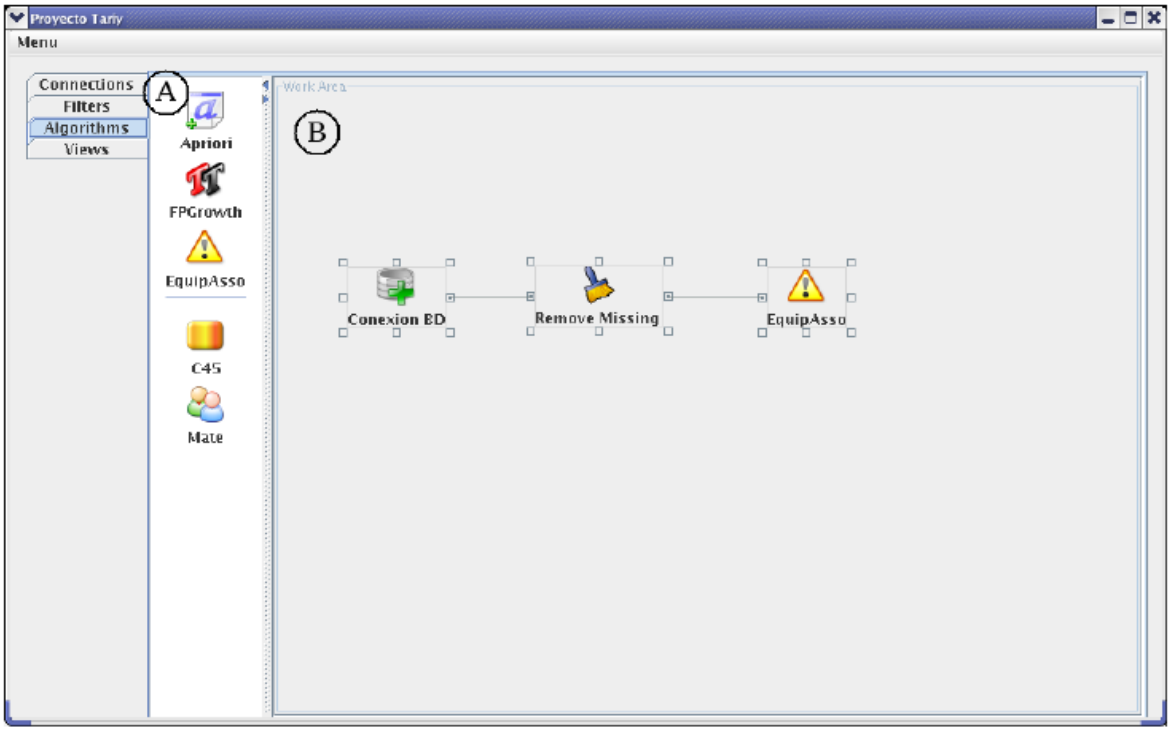
Figura C.8.2: Algoritmo FPGrowth



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. Si el usuario quiere minar los datos con FPGrowth y presiona sobre A (Area de opciones), en el icono respectivo.	2. En B (Area de trabajo) aparece el icono del algoritmo FPGrowth.

Algoritmo EquipAsso

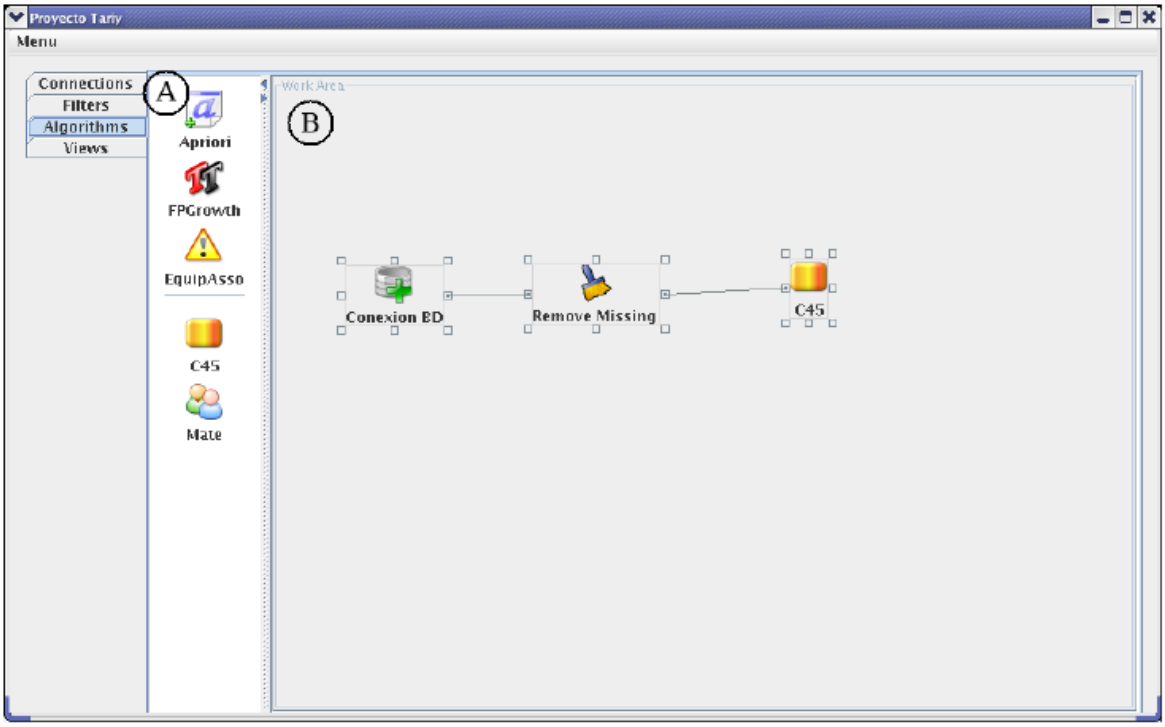
Figura C.8.3: Algoritmo EquipAsso



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. Si el usuario quiere minar los datos con EquipAsso y presiona sobre A (Area de opciones), en el icono respectivo.	2. En B (Area de trabajo) aparece el icono del algoritmo EquipAsso.

Algoritmo C4.5

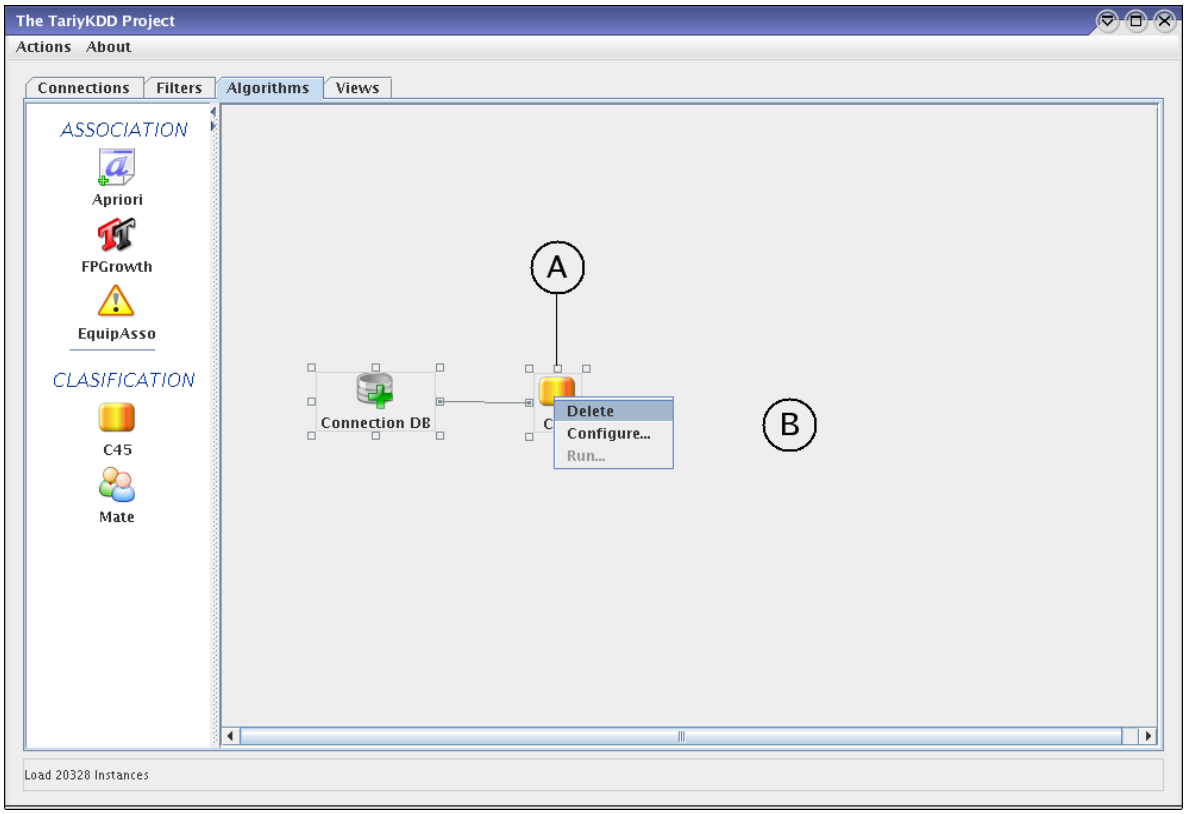
Figura C.8.4: Algoritmo C4.5



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. Si el usuario quiere minar los datos con C4.5 y presiona sobre A (Area de opciones), en el icono respectivo.	2. En B (Area de trabajo) aparece el icono del algoritmo C4.5.

Eliminar el icono “C45” del área de trabajo

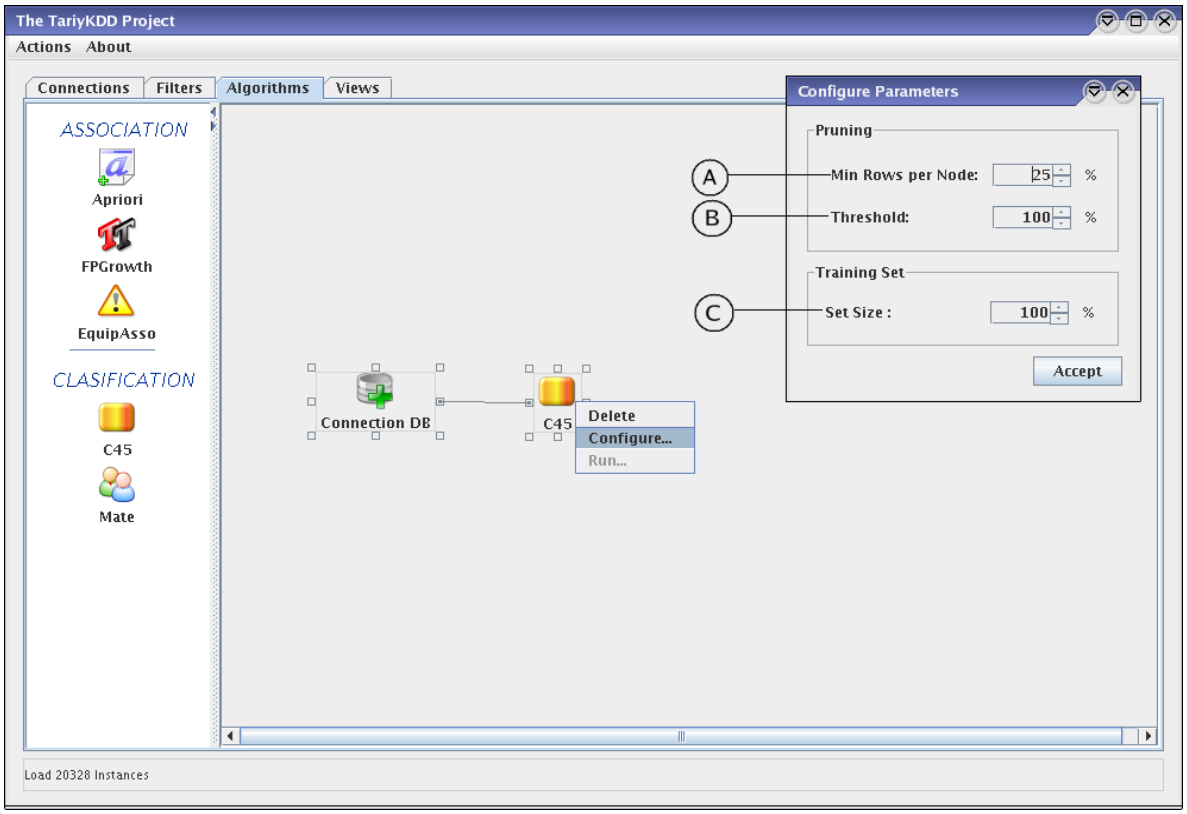
Figura C.8.4.1: opción “Delete” del icono “C45”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’C45’ y selecciona la opción “Delete”.	2. El ıcono A ’C45’ es borrado del área de trabajo B.

Configuración del algoritmo “C45”

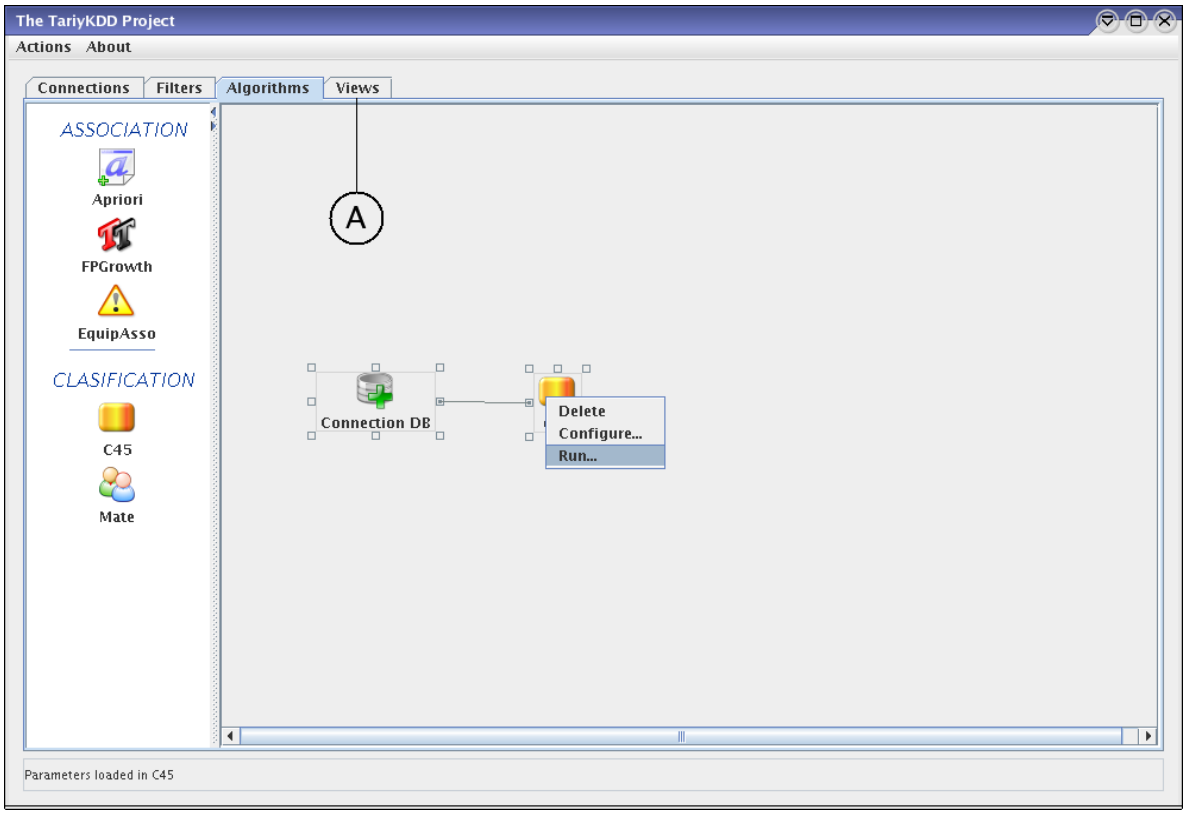
Figura C.8.4.2: Opción “Configure” del icono “C45”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’C45’ y selecciona la opción “Configure”.	2. Es desplegada la ventana de configuración del algoritmo “C45” compuesta por: los campos de poda del árbol A “Min Rows per Node” en el cual se especifica el porcentaje del número mínimo de filas en el conjunto de datos, en las que tiene que aparecer un nodo del ábol para que no sea parcializado hacia la clase de mayor ocurrencia hasta ese momento, y B “Threshold” umbral o porcentaje en el que un nodo se parcializa hacia la clase que logre pasar ese umbral. En el campo “Training Set” se introduce C “Set Size” tamaño del conjunto de entrenamiento que usará el algoritmo. El modelo generado será aplicado al porcentaje restante de datos.

