UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

BÚSQUEDA Y VISUALIZACIÓN DEL REGISTRO GENEALÓGICO DEL CABALLO CHILENO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

DANIEL HERNÁN BOBADILLA LEAL

PROFESOR GUÍA:
CARLOS HURTADO LARRAÍN

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIERREZ GALLARDO
JORGE OLIVOS ARAVENA

SANTIAGO DE CHILE
OCTUBRE 2008

“BÚSQUEDA Y VISUALIZACIÓN DEL REGISTRO GENEALÓGICO DEL CABALLO CHILENO”

El problema de visualización de redes y/o grafos es un área muy estudiada en ciencias de la computación y también ha sido objeto de gran atención de investigadores en los últimos años. El presente trabajo está motivado a construir una aplicación que ayude en la visualización de grafos obtenidos de consultas al registro genealógico del Caballo Chileno.

En la primera parte de este trabajo se realiza un estudio de algoritmos y técnicas para la visualización de grafos en dos dimensiones, revisando distintos autores y clasificación de éstos debido a su utilización en cierto tipo de grafo y también de restricciones introducidas a la visualización para considerarse válida y admisible. Luego se procede a plantear una solución a los requerimientos del problema: se decribe una arquitectura de la aplicación a construir, decisiones de diseo involucradas debido a restricciones del problema mismo y una descripción de la implementación. Más tarde se revisan los algoritmos construidos para generar consultas y recuperación de información y se detallan los algoritmos elegidos para la visualización de grafos generados de las consultas al registro genealógico equino.

El resultado de utilizar las tecnologías propuestas hace que el requerimiento primario de la aplicación se cumpla. No obstante por el diseo de la misma, ésta puede extenderse para nuevas interfaces que se requieran. El desempeo de la aplicación se verificó utilizando pruebas sintéticas que fueron satisfactorias. Debido a la naturaleza de la información genealógica equina y a los tipos de consultas realizadas, éstas no estuvieron exentas del problema que representa la visualización de grandes grafos; problema observado y estudiado. Para lo último se plantearon aproximaciones; que agreguen valor a la aplicación diseada para el usuario.

AGRADECIMIENTOS

A mi padre.

Agradezco a mi familia por todo.

Agradezco a mis compañeros: Álvaro, Manuel, Mauricio, Óscar, Raúl.

Mención honrosa para el resto.

A.M.D.G

Índice general

Capítulo 1

Introducción

La visualización de información ha ido tomando importancia en los últimos años y que se ha consolidado como una disciplina dentro de las ciencias de la computación. Si bien el trabajo en esta área es extenso aún queda mucho por desarrollar e investigar, es un campo abierto e interrelacionado con otras ramas de investigación en las ciencias de la computación.

Uno de los de los estudios clásicos dentro de la visualización de información es la representación gráfica de redes y/o grafos por medio del uso de computador, desarrollando así diferentes técnicas y aproximaciones al problema. Las soluciones en la mayoría de los casos dependen naturaleza de la información manipulada.

La idea de este trabajo es implementar una aplicación sobre la base de datos de genealogía e información del Caballo Chileno, que pertence a la Federación de Criadores del Caballo Chileno. Esta base de datos almacena más de cien años de historia, sobre el Caballo Chileno. Más de 180000 ejemplares aproximadamente están catastrados en ella. Se propuso implementar un sistema Web que permita a criadores y otros usuarios acceder a la información genealógica de los ejemplares inscritos en el registro de la raza.

1.1. Contexto

1.1.1. Contexto General

Uno de los desafíos y/o metas de la visualización de informaciónes dar solución al problema ubicuo en ciencias de la computación que es: *dada una estuctura modelada como un conjunto de entidades y sus relaciones* (grafo): ¿Cómo se pueden mostrar de forma entendible y fácil a una persona?

La visualización de redes, trata de resolver el problema de representar visualmente un grafo; en el captítulo siguiente se dará una definición clara de lo que se entenderá por esta estructura. El problema visualización de grafos tiene múltiples aproximaciones y también diversas soluciones, como se introducirá en el capítulo dos.