Ejecución del algoritmo C45

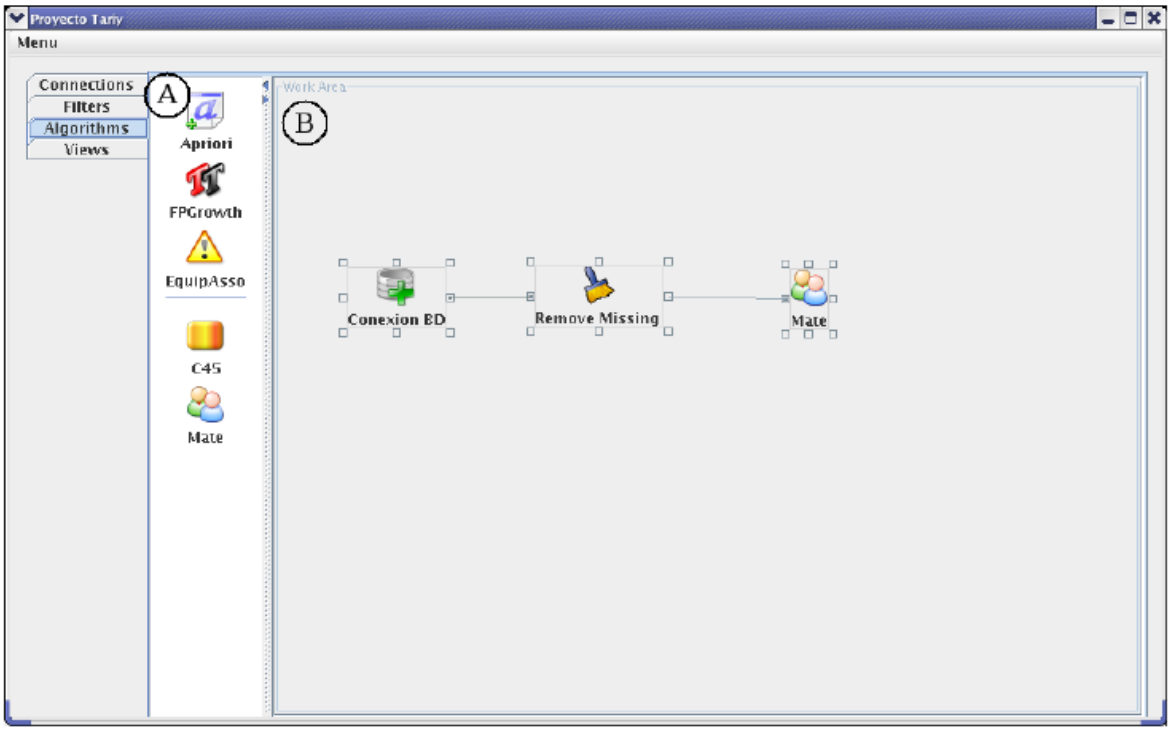
Figura C.8.4.3: Opción “Run” del icono “C45”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’C45’ y selecciona la opción “Run”.	2. El algoritmo es ejecutado teniendo en cuenta los parámetros de configuración. Se genera un árbol y reglas que pueden ser visualizados a través de las herramientas para visualización de árboles del panel A “View”.

Algoritmo Mate

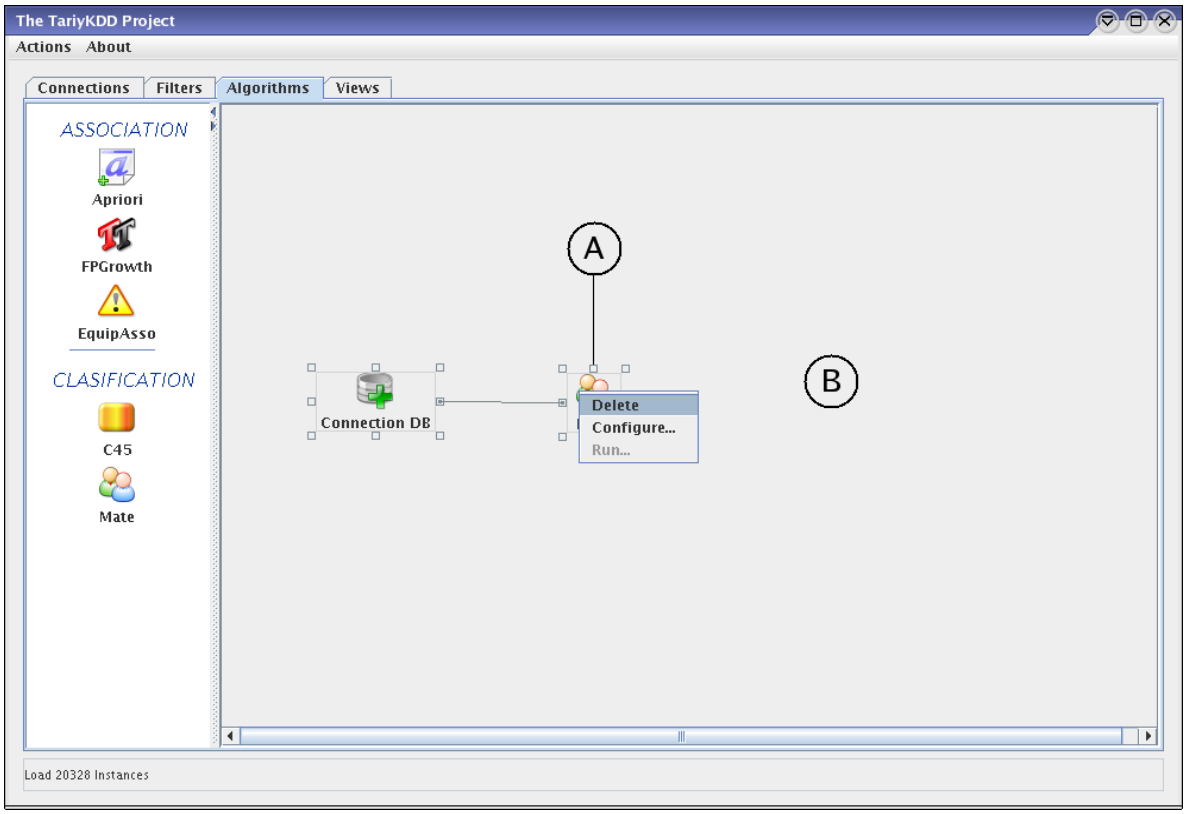
Figura C.8.5: Algoritmo Mate



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. Si el usuario quiere minar los datos con Mate y presiona sobre A (Area de opciones), en el icono respectivo.	2. En B (Area de trabajo) aparece el icono del algoritmo Mate.

Eliminar el icono “Mate” del área de trabajo

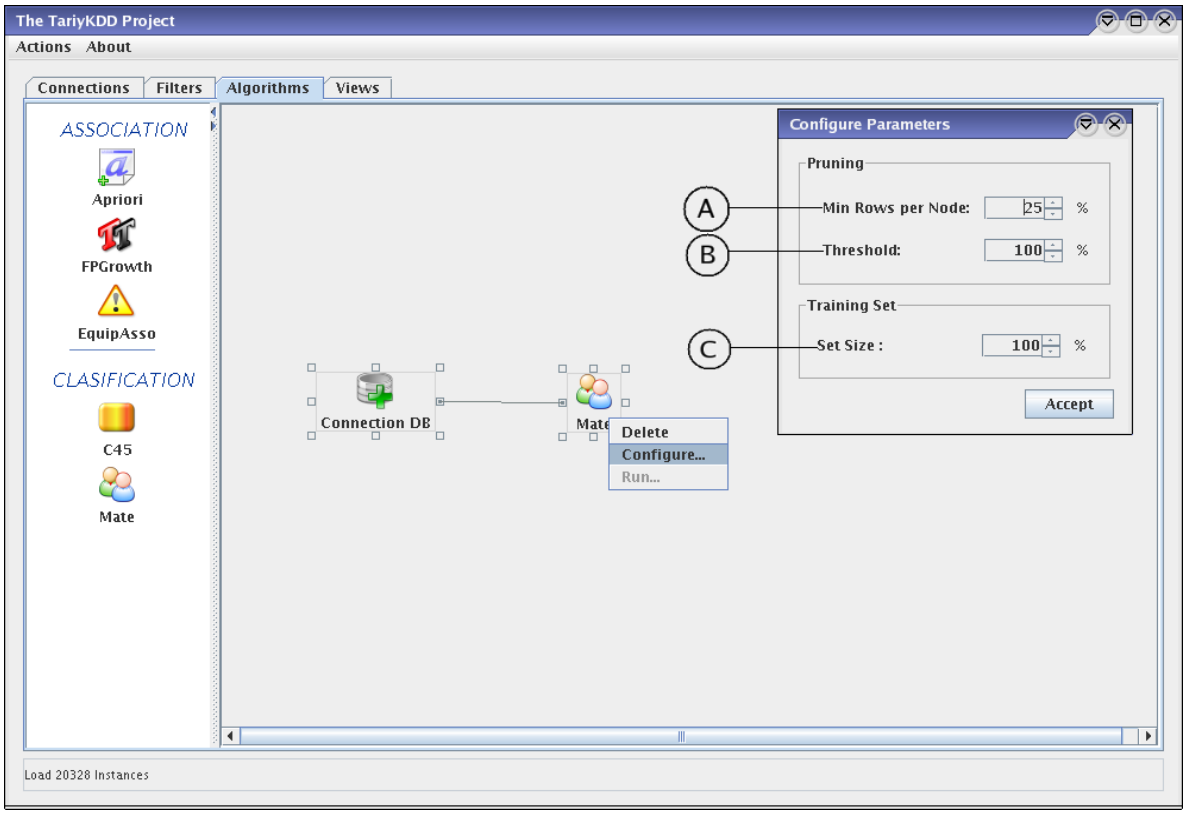
Figura C.8.5.1: opción “Delete” del icono “Mate”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Mate’ y selecciona la opción “Delete”.	2. El ıcono A ’Mate’ es borrado del área de trabajo B.

Configuración del algoritmo “Mate”

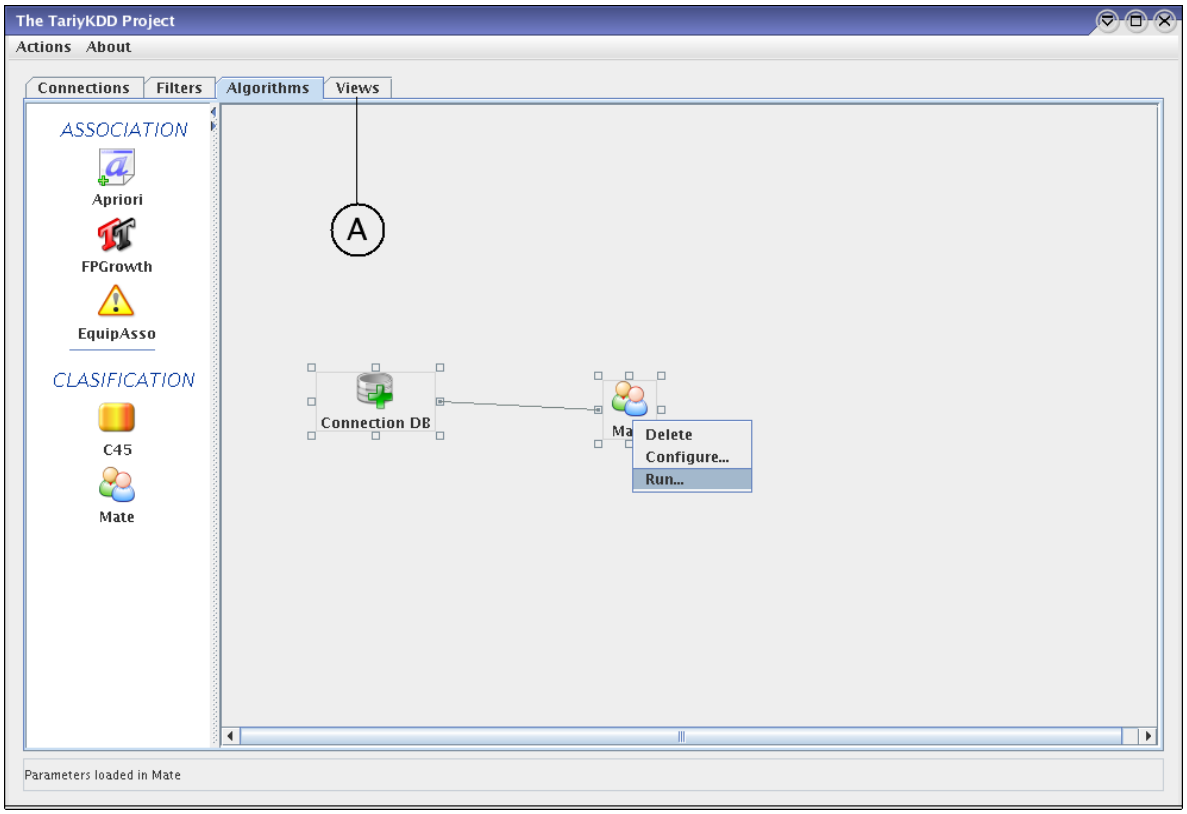
Figura C.8.5.2: Opción “Configure” del icono “Mate”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Mate’ y selecciona la opción “Configure”.	2. Es desplegada la ventana de configuración del algoritmo “Mate” compuesta por: los campos de poda del árbol A “Min Rows per Node” en el cual se especifica el porcentaje del número mínimo de filas en el conjunto de datos, en las que tiene que aparecer un nodo del ábol para que no sea parcializado hacia la clase de mayor ocurrencia hasta ese momento, y B “Threshold” umbral o porcentaje en el que un nodo se parcializa hacia la clase que logre pasar ese umbral. En el campo “Training Set” se introduce C “Set Size” tamaño del conjunto de entrenamiento que usará el algoritmo. El modelo generado será aplicado al porcentaje restante de datos.

Ejecución del algoritmo Mate

Figura C.8.5.3: Opción “Run” del icono “Mate”

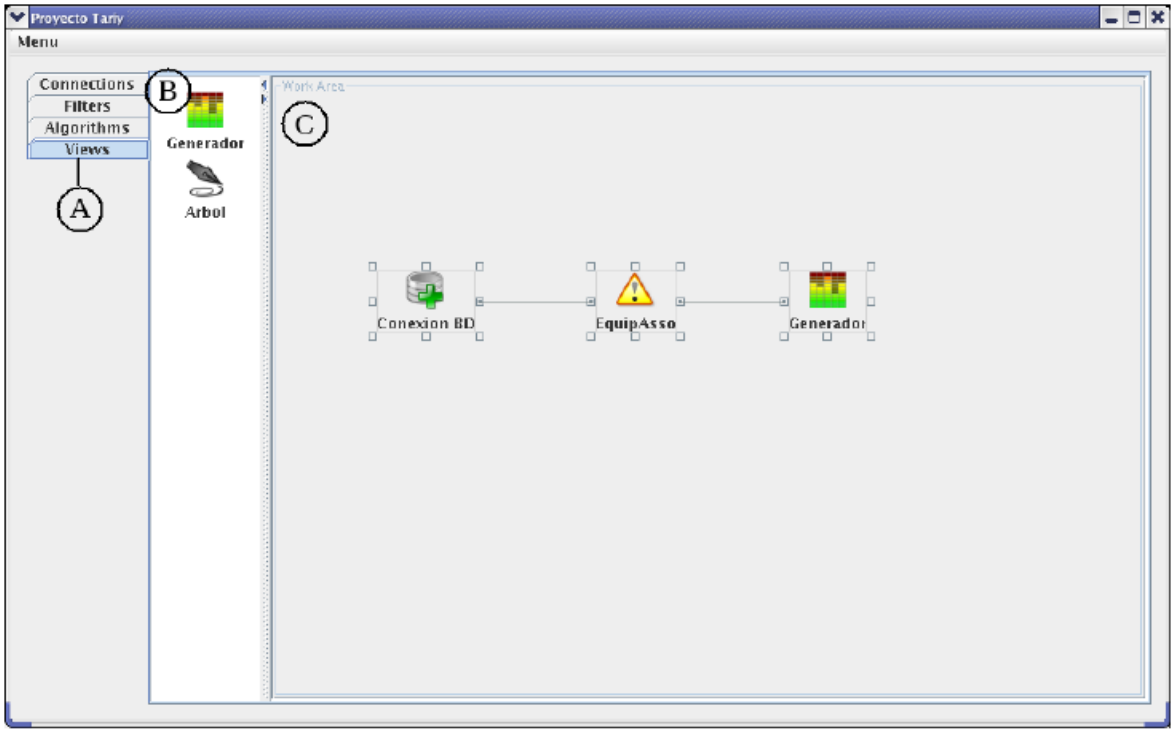


ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el ıcono ’Mate’ y selecciona la opción “Run”.	2. El algoritmo es ejecutado teniendo en cuenta los parámetros de configuración. Se genera un árbol y reglas que pueden ser visualizados a través de las herramientas para visualización de árboles del panel A “View”.