La introducción de restricciones y consultas a la información genealógica trae consigo desafíos importantes como lo son generar algoritmos e interfaces para poder responder las consultas requeridas. Con ello nace la necesidad de crear una aplicación tal que responda las consultas que un usuario, experto en el tema, tenga sobre un aspecto de la información manipulada.

1.1.2. Contexto Específico

En el contexto más específico, la presente memoria intenta y busca solucionar los siguientes problemas:

- Posibilitar la consulta sobre la base de datos de genealogía equina, por medio de una aplicación que permita usar más y de mejor forma la información ahí contenida.
- Permitir al usuario experto por medio de la aplicación un enriquecimiento de su comprensión de la genealogía equina chilena.
- Crear y diseñar una aplicación que contenga la información genealógica y que permita ser consultada para obtener información de interés. Y que su vez esta misma pueda nutrir distintas interfaces de usuario independientemente si así se quiera en un futuro.

1.2. Motivación

La motivación de esta memoria surge de la necesidad de La Federación de Criadores de Caballos Chilenos (FCCCH), para contar con una aplicación computacional para responder consultas y/o exploración sobre la base de datos del registro genealógico del Caballo Chileno.

Esta base de datos se encuentra publicada actualmente en el sitio web de la Federación *http://www.caballoyrodeo.cl/*, que se se realiza bajo su supervisión de la misma para información de sus usuarios y público en general, la cual cuenta con algunas funcionalidades que en este trabajo se extendieron y se agregaron otras también.

El presente trabajo consistió en el diseño y construcción una herramienta que permita visualizar información de la genealogía que básicamente se estructura como un gran grafo. Este software debe proveer visualización de datos genealógicos, que por medio de consultas, pueda mostrar características de interés sobre equinos de un cierto árbol familiar.

El presente trabajo es pionero en incorporar tecnología para construir una herramienta computacional de fácil uso, para la exploración de la genealogía equina chilena.

1.3. Objetivos Generales

1.3.1. Requerimientos del Problema

Diseñar e implementar una aplicación interactiva que permita manipular información de la genealogía de la Federación de Criadores del Caballo Chileno, capaz de responder consultas de interés tanto como para el usuario experto como para el público en general.

1.3.2. Objetivos específicos

Como objetivos específicos, se tiene:

- Construir una aplicación, servicio, que soporte el almacenamiento y consulta del registro genealógico del Caballo Chileno,
- Implementar una interfaz que permita al usuario ingresar los parámetros de búsqueda y consulta al registro genealógico del Caballo Chileno,
- Recomendar al usuario formas adecuadas de visualización de los grafos generados por sus consultas al registro genealógico del Caballo Chileno.

1.4. Contenido de la memoria

El capítulo de Antecedentes, se entrega un resumen general de las ideas y/o paradigmas de visualización de grafos, así como también de las principales restricciones usadas en los métodos de visualización de grafos. Este comienza definiendo lo que se entederá por un grafo y profundizando en los métodos de visualización de grafos en dos dimensiones (2D).

En el capítulo siguiente: Metodología, se analiza en detalle la arquitectura de la aplicación construida, sus capacidades y limitaciones. Se mencionan algunas consideraciones de diseño con sus ventajas y desventajas de la elección de cierto enfoque respecto a otro.

Los resultados de la visualización, de acuerdo restricciones impuestas, se entregan el capítulo cuatro, discutiendo y señalando los resultados y desempeño de los módulos que componen el desarrollo hecho.

En el capítulo final, se entregan las conclusiones contrastando con la bibliografía disponible, así como también de los resultados obtenidos por el desarrollo.

Capítulo 2

Antecedentes

En el presente capítulo, se entrega una visión general de los métodos de cálculo de visualización de grafos, desde una perspectiva de de visualización de la información, sin profundizar características propias de los algoritmos de trazado.

El estudio de métodos de visualización de grafos es de gran utilidad práctica y teórica, usándose en muchas áreas tales como: genética, ingeniería, computación (representación de la web como grafos, diagramas de clases, networking, etc.), especialmente donde los datos constituyen grandes redes. Una de las tareas considera la representación ágil de información, para ello la atención se debe centrar en estudiar técnicas de visualización de información [1], trazado de grafos en este caso particular.

El problema principal de la trazado de un grafo pueder ser enunciado de la siguente forma: *dado un conjunto de nodo y arcos (relaciones), calcular la posición de los nodos y dibujar la curva que representa los arcos.* La bibliografía especializada [2] tiene una buena descripción de cuan bueno es un algoritmo para calcular el trazado de un grafo, dada su naturaleza las restricciones que se impongan a la visualización requerida.

2.1. Visualización de grafos

2.1.1. Grafo

Se entenderá por grafo, a un conjunto de entidades y sus relaciones. A las entidades se les llamará, nodos del grafo y a las relaciones arcos. Cada vértice unirá sólo dos nodos. Generalemente un gráfo se denota por un conjunto de arcos E y un conjunto de nodos V, es decir, un grafo se denotaría como $G(E, V)$.

Dependiendo del tipo sea el grafo: si es dirigido, acíclico, árbol o planar; los algoritmos de trazado de la visualización de un grafo se pueden categorizar de las siguientes formas y características:

- Si se desea mostrar propiedades específicas de un grafo hay que introducir restricciones en el trazado de su visualización. Ello trae consigo costos en la complejidad de cálculo para los algoritmos de trazado.
- Muchos de los algoritmos de para trazar un grafo trabajan sólo (o bien) si el grafo pertenece a una clase específica (árbol, planar, etc) .Por ello se tiene que la *clase de un grafo* es un de los parámetros esenciales para calcular su visualización.

Para especificar bien lo anterior se tienen los siguientes requerimientos para el cálculo de la visualización de un grafo:

1. Convención de dibujo.
2. Estética.
3. Restricciones.

2.1.2. Convención de dibujo

Una convención de dibujo es una regla que tiene que cumplir el dibujo del grafo para que este sea admisible y se refiere a la representación de los arcos. Por ejemplo, en un árbol, el arco es una recta de nodo a nodo, indicando sentido de la relación. Una convención puede ser muy compleja y puede acarrear muchos detalles al cálculo del trazado del grafo. Una lista de la convenciones de dibujo más usuadas son las siguiente:

Dibujo polilínea : El arco es dibujado por una polilínea conformada por segmentos de recta.

Dibujo en línea recta : El arco es representado como recta.

Dibujo ortogonal : El arco es presentado como una conjunto cadena de rayos ortogonalmente alternados.

Dibujo en cuadrícula : Nodos, cruces y extremos de los arcos tienen coordenadas enteras.

Dibujo Planar : No hay cruce de arcos.

2.1.3. Estética

La estética se refiere a las propiedades gráficas que el dibujo de un grafo pueda tener, para obtener la legibilidad en el trazado de la visualización. Las estéticas comunmente adoptadas [3] son las siguientes:

Cruces : Minimización del número total de cruces entre arcos. Generalmente se opta por tener una dibujo planar del grafo, pero no todos los grafos son planares.

Área : Minimización del área requerida por la visualización (trazado). La capacidad para tener el despliegue eficiente en área es escencial para aplicaciones de visualización.

El área de dibujo puede ser definida como el algoritmo lo requiera. Por ejemplo el área puede ser el área del polígono convexo que cubre el dibujo *convex-hull*, o el área del mínimo rectángulo que cubre el dibujo del grafo.

Largo total de los arcos : Minimización de la suma de los largos de los arcos. Esta estética sólo tiene sentido si las convenciones de dibujo permiten que esta pueda ser escalada arbitrariamente.

Arco de máximo largo : Minimización del largo máximo de un arco. Esta estética sólo tiene sentido si las convenciones de dibujo permiten que esta pueda ser escalada arbitrariamente.