C.6 Visualización

Diagrama de Visualización Asociación

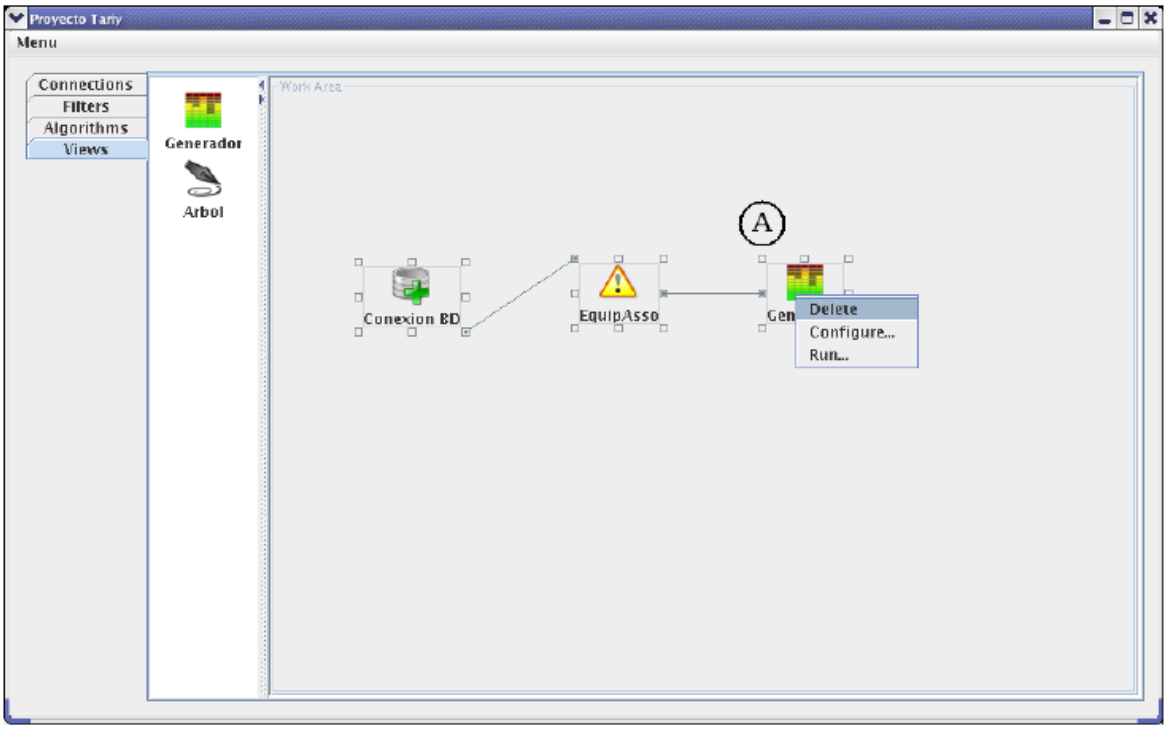
Figura C.9: Diagrama de Visualización



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. Cuando el usuario ha construido una secuencia de Minería de Datos, con cualquiera de los algoritmos, en A se encuentra en la sección de vistas y en B (Area de opciones) ha hecho click en el icono generador.	2. Entonces en C (Area de trabajo) aparece el icono del generador, a través del cual el usuario puede acceder a las opciones de este módulo.

Opción Delete

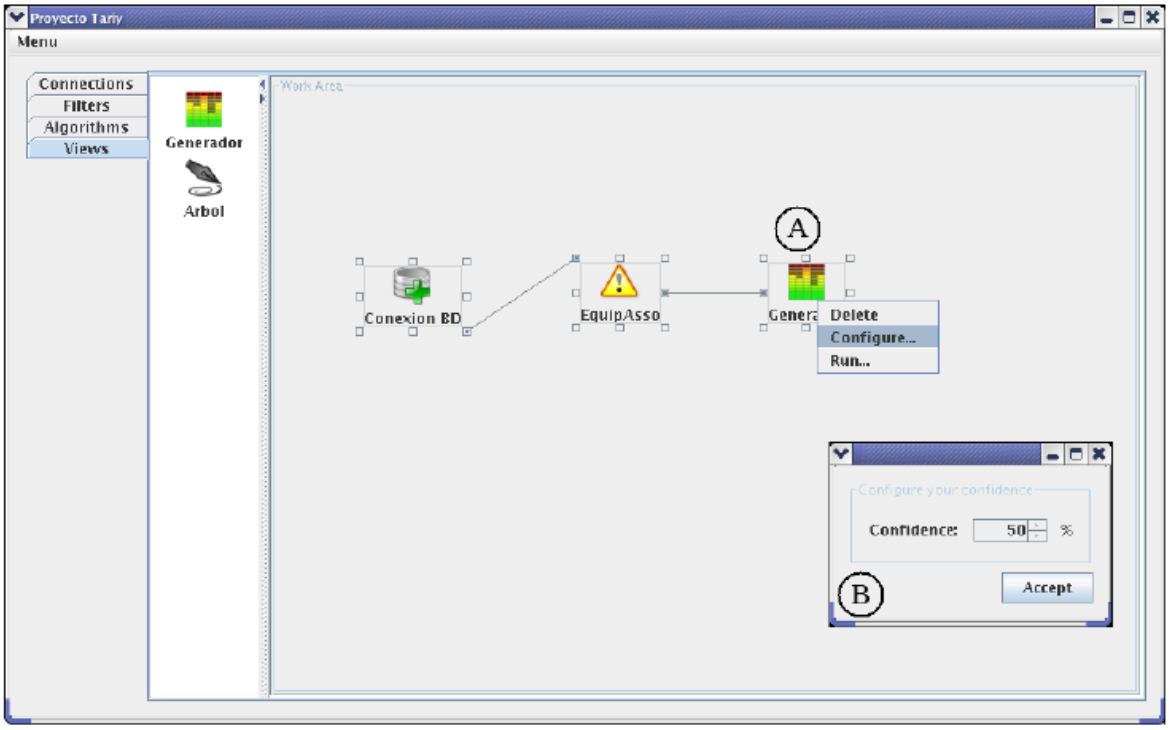
Figura C.9.1: Opción Delete



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre A: el icono generador y elige la opción Delete.	2. El icono desaparece del área de trabajo, esperando un nuevo icono en la secuencia de Minería de Datos.

Opción Configure

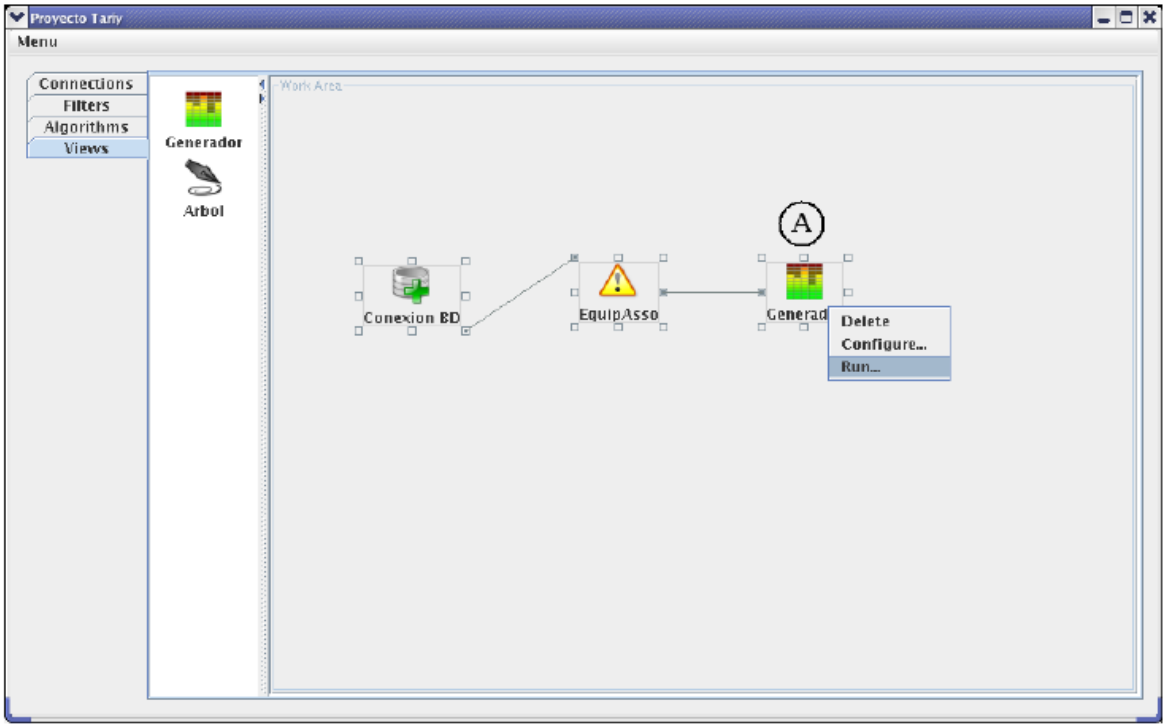
Figura C.9.2: Opción Configure



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click sobre A: el icono del generador y elige configurar sus parametros.	2. Sobre el área de trabajo aparece una ventana B, para que el usuario configure la confianza con la cual se van a filtrar las reglas de asociación.

Opción Run

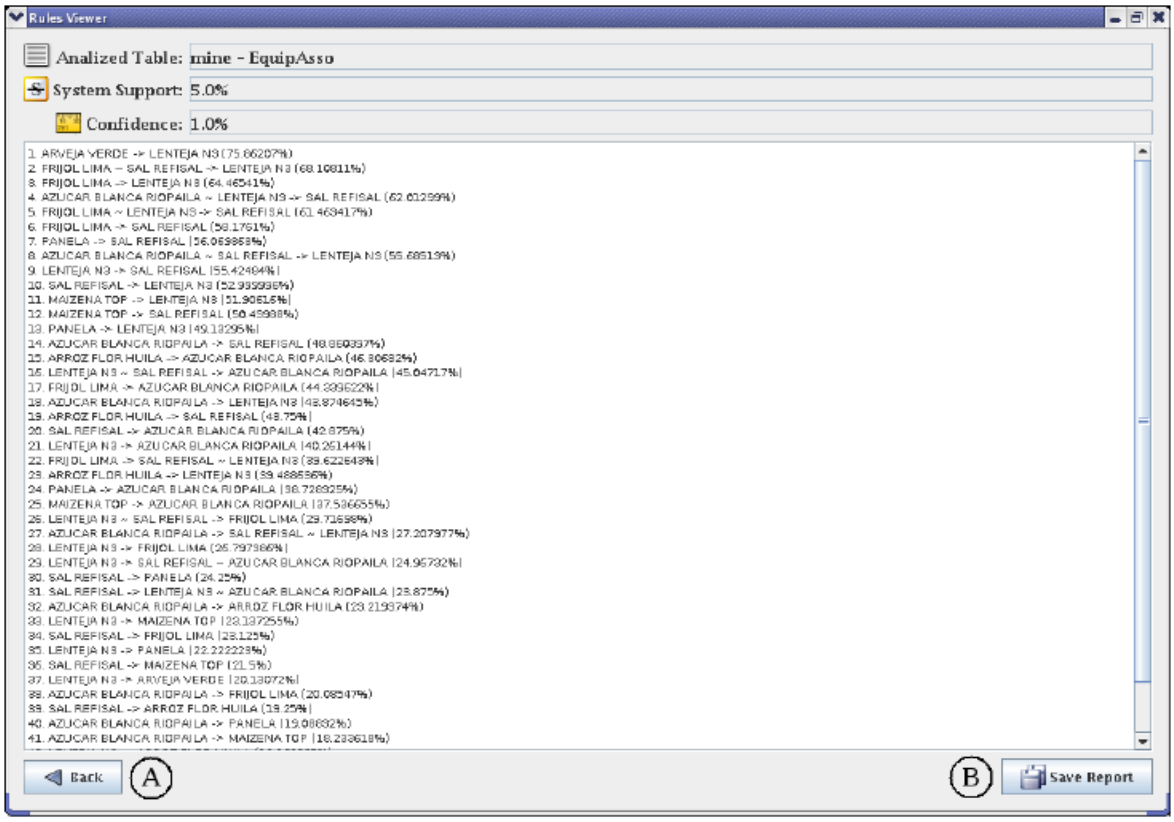
Figura C.9.3: Opción Run



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre A: el icono generador y elige la opción Run.	2. En el área de trabajo aparece una ventana B, con las reglas obtenidas a partir de los algoritmos de Minería de Datos (La cual se explica en la siguiente figura).

Visor de Reglas

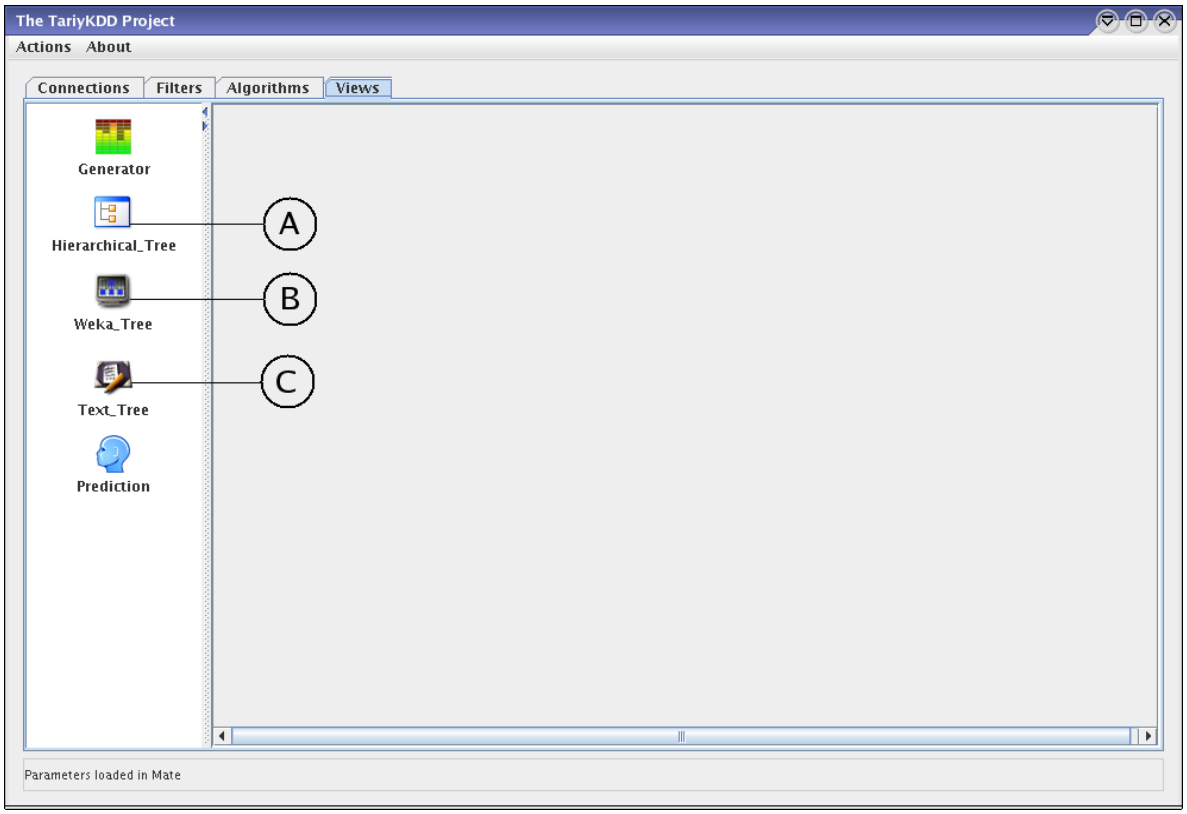
Figura C.9.4: Visor de Reglas



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario tiene la opción de hacer click en A, o en B.	2. Al hacer click en A, la ventana de reglas desaparece y si hace click en B el usuario tiene la opción de guardar el reporte de las reglas de asociación.