Arco de largo uniforme : Minimización de la varianza de los tamaños de los arcos.

Máximo de segmentos en polilínea : Minimización del número total de de segmentos en polilínea en que representan un arco.

Uniformidad de segmentos en polilíneas : Minimización de la varianza del número total en los segmentos que conforman las polilíneas que representan los arcos.

Resolución angular : Maximización del ángulo mínimo entre dos arcos incidentes en un mismo nodo. Ocupada para los arcos dispuestos como líneas rectas.

Simetría : Mostrar las simetrías del grafo en el dibujo. Esta estética se puede formalizar introduciendo modelos matemáticos como los citados en [4].

Las estéticas están relacionadas con problemas de naturaleza combinatorial y problemas de optimización, ello repercute que en los cálculos del trazado de la visualización estos problemas sean computacionalmente difíciles. Debido a la alta complejidad, introducidas por las

estéticas en el cálculo del trazado de la visualización, sean han desarrollado aproximaciones y huerísticas que tienen buenos resultados en la práctica.

2.1.4. Restricciones

Las dos convenciones anteriores son reglas o criterios que rigen el trazado del grafo completo. Las *restricciones* se refieren a un subgrafo del grafo tratado o a una parte del trazado completo. Por ejemplo dado un nodo: *este debe estar en el centro del dibujo*, por ejemplo la raíz en una disposición de grafo de tipo *árbol*.

Las restricciones más adoptadas en aplicaciones directas son las siguientes ([5],[6]) :

Centrado : Disponer un nodo dado en el centro de del dibujo.

Externo : Dado un nodo disponerlo en los límites exteriores del dibujo.

Secuencia izquierda-derecha (arriba-abajo) : Dibujar un camino de nodos entre dibujarlo horizontalmente de izquierda a derecha (o verticalmente alineado desde arriba hacia abajo).

Figura : Dado un subgrafo del grafo original dibujarlo como cierta figura, una conformación geométrica definida a priori.

Todo lo enunciado anteriormente conlleva que los requerimientos de una visualización de un grafo esté modelada en términos de un conjuntos de ésteticas, conjuntos de conveciones de dibujo y de restricciones. Estos son los parámetros fundamentales de las metodologías de cálculo de visualización de grafos.

2.1.5. Eficiencia

Un parámetro muy importante en el cálculo de visualización de grafos es su eficiencia computacional como algoritmo. En aplicaciones de tratamiento y despliegue de grafos se requiere responsibidad frente al usuario, incluso para grafos de gran tamaño. De ello se desprende que la eficiencia computacional de una técnica de cálculo de visualización para el trazado de un grafo en un parámetro a tener en consideración.

2.2. Algoritmos para visualización

Al describir antes los requerimientos para las técnicas de visualizar grafos se procederá a revisar algunos de los los algoritmos clásicos para la visualización de grafos en dos dimensiones. Una buena bibliografía en el tema puede encontrarse en *Graph Drawing: Algorithms*

for the Visualization of Graphs [2]. y las distintas versiones del Anual Symposia on Graph Drawing. Áreas de investigación relacionadas con el tema son: geometría, interfaces humano-computador, optimización combinatorial, geometría computacional. Acá sólo se nombrarán aproximaciones clásicas, y no se profundizará en ellas.

Cuando se habla de algoritmos para el dibujo de grafos, éstos se pueden clasificar en dos tipos:

- Algoritmos para clases específicas de grafos.
- Algoritmos para grafos en general.

2.2.1. Algoritmos para clases específicas de grafos

Hay una variedad de algoritmos para clases específicas de grafos. Hay tres clases de grafos que resultan los más estudiados y tienen atención de los investigadores, entre estos están:

- Árboles
- Grafos dirigidos acíclicos.
- Grafos planares.

Grafo tipo árbol

Un árbol es un grafo conectado y acíclico, donde cada nodo está conectado con un único nodo. La mayoría de los algoritmos para el dibujo de árboles suponen como convención que los arcos se dibujan como líneas rectas dirigidas. En [7] se describen seis restricciones estéticas para dibujar el árbol, demoninadas *eumorphous*.