Visualización Clasificación

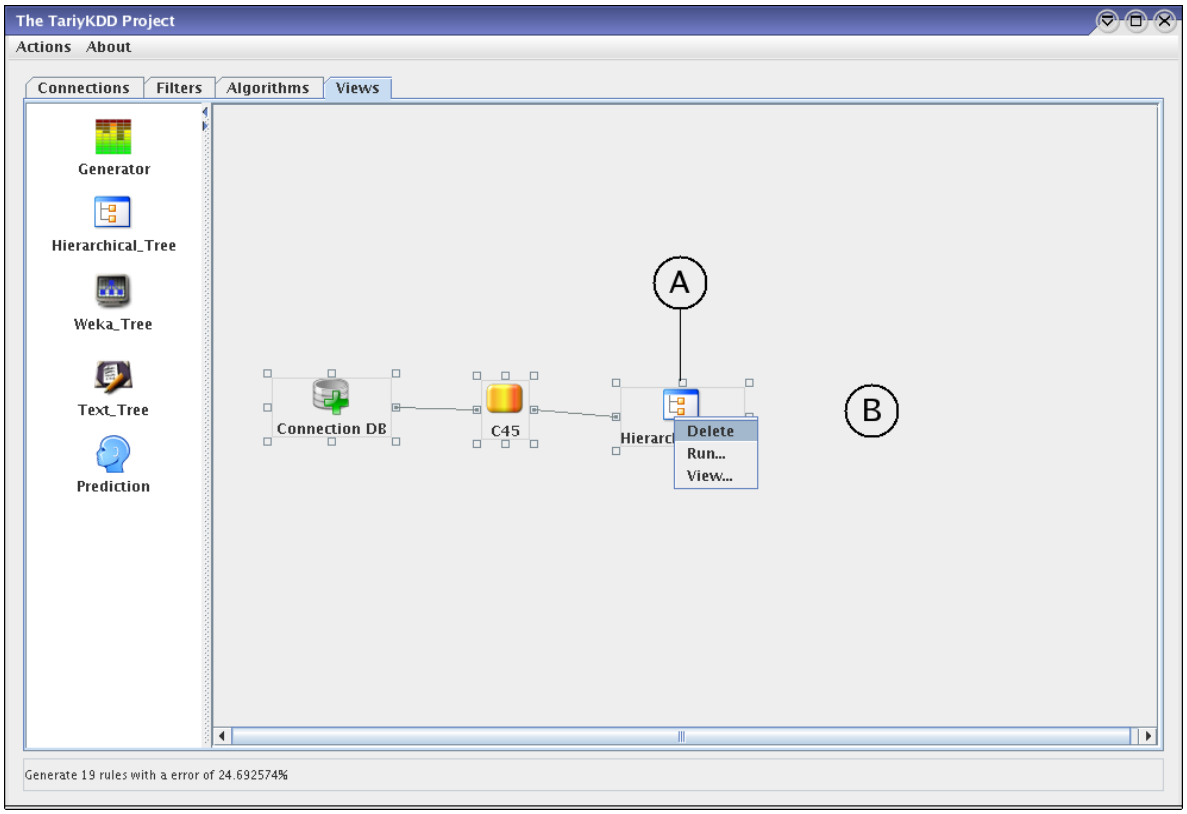
Figura C.10: Herramientas de visualización para clasificación



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña “View”.	2. Se despliegan las opciones de visualización. Para clasificación se encuentran: A “Hierarchical_Tree” muestra un arbol en entorno gráfico que puede ser colapsado y expandido. Tiene una pestaña para ver la lista de reglas obtenidas. B “Weka_Tree” implementación del árbol gráfico de la herramienta Weka. Tiene una pestaña para ver la lista de reglas obtenidas. C “Text_Tree” muestra el árbol en modo texto el cual puede ser copiado a un archivo plano. Además, tiene una pestaña para ver la lista de reglas obtenidas.

Eliminar el icono del árbol jerárquico del área de trabajo

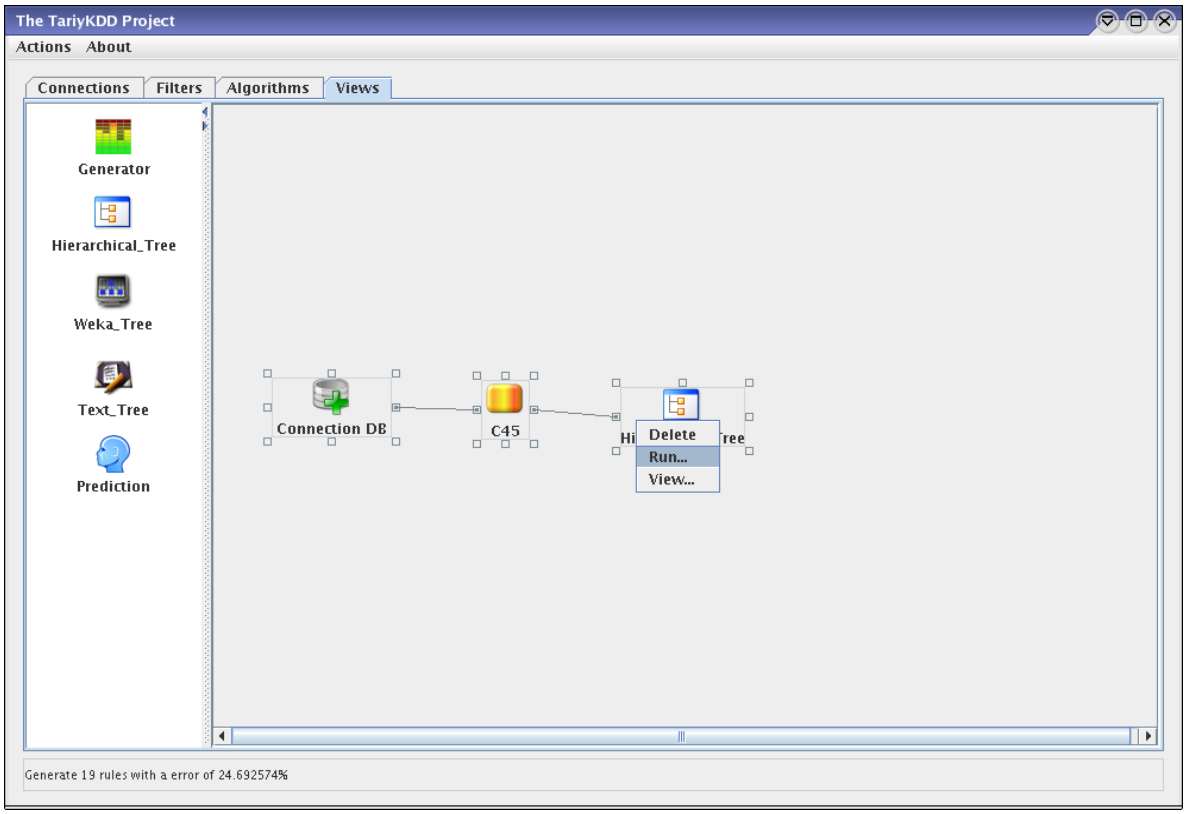
Figura C.10.1.1: Opción “Delete” del icono “Herarchical_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Herarchical_Tree” y elije la opción “Delete”.	2. El icono A “Herarchical_Tree” es borrado del área de trabajo B.

Ejecutar el icono del árbol jerárquico del área de trabajo

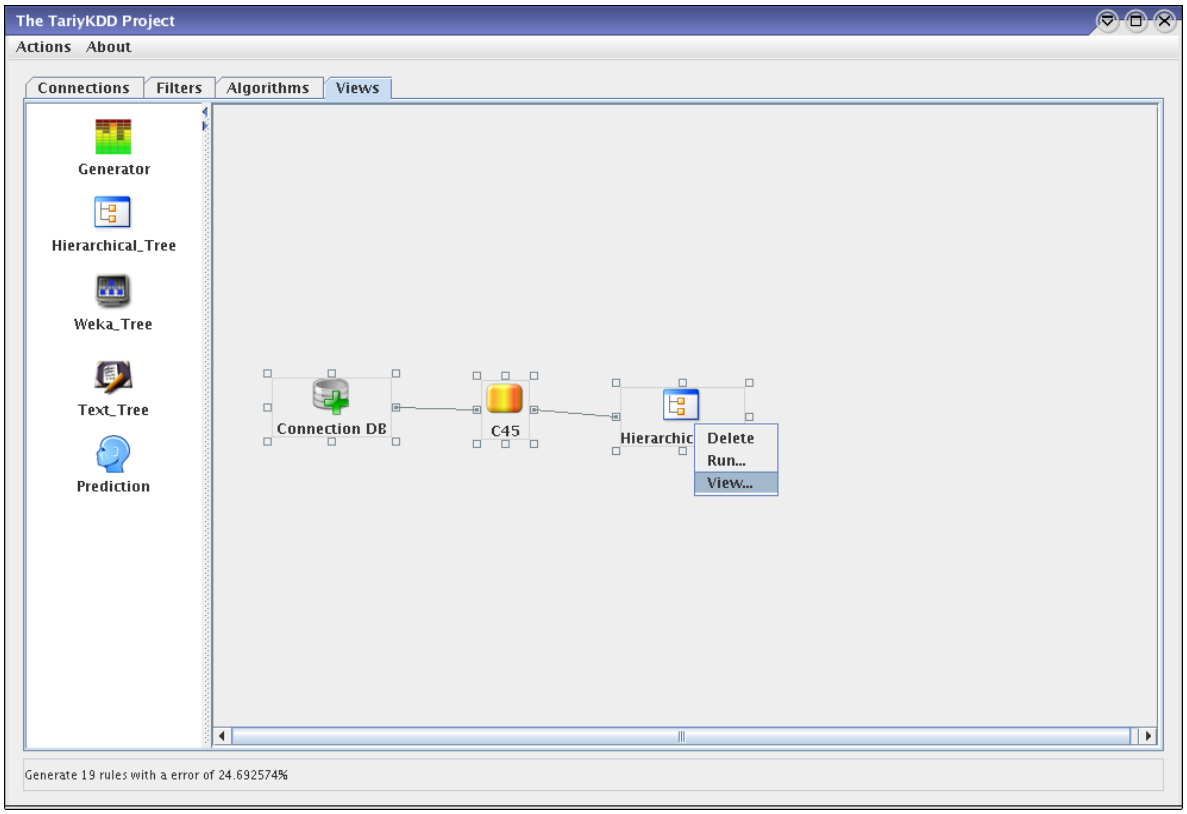
Figura C.10.1.2: Opción “Run” del icono “Herarchical_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Herarchical_Tree” y elije la opción “Run”.	2. Se ejecuta la creación del árbol.

Visualizar el árbol jerárquico

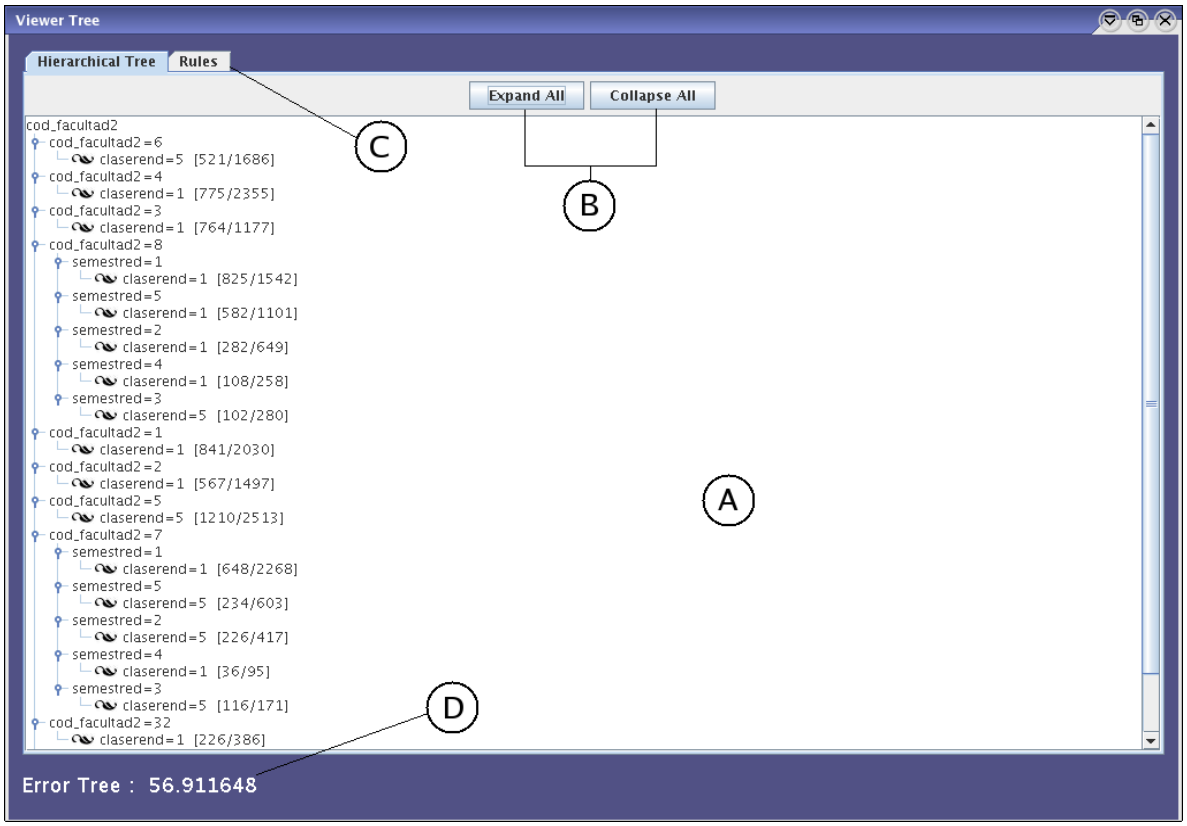
Figura C.10.1.3: Opción “View” del icono “Herarchical_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Herarchical_Tree” y elije la opción “View”.	2. Se despliega la ventana que contiene el árbol y las reglas generadas. Sus detalles se explican a continuación.

Árbol jerárquico

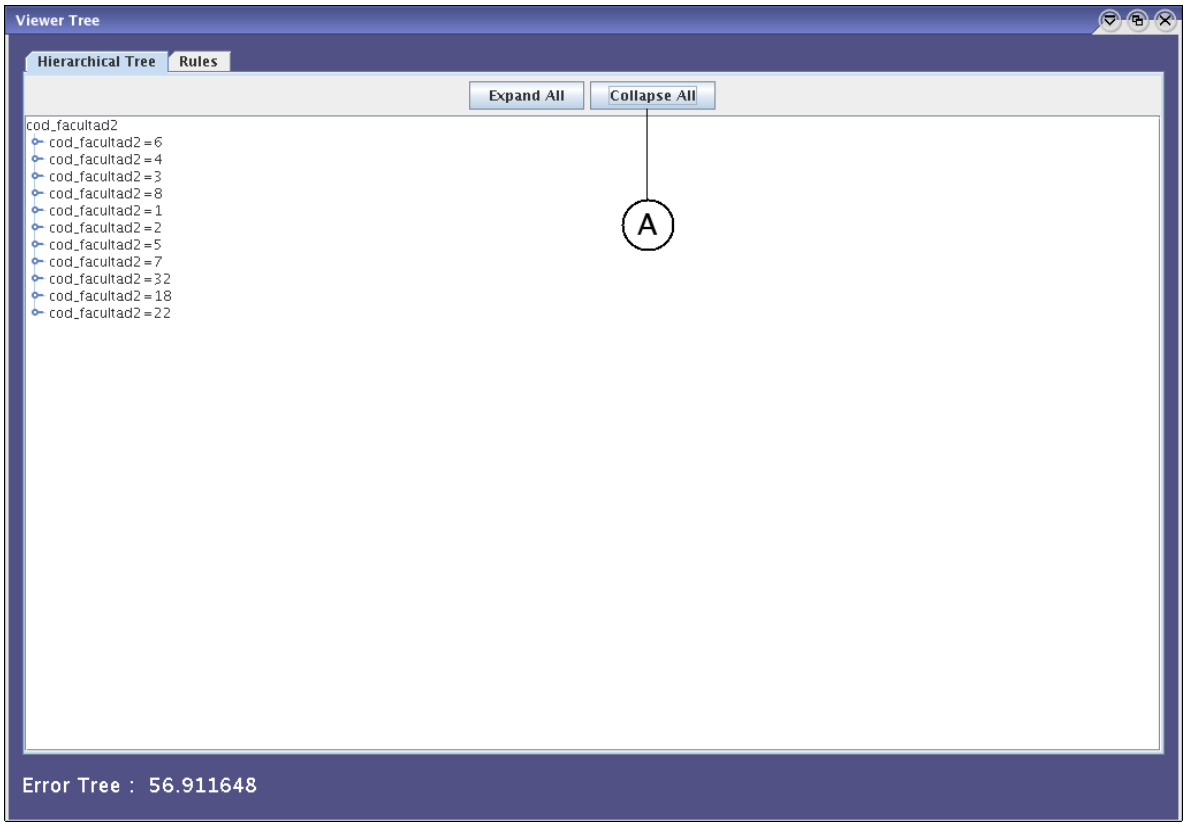
Figura C.10.1.4: Árbol jerárquico



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
	3. Se despliega el árbol jerárquico. Los campos de esta ventana son: A área del árbol, B botones para expandir o colapsar el árbol, C pestaña para visualizar las reglas generadas a partir del árbol y D porcentaje de error del árbol.