1. La altura de un nodo (distancia a la raíz), debe ser proporcional a su distancia de la raíz medida desde las ramas de éste. Es decir los vétices son dispuestos en niveles horizontales discretos.
2. Cuando los nodos hijos estan ordenados (árbol binario), el hijo izquierdo debe ser ubicado estrictamente a la izquierda de su padre. Para el hijo derecho debe ser ubicado a la derecha.
3. Los nodos en cada nivel horizontal deben ser dispuestos para minimizar la separación entre los mismos pero no deben sobreponerse.
4. Subarboles isomorfos deben trazarse de la misma manera.

5. Los nodos raíces (padres) deben estar centrados sobre sus hijos.

6. Los arcos no deben cruzarse, el dibujo debe representar la planaridad del árbol.

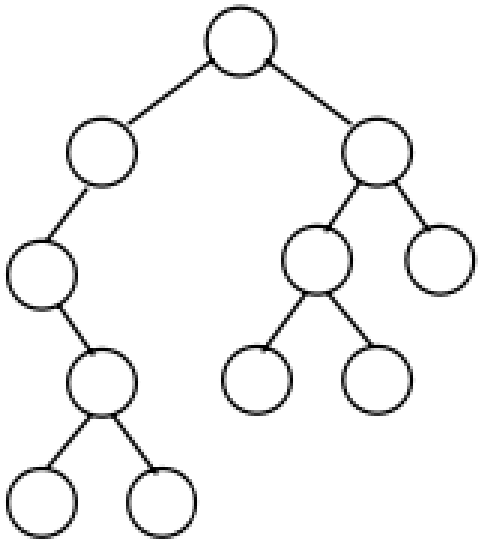


Figura 2.1: Grafo Árbol según Supowit y Reingold.

Se ha demostrado por Supowit y Reingold y otras investigaciones que se puede minimizar el tamaño del árbol debido a estas restricciones. El algoritmo de Reingold y Tilford [8], cumple todos las restricciones mencionadas anteriormente pero no entrega un árbol con tamaño óptimo. Posteriores avance demostraron que si las posiciones de los nodos en el dibujo, son coordenadas reales, entonces el problema se convierte en uno de optimización tal que puede ser solucionado en tiempo polinomial. Si se restringe las coordenadas de los nodos a coordenadas enteras, entonces el problema es NP-duro.

El problema de dibujar árboles sin una raíz definida no ha recibido demasiada atención. En [9] se describe una aproximación para despliegue de árboles *radialmente* . El algoritmo consiste en escoger una raíz téorica del grafo, como centro del árbol esto es un nodo que minimiza al altura del árbol hacia afuera desde ese nodo. Si existe más de una de esas raíces se escoge una al azar. Despues se disponen los demás los arcos como en un círculo concéntrico alrededor de esa raíz. Los arcos son dibujados como líneas rectas. El algoritmo trata de minimizar la varianza del largo de los arcos así como respeta la planaridad del grafo. El algortimo es recursivo y ocupa tiempo lineal con respecto a la cantidad de nodos.

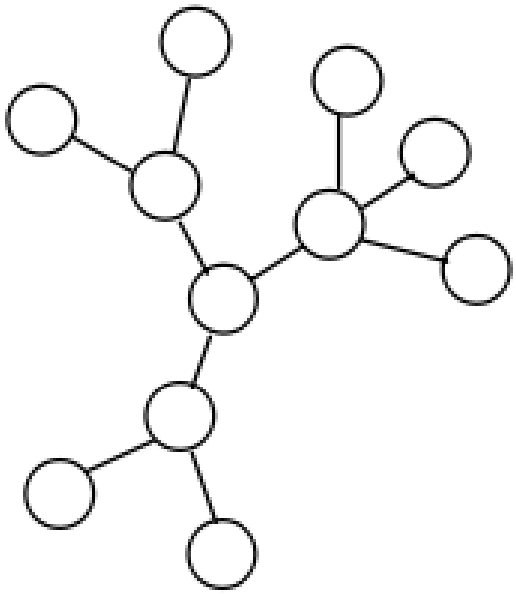


Figura 2.2: Disposición radial.

Grafos dirigidos acíclicos

Los grafos dirigidos acíclicos, como los árboles, tienen una inherente dirección de flujo. Son usados para representar estructuras jerárquicas. Las convenciones de dibujo son muy similares a la restricciones para la clase de tipo árbol pero éstos pueden ser no planares por lo que la restricción de planaridad se reemplaza para por una restricción del no cruce de arcos.

Un algoritmo clásico para trazar grafos dirigidos acíclicos fue propuesto por Sugiyama [10]. El algoritmo cuenta de tres fases ilustradas en la figura 2.3

La primera fase 2.3(b) asigna nodos a niveles tales que cada arco es dirigido *hacia arriba*, esto desde un nivel bajo a otro más alto. En esta fase se crean nodos *falsos* (dummies) tales como sean necesarios, para conectar a los nodos dentro de esos niveles y recrear el arco original. La cantidad de nodos *falsos* está dada por la cantidad de niveles que separan a los nodos no *falsos*, esto por cada arco. Para determinar el número de niveles se usa diferentes aproximaciones la original consiste en un *layering* del camino más largo, esto es el nivel de un nodo es el número de arcos que inciden desde el camino más largo en ese nodo.

La segunda 2.3(c) fase consiste en ordenar cada nivel horizontal con el objeto de mini-