Árbol jerárquico colapsado

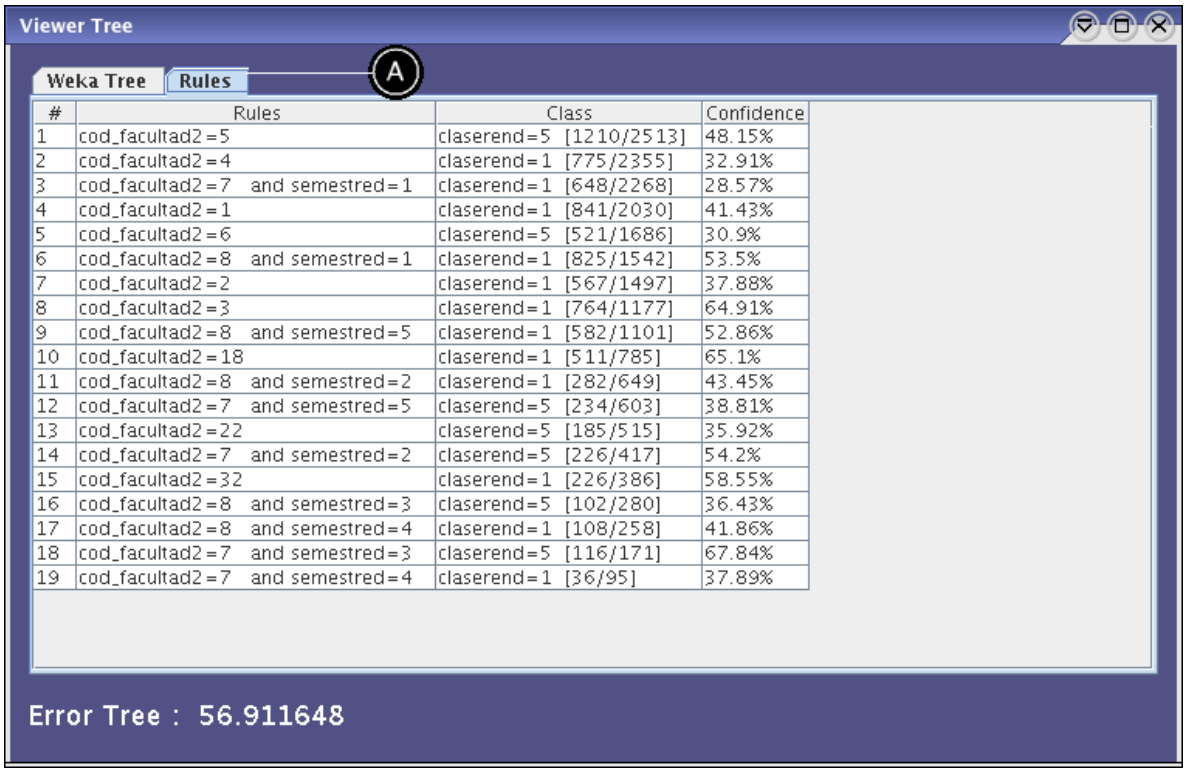
Figura C.10.1.5: Árbol jerárquico colapsado



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en el botón A “Collapse All”.	3. El árbol se contrae a su primer nivel.

Visualización de Reglas del árbol jerárquico

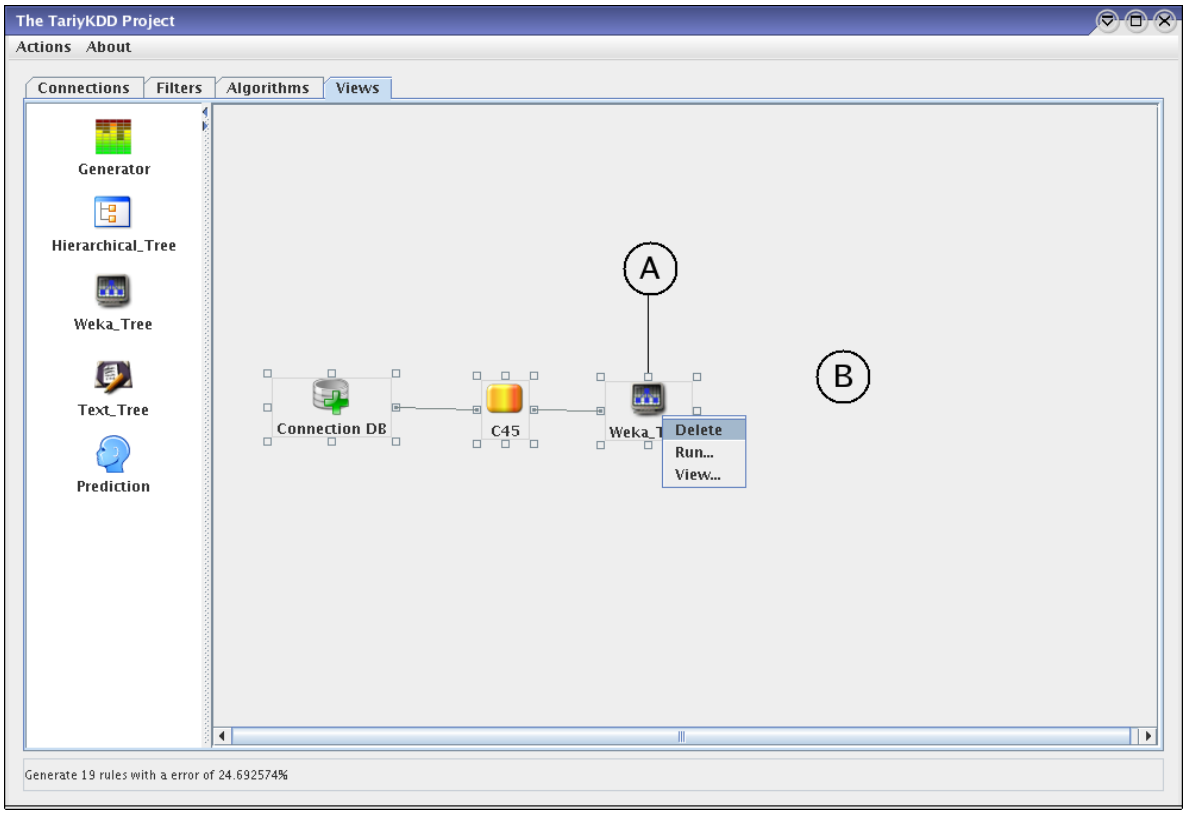
Figura C.10.1.6: Visualización de Reglas



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A “Rules”. 3. Se hace doble click sobre el nombre de una de las columnas.	2. Las reglas generadas a partir del árbol son mostradas. Esta lista puede ser ordenada de acuerdo a la columna. 4. La lista se ordena de mayor a menor o viceversa teniendo como criterio esa columna.

Eliminar el icono del árbol de Weka del área de trabajo

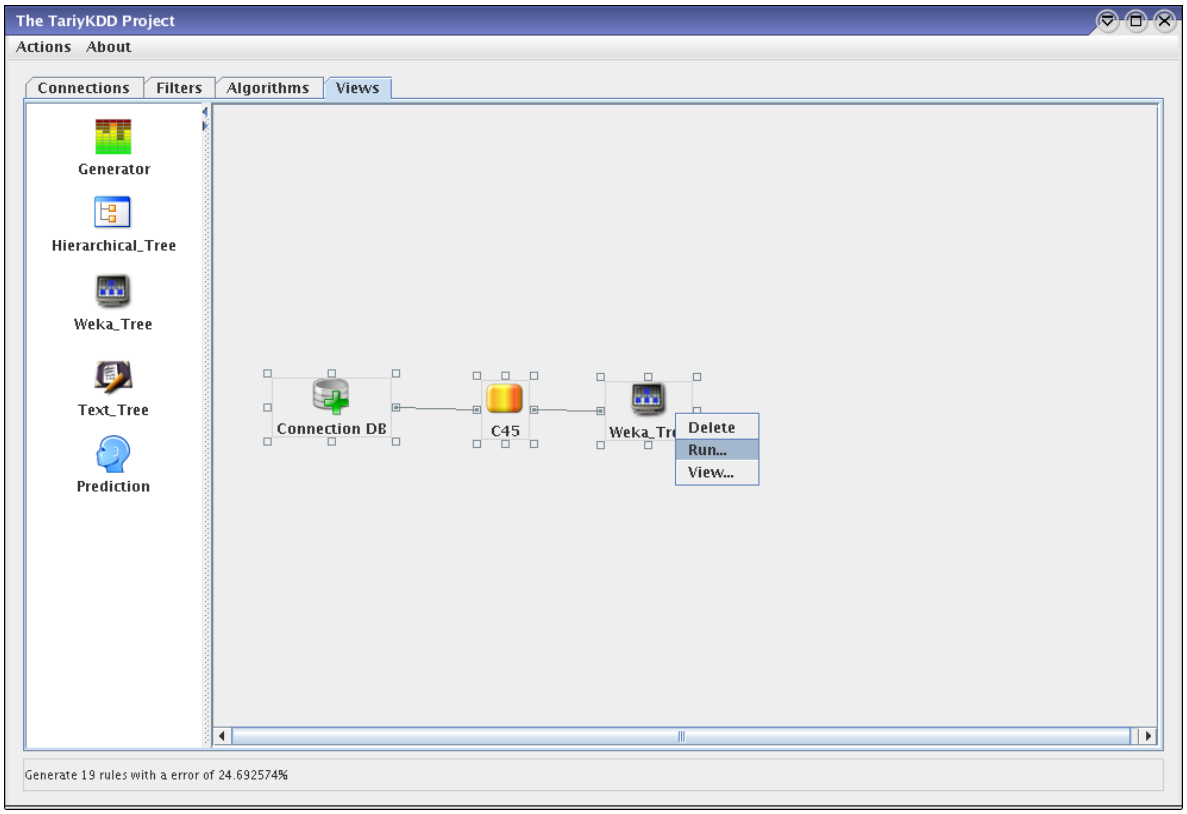
Figura C.10.2.1: Opción “Delete” del icono “Weka_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Weka_Tree” y elije la opción “Delete”	2. El icono A “Weka_Tree” es borrado del área de trabajo B.

Ejecutar el icono del árbol de Weka del área de trabajo

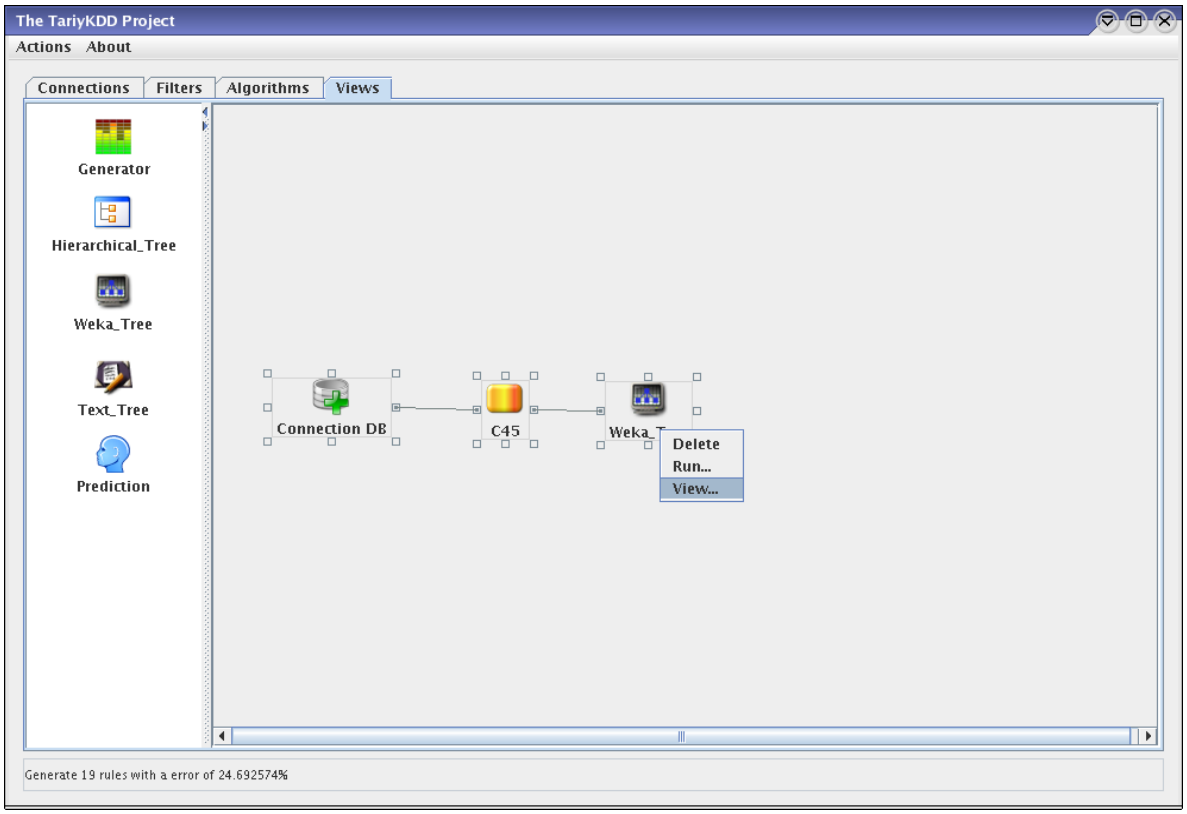
Figura C.10.2.2: Opción “Run” del icono “Weka_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Weka_Tree” y elije la opción “Run”.	2. Se ejecuta la creación del árbol.

Visualizar el árbol de Weka

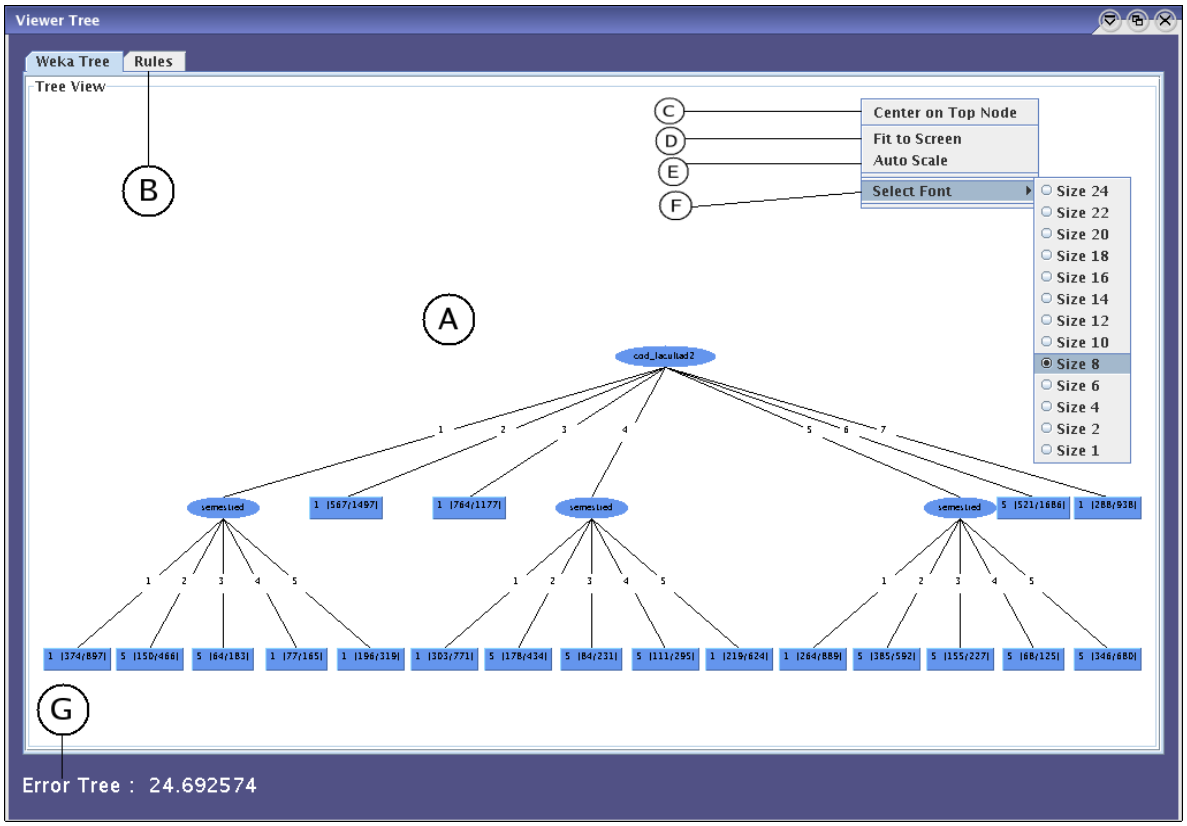
Figura C.10.2.3: Opción “View” del icono “Weka_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Weka_Tree” y elije la opción “View”.	2. Se despliega la ventana que contiene el árbol y las reglas generadas. Sus detalles se explican a continuación.

Visualización del árbol Weka

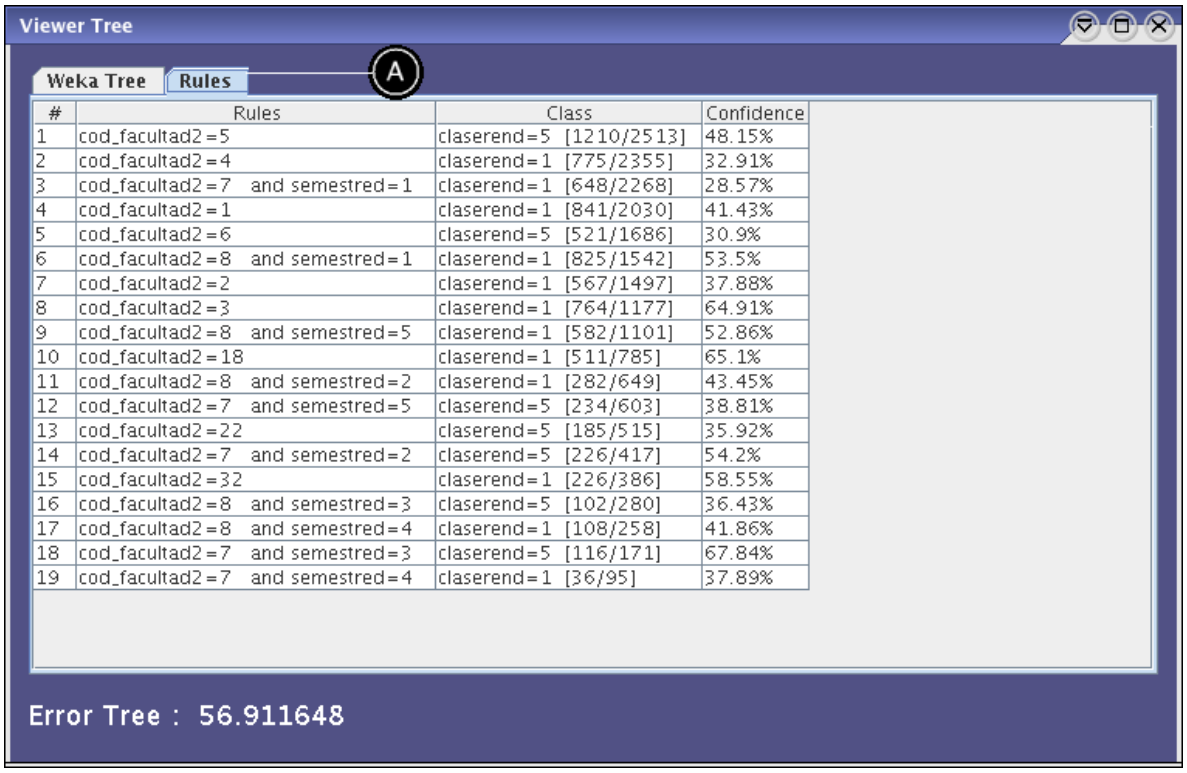
Figura C.10.2.4: Árbol Weka



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
4. El usuario hace click derecho en el área A.	3. Se muestra el árbol Weka. A área del árbol, B pestaña para ver las reglas generadas a partir del árbol y G porcentaje de error del árbol. 5. Se despliega el menú para ajustes del árbol. Las opciones son: C “Center on Top Node” el cual mueve el nodo raíz al centro de la pantalla y al árbol consigo. D “Fit to Screen” el cual ajusta el árbol al tamaño de la ventana. E “Auto Scale” el cual despliega el árbol en su totalidad sin importar que sobrepase el tamaño de la ventana y F “Select Font” que permite seleccionar el tamaño de la fuente usada.

Visualización de Reglas del árbol Weka

Figura C.10.2.5: Visualización de Reglas

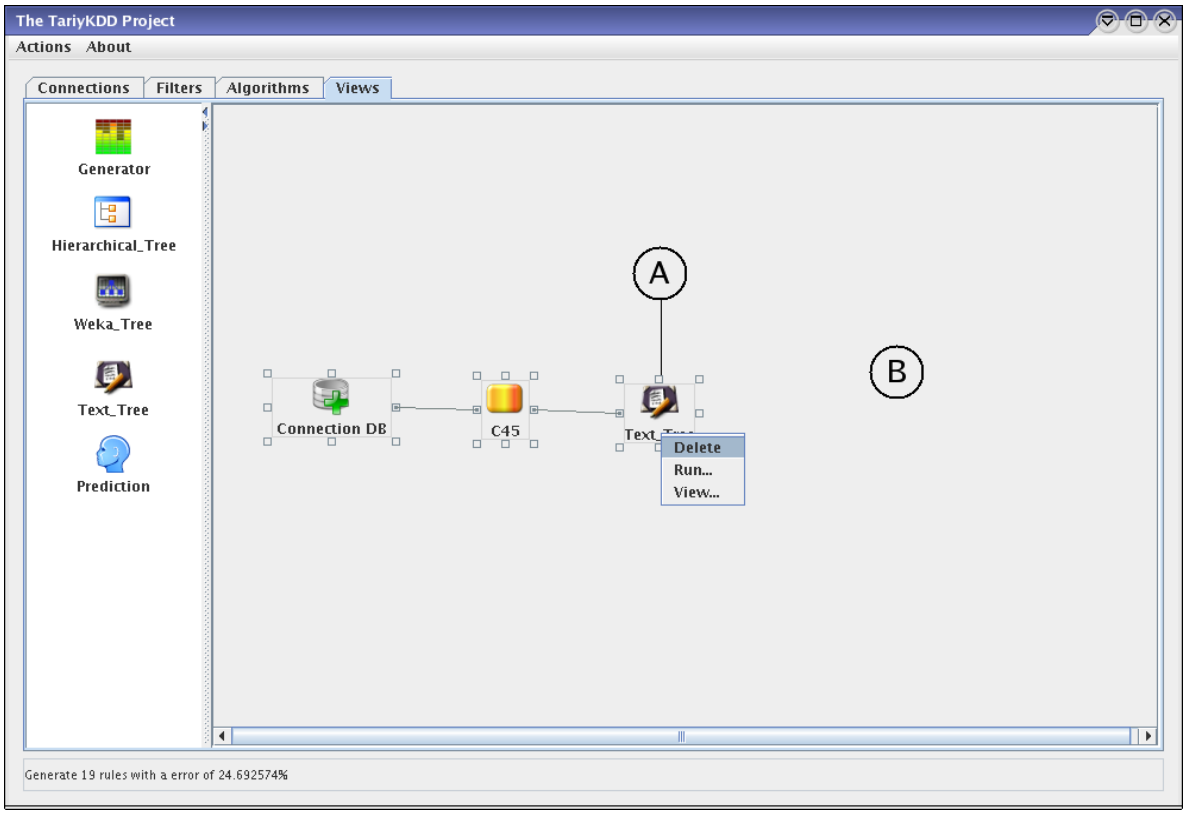


#	Rules	Class	Confidence
1	cod_facultad2=5	claserend=5 [1210/2513]	48.15%
2	cod_facultad2=4	claserend=1 [775/2355]	32.91%
3	cod_facultad2=7 and semestred=1	claserend=1 [648/2268]	28.57%
4	cod_facultad2=1	claserend=1 [841/2030]	41.43%
5	cod_facultad2=6	claserend=5 [521/1686]	30.9%
6	cod_facultad2=8 and semestred=1	claserend=1 [825/1542]	53.5%
7	cod_facultad2=2	claserend=1 [567/1497]	37.88%
8	cod_facultad2=3	claserend=1 [764/1177]	64.91%
9	cod_facultad2=8 and semestred=5	claserend=1 [582/1101]	52.86%
10	cod_facultad2=18	claserend=1 [511/785]	65.1%
11	cod_facultad2=8 and semestred=2	claserend=1 [282/649]	43.45%
12	cod_facultad2=7 and semestred=5	claserend=5 [234/603]	38.81%
13	cod_facultad2=22	claserend=5 [185/515]	35.92%
14	cod_facultad2=7 and semestred=2	claserend=5 [226/417]	54.2%
15	cod_facultad2=32	claserend=1 [226/386]	58.55%
16	cod_facultad2=8 and semestred=3	claserend=5 [102/280]	36.43%
17	cod_facultad2=8 and semestred=4	claserend=1 [108/258]	41.86%
18	cod_facultad2=7 and semestred=3	claserend=5 [116/171]	67.84%
19	cod_facultad2=7 and semestred=4	claserend=1 [36/95]	37.89%

ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A “Rules”. 3. Se hace doble click sobre el nombre de una de las columnas.	2. Las reglas generadas a partir del árbol son mostradas. Esta lista puede ser ordenada de acuerdo a la columna. 4. La lista se ordena de mayor a menor o viceversa teniendo como criterio esa columna.

Eliminar el icono del árbol en modo texto del área de trabajo

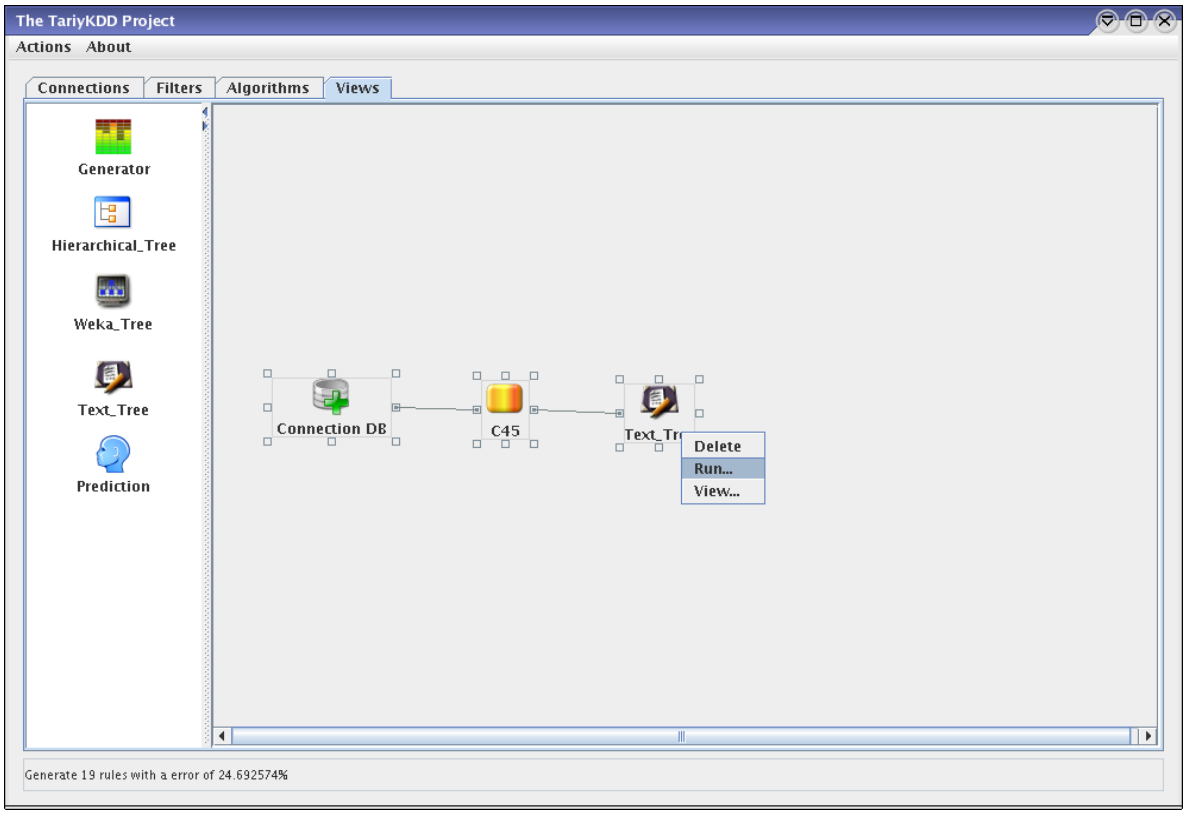
Figura C.10.3.1: Opción “Delete” del icono “Text_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Text_Tree” y elije la opción “Delete”.	2. El icono A “Text_Tree” es borrado del área de trabajo B.

Ejecutar el icono del árbol en modo texto del área de trabajo

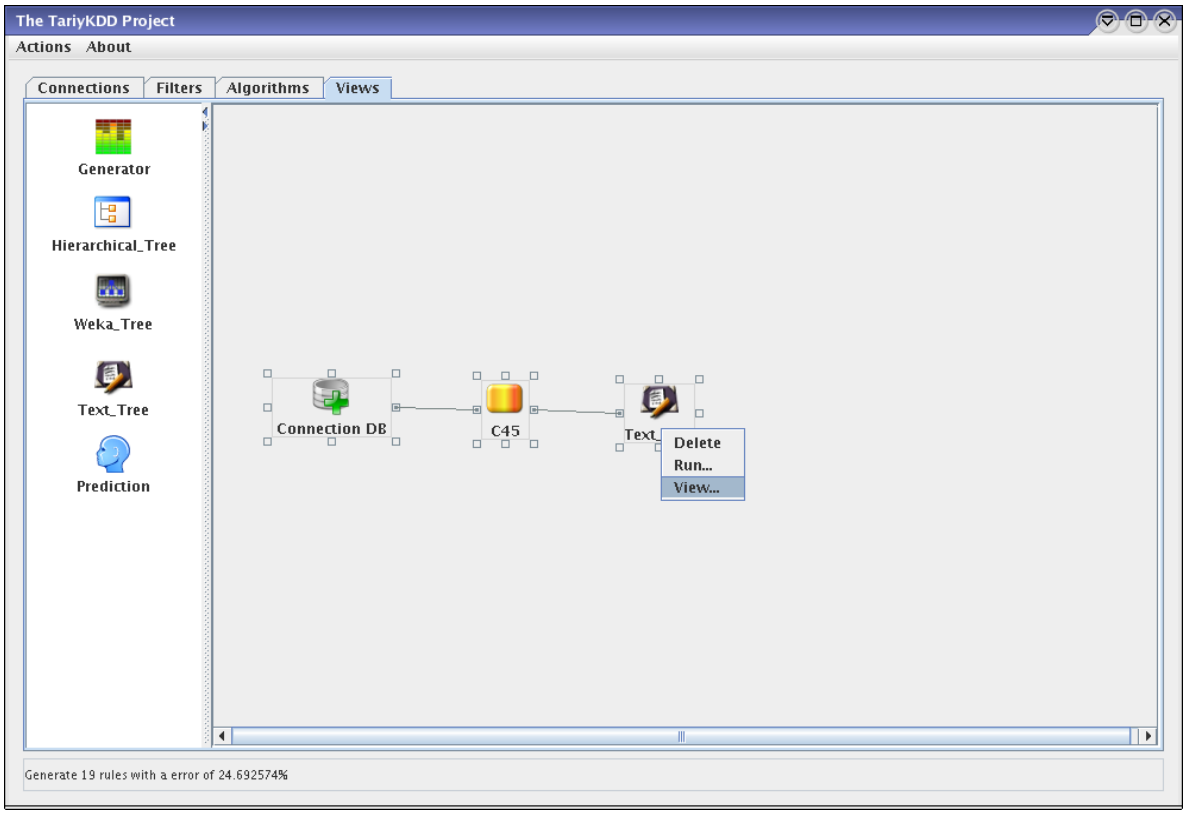
Figura C.10.3.2: Opción “Run” del icono “Text_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Text_Tree” y elije la opción “Run”.	2. Se ejecuta la creación del árbol.

Visualizar el árbol en modo texto

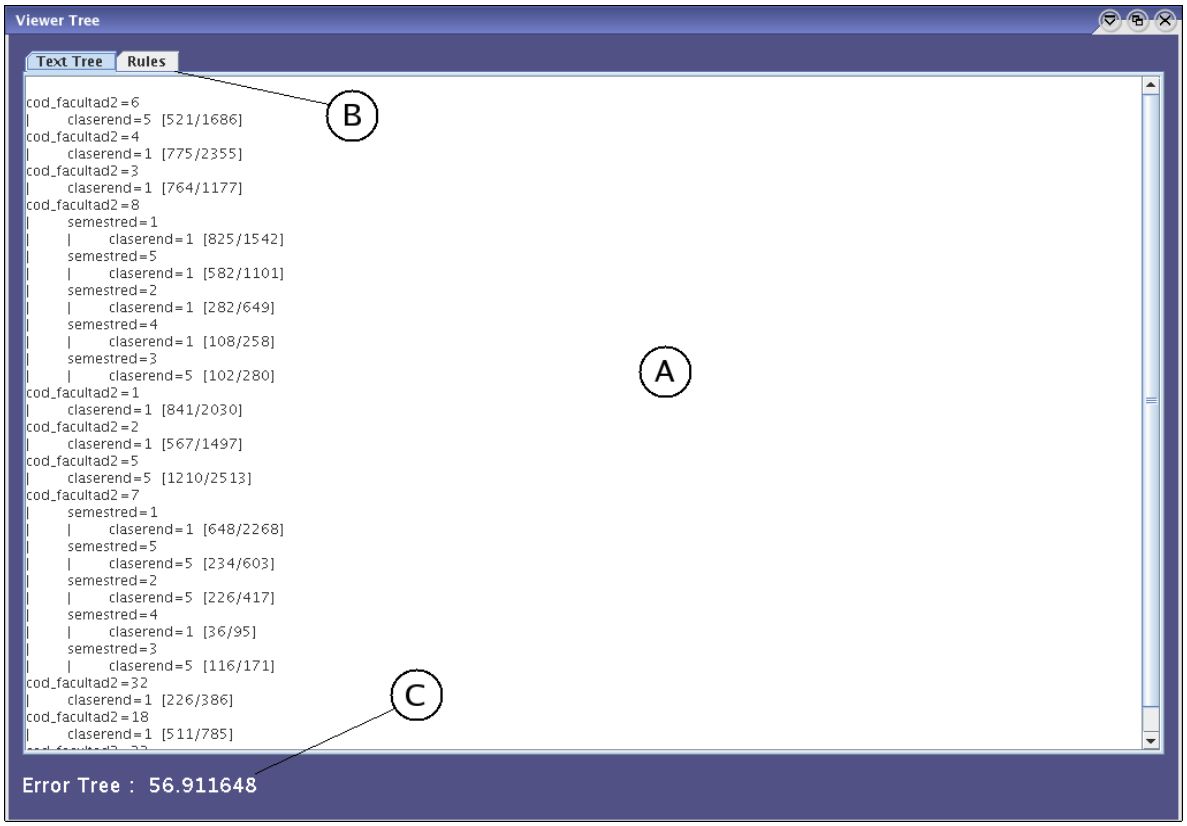
Figura C.10.3.3: Opción “View” del icono “Text_Tree”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Text_Tree” y elije la opción “View”.	2. Se despliega la ventana que contiene el árbol y las reglas generadas. Sus detalles se explican a continuación.