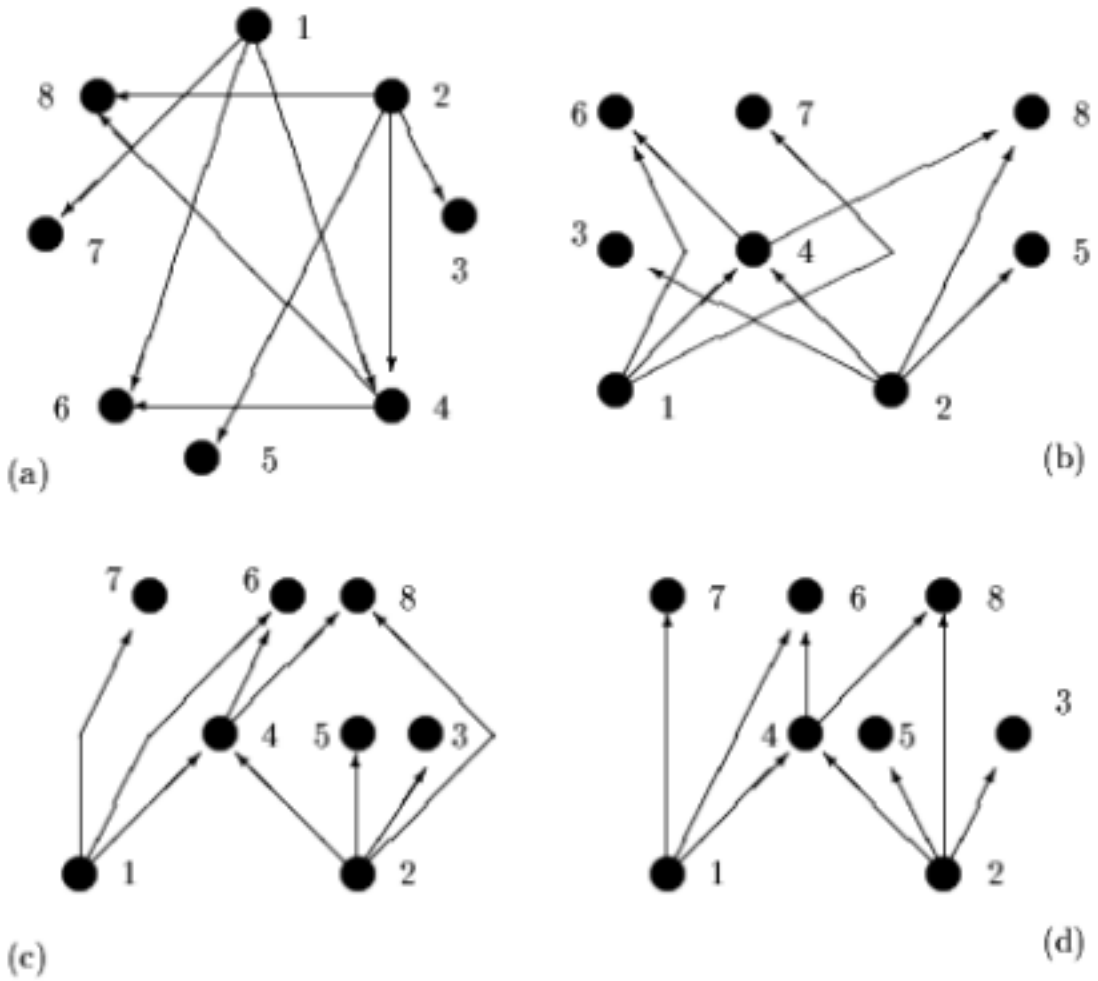


Figura 2.3: Trazado en tres fases (a) Trazado original, (b) Ordenamiento de niveles, (c) Permutación de nodos, (d) Trazado final

mizar el número de cruces de arcos.

La tercera fase del algoritmo usa el *layering* (cálculo de los niveles) y el ordenamiento de los niveles para calcular un nuevo dibujo. El objetivo final es minimizar el largo horizontal de los arcos y el numero de nodos *falsos* introducidos (lo que equivale a minimizar las polilíneas). Finalmente los arcos originales son dibujados como polilíneas, líneas rectas o *splines* interpolados desde las polilíneas.

La principal desventaja de este algoritmo, es que usando el *layering* y la introducción de polilíneas que es consecuencia de los nodos *falsos*, hace que el dibujo no sea muy legible estéticamente. Aunque se ha demostrado que en la práctica el algoritmo tiene buen rendimiento, respecto al tiempo de cálculo del trazado del dibujo.

Grafos planares

El gran restricción para este tipo de grafos es que el dibujo sea planar, es decir que no tenga cruces en los arcos. La estrategia de estos algoritmos primero consiste en probar la planaridad de los grafos, antes de calcular un dibujo. Se han encontrado algoritmos que prueban la planaridad de un grafo en tiempo lineal [2].

Cuando se calcula la planaridad de un grafo, se calcula lo que llamaremos el *embedding*, que es una abstracción de la representación física del trazado del grafo, para transformar este *embedding*, en la visualización final los algoritmos utilizan diferentes objetivos. Generalmente los arcos son dispuestos como líneas rectas o polilíneas ortogonales.

2.2.2. Algoritmos para grafos en general

Si se considera grafos en general, se tienen dos aproximaciones claras al problema algoritmos basados en topologías donde el trazado del grafo es hecho siguiendo una convención de dibujo ortogonal priorizando la estética. La otra aproximación son los algoritmos dirigidos por fuerzas.

Trazado ortogonal

El trazado ortogonal es aquel que en que los arcos son dibujados como cadenas alternantes de segmentos horizontales y verticales. Esta clase de trazado es utilizado en el diseño de circuitos VLSI.

Un metodología en tres etapas es presentada en [2] llamada *topology-shape-metrics*. El proceso de trazado ortogonal se conforma de tres fases:

Topología : El grafo se convierte en planar reemplazando los cruces de arcos por nodos *falsos*. Una vez hecho eso se calcula el trazado del grafo con los nuevos nodos, se calcula un trazado plano pero no ortogonal del mismo (ver figura 2.4).

Ortogonalidad : Los nodos son alineados de forma ortogonal cada arco del grafo es dispuesto como una polilínea de segementos ortogonales. Con la intención es minimizar el número de alternancias entre segementos verticales y horizontales de la polilínea que compone cada arco.