Visualización del árbol en modo texto

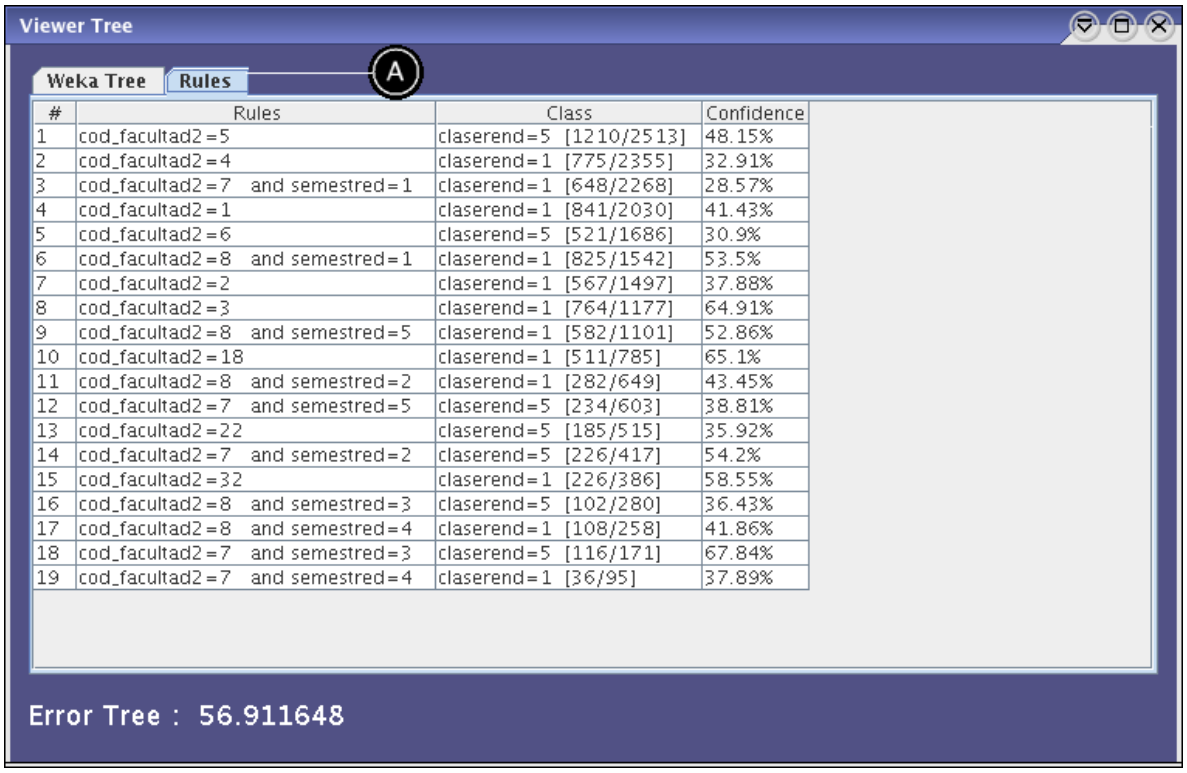
Figura C.10.3.4: Visualización del árbol en modo texto



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
	3. Se despliega la ventana que contiene el árbol en modo texto. A área del árbol, B pestaña para visualizar las reglas generadas a partir del árbol y C porcentaje de error del árbol.

Visualización de Reglas del árbol en modo texto

Figura C.10.3.5: Visualización de Reglas



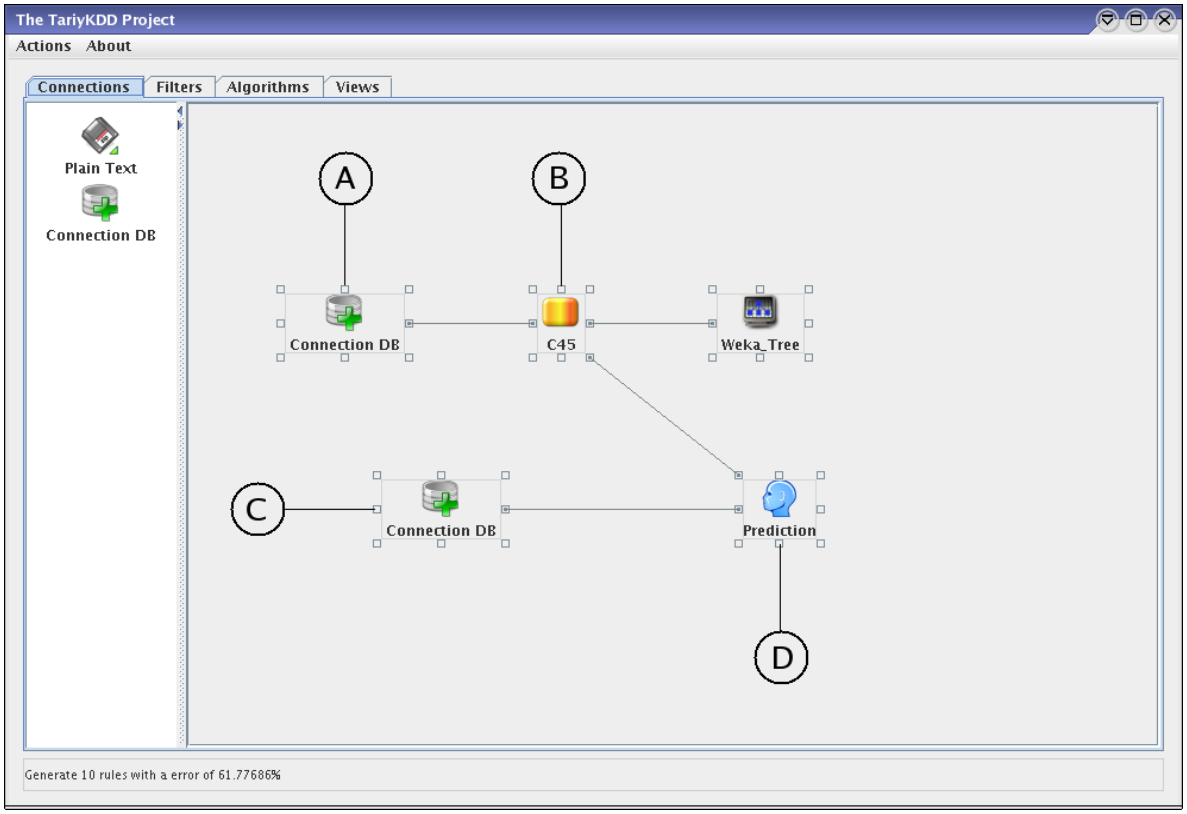
#	Rules	Class	Confidence
1	cod_facultad2=5	claserend=5 [1210/2513]	48.15%
2	cod_facultad2=4	claserend=1 [775/2355]	32.91%
3	cod_facultad2=7 and semestred=1	claserend=1 [648/2268]	28.57%
4	cod_facultad2=1	claserend=1 [841/2030]	41.43%
5	cod_facultad2=6	claserend=5 [521/1686]	30.9%
6	cod_facultad2=8 and semestred=1	claserend=1 [825/1542]	53.5%
7	cod_facultad2=2	claserend=1 [567/1497]	37.88%
8	cod_facultad2=3	claserend=1 [764/1177]	64.91%
9	cod_facultad2=8 and semestred=5	claserend=1 [582/1101]	52.86%
10	cod_facultad2=18	claserend=1 [511/785]	65.1%
11	cod_facultad2=8 and semestred=2	claserend=1 [282/649]	43.45%
12	cod_facultad2=7 and semestred=5	claserend=5 [234/603]	38.81%
13	cod_facultad2=22	claserend=5 [185/515]	35.92%
14	cod_facultad2=7 and semestred=2	claserend=5 [226/417]	54.2%
15	cod_facultad2=32	claserend=1 [226/386]	58.55%
16	cod_facultad2=8 and semestred=3	claserend=5 [102/280]	36.43%
17	cod_facultad2=8 and semestred=4	claserend=1 [108/258]	41.86%
18	cod_facultad2=7 and semestred=3	claserend=5 [116/171]	67.84%
19	cod_facultad2=7 and semestred=4	claserend=1 [36/95]	37.89%

ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A “Rules”. 3. Se hace doble click sobre el nombre de una de las columnas.	2. Las reglas generadas a partir del árbol son mostradas. Esta lista puede ser ordenada de acuerdo a la columna. 4. La lista se ordena de mayor a menor o viceversa teniendo como criterio esa columna.

Predicción en clasificación

Condiciones necesarias para aplicación del componente de predicción.

Figura C.11: Área de trabajo para predicción

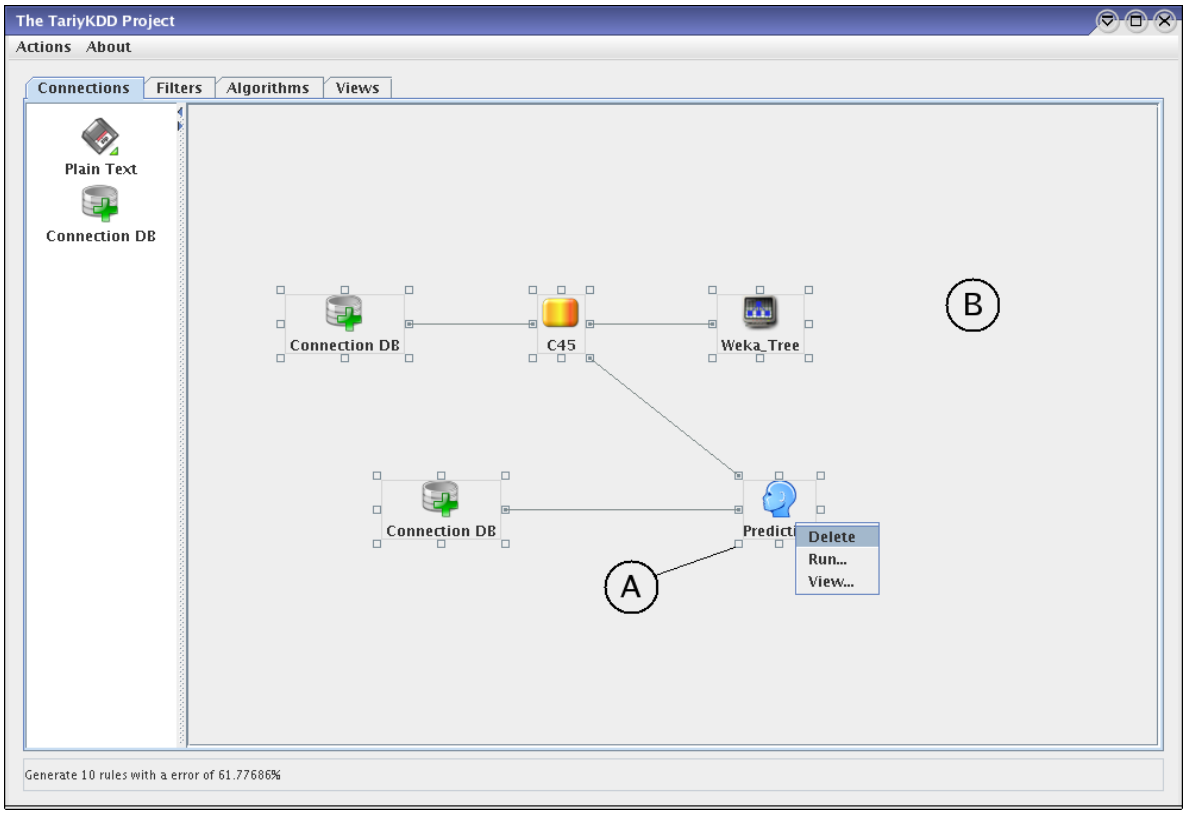


ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en el icono “Prediction”.	2. El icono D “Prediction” aparece en el área de trabajo. El uso de este componente esta condicionado a los siguientes requerimientos. Debe haberse cargado un conjunto de datos de entrenamiento A, debió haberse aplicado un algoritmo de clasificación B a conjunto A y debe haberse cargado un conjunto de datos de evaluación C que es al que se le va aplicar el modelo obtenido después de ejecutar B. Al icono D le llegan alimentación de información tanto de A como B y de esta manera es posible hacer predicción sobre el conjunto de

ACCION DEL ACTOR	*RESPUESTA DEL SISTEMA*
	evaluación. El uso de este componente dadas las condiciones citadas se explica en el próximo caso de uso real.

Eliminar el icono de predicción del área de trabajo

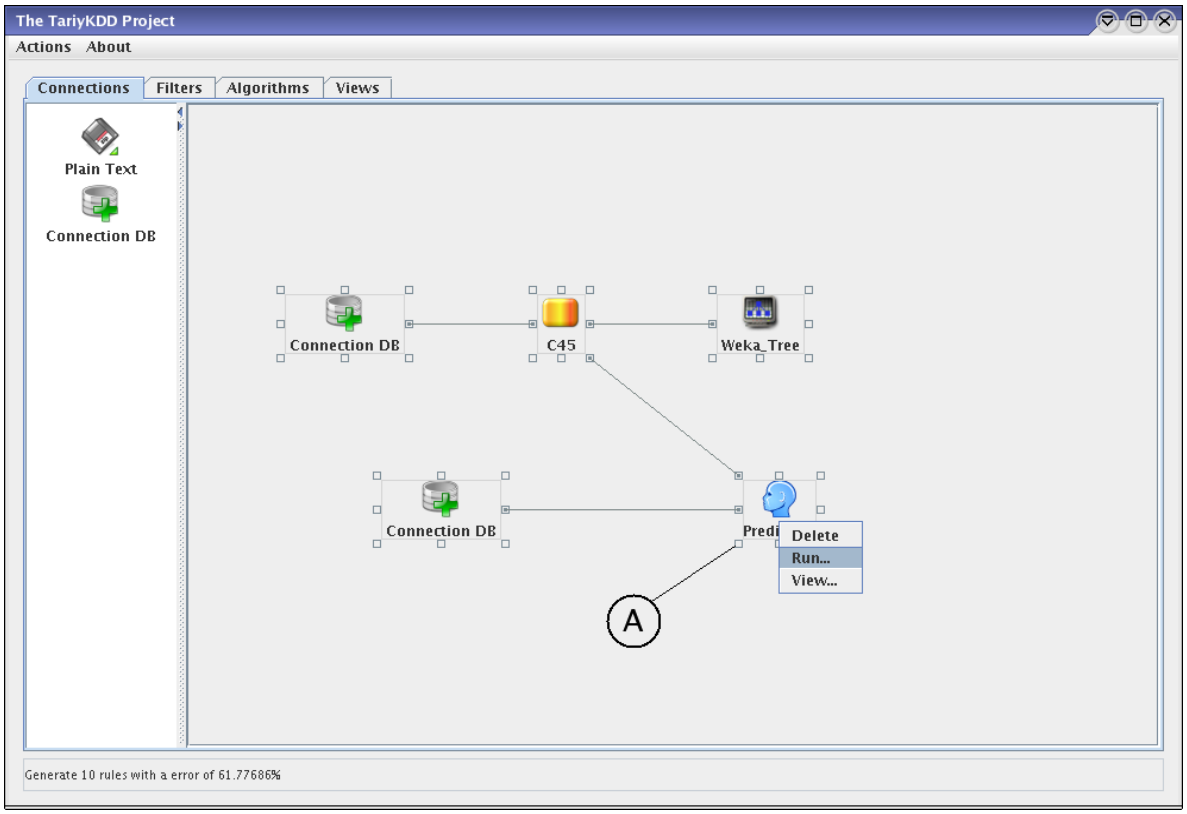
Figura C.11.1: Opción “Delete” del icono “Prediction”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono “Prediction” y elije la opción “Delete”.	2. El icono A “Prediction” es borrado del área de trabajo B.

Ejecutar la predicción sobre el conjunto de evaluación

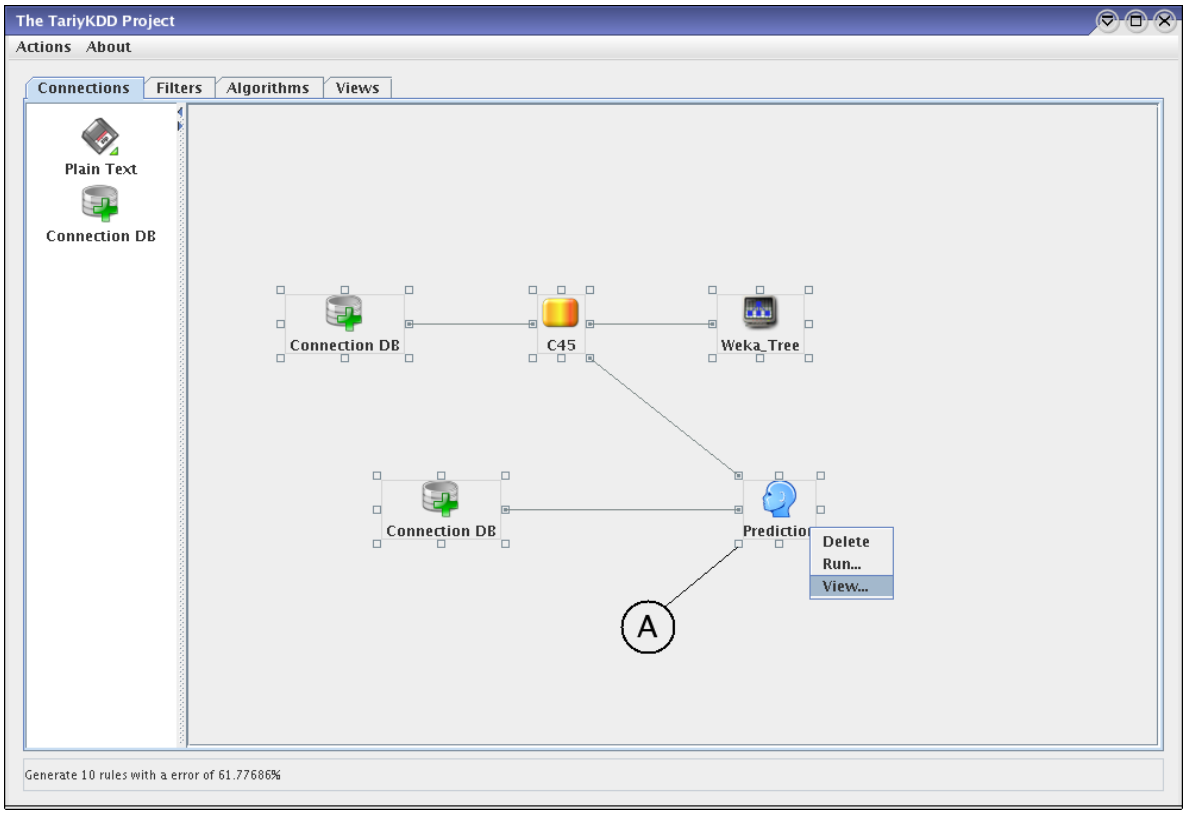
Figura C.11.2: Opción “Run” del icono “Prediction”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono A “Prediction” y elije la opción “Run”.	2. La tarea de predicción es aplicada al conjunto de evaluación.

Visualizar los resultados de la aplicación del modelo obtenido en el algoritmo de clasificación.

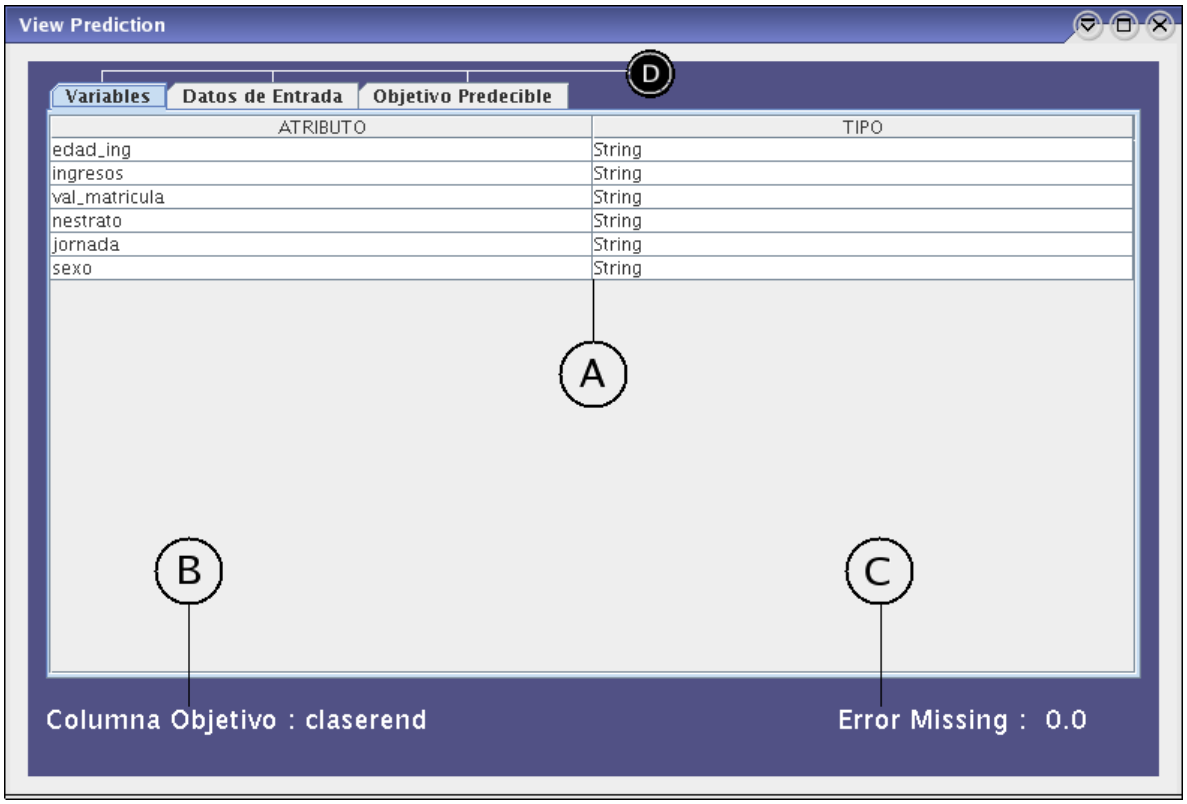
Figura C.11.3: Opción “View” del icono “Prediction”



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click derecho sobre el icono A “Prediction” y elije la opción “View”.	2. Los datos obtenidos se muestran en la ventana que se estudia en el siguiente caso de uso real.

Visualización de resultados después de aplicar predicción.

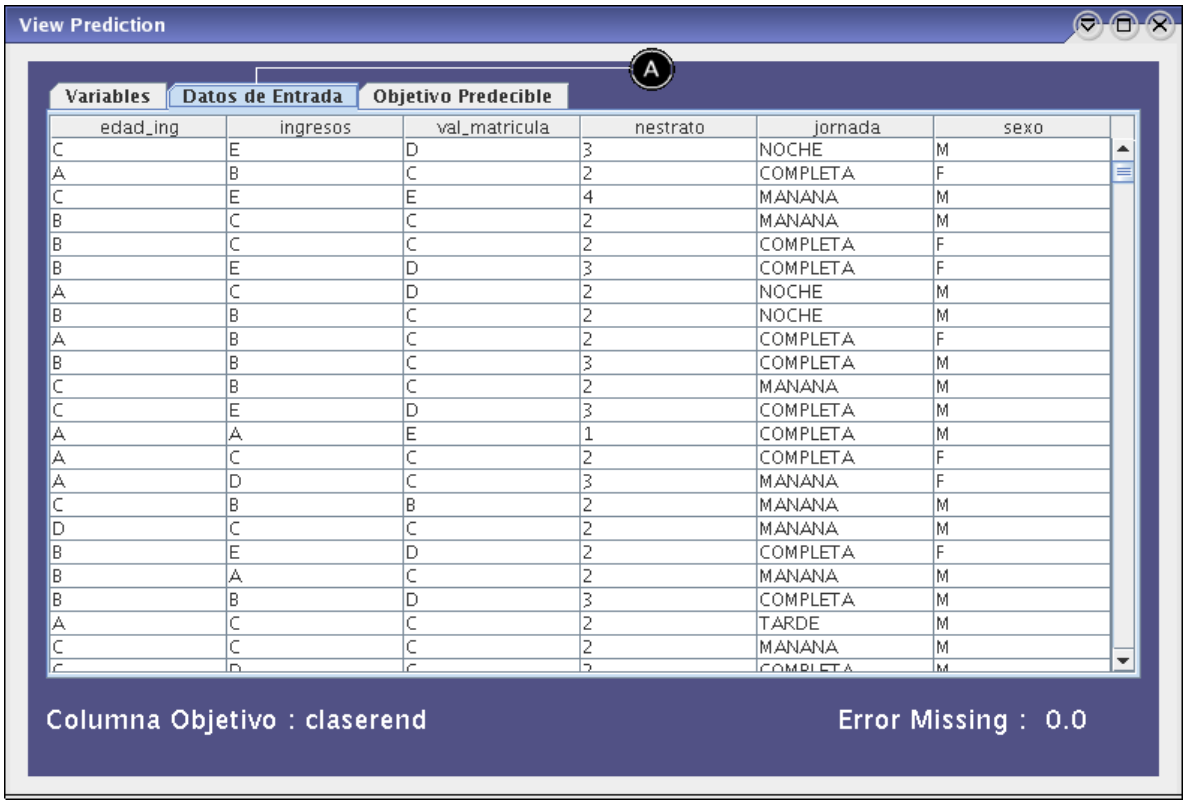
Figura C.11.4.1: Ventana de visualización de las variables o columnas utilizadas.



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
	3. Se muestra la ventana de visualización de resultados de predicción la cual comprende: A grilla de datos, B nombre de la columna objetivo, C porcentaje de error de pérdida y D pestañas para visualizar los demás datos. Esta primera pestaña muestra las variables que se evaluaron para hacer la predicción.

Visualización de resultados después de aplicar predicción.

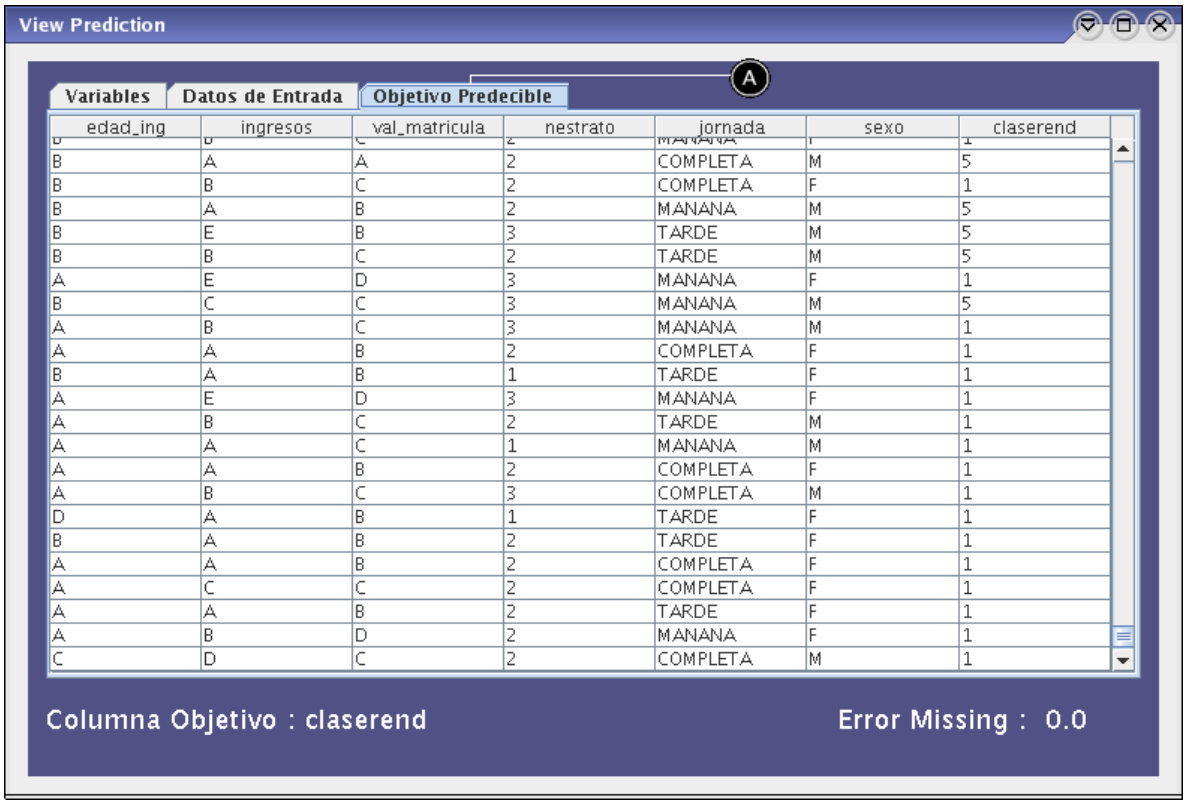
Figura C.11.4.2: Datos de entrada a predicción.



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A “Datos de entrada” o “Input Data”.	2. Se muestran los datos iniciales antes de aplicar predicción.

Visualización de resultados después de aplicar predicción.

Figura C.11.4.3: Datos a los que se ha aplicado predicción.



ACCION DEL ACTOR	***RESPUESTA DEL SISTEMA***
1. El usuario hace click en la pestaña A “Objetivo Predecible”.	2. Se muestran los datos después de aplicar predicción.

D. MANUAL DE INSTALACION SISTEMAS GNU/LINUX

D.1 Instalación de PostgreSQL

Se obtiene login como *root* con el comando *su -l*:

```
[shell]$ su -l
Password:
[shell]#
```

Se añade el grupo *postgres* y se crea el usuario *postgres* dentro del grupo *postgres*:

```
[shell]# groupadd postgres
[shell]# useradd -g postgres postgres
```

Las fuentes para compilación se encuentran en el directorio *Instalacion/Software* dentro del CD de instalación.

Se descomprime las fuentes:

```
[shell]# gunzip postgresql-8.1.16.tar.gz
[shell]# tar -xvf postgresql-8.1.16.tar
```

Se preparan las fuentes para compilarlas:

```
[shell]# cd postgresql-8.1.16
[shell]# ./configure
```

Ahora se compilan las fuentes de PostgreSQL:

```
[shell]# gmake && gmake install && echo "Bien compilado e instalado"
```

- Configuración Post-Instalación

Se crea un directorio *data* y se asignan los directorios a sus propietarios:

```
[shell]# mkdir /usr/local/pgsql/data
[shell]# chown postgres /usr/local/pgsql/data
```

Ahora se cambia a usuario *postgres* para instalar la BD:

```
[shell]# su - postgres
```

```
[shell]$ /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

Se inicia la BD:

```
[shell]$ /usr/local/pgsql/bin/postmaster -i -S -D /usr/local/pgsql/data
```

Para arrancar el servicio de *postgres* en cada inicio haga lo siguiente:

Coloque el archivo:
…postgresql-8.1.6/contrib/start-scripts/linux

*e*n */etc/init.d/postgresql* (or */etc/rc.d/init.d/postgresql*)
y digite:

```
[shell]# chkconfig --add postgresql on
```

como usuario *root*.

D.2 Instalación de la Máquina Virtual de Java

Entre al sistema como usuario *root*.

Copie el archivo *jre-1_5_0_10-linux-i586.bin* de la carpeta *Instalacion/Software* a la carpeta */opt* de su sistema

Asegúrese que el archivo tiene permisos de ejecución

```
[shell]$ su -l
Password:
[shell]#
```

Deberia mirar lo siguiente:

```
[shell]# ls -la jre-1_5_0_10-linux-i586.bin
-rwxr-xr-x 1 and and 17075579 nov 29 21:00 jre-1_5_0_10-linux-i586.bin
```

Si no es así, digite:

```
[shell]# chmod +x jre-1_5_0_10-linux-i586.bin
```

y ahora si a instalar:

```
[shell]# ./jre-1_5_0_10-linux-i586.bin
```

D.3 Ejecución de TariyKDD

Copie la carpeta *Instalacion/TariyKDD* del CD a la carpeta */opt* de su sistema

Cree un enlace simbólico al script ejecutable de TariyKDD

```
[shell]# ln -s /opt/TariyKDD/TariyKDD /usr/bin/TariyKDD
```

Para ejecutar TariyKDD digite desde cualquier ubicación:

```
[shell]$ TariyKDD
```