Compactación : El objetivo es minimizar el área del trazado, compactando el grafo todo lo posible, preserveando la forma ortogonal de los arcos del mismo.

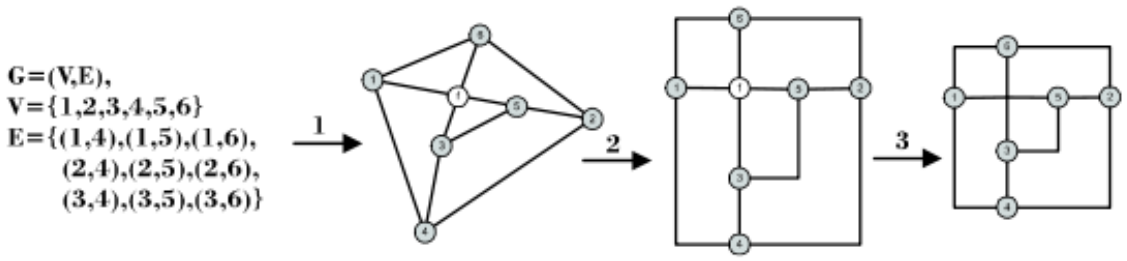


Figura 2.4: Trazado planar en tres fases (1) Topología, (2) Ortogonalidad, (3) Compactación.

Algoritmos dirigidos por fuerzas

Los algoritmos dirigidos por fuerzas son otro tipo de trazado para grafos generales, produciendo arcos con forma de rectas en la mayoría de los casos. Su nombre lo llevan gracias a que el grafo es modelado como un sistema de fuerzas donde interactúan los nodos y los arcos. La aproximación canónica para este tipo de algoritmos es el sistema de físico donde los nodos se mueven según las fuerzas que los afectan. Esta sistema trata alcanzar un mínimo de energía, un sistema en equilibrio. Las posiciones de los nodos en el equilibrio determinan las posiciones en el trazado del grafo.

Generalmente la resolución de los sistemas físicos modelados se hace mediante métodos optimización de funciones no lineales.

2.2.3. Áreas de aplicación típicas

El trazado de grafos tiene vastas áreas de aplicación. Un ejemplo familiar para el usuario de un computador es la jerarquía del sistema de archivos. Donde la usabilidad y la navegación sobre el mismo se torna crítico para a la hora de uso y productividad al trabajar con un computador. Otras aplicaciones tradicionales y familiares son : los diagramas organizacionales en una empresa o la representación gráfica de la genealogía de una familia.

Otras aplicaciones del trazado de grafos es biología y química tales como: mapas moleculares, mapas de genes, árboles filogéneticos. En computación se aplicacián para representar diagramas de clases, redes de petri, máquinas de estado, diagramas entidad-relación, UML, diagramas de entidad-relación, visualización de redes sociales.

2.2.4. Algunos problemas de la visualización de grafos

En ciencias de la computación lidiar con grandes cantidades de información es complejo ya sea en tiempo o en espacio. Teniendo esto en cuenta, el tamaño del grafo a tratar es un asunto clave en tratar de visualizar el mismo. El gran problema que trae esto es que un

grafo de gran tamaño puede comprometer o alcanzar los límites de un algoritmo de trazado utilizado, haciendo que en la práctica el cálculo el trazado se imposible con los recursos computacionales ocupados.

Y si es posible tratar con grafos de gran tamaño, si la plataforma así lo permite , aparecen para el usuario problemas de usabilidad o confusión visual, ejemplo de ello es que es difícil o casi imposible distinguir entre los nodos y los arcos del grafo. Pero es un hecho que la comprensión y el análisis de los datos modelados como grafo se ven beneficiados cuando el tamaño del grafo a visualizar es pequeño.

Capítulo 3

Trabajo realizado

Luego del estudio detallado en el capítulo anterior de algunos de los métodos clásicos para la visualización de grafos(trazado), en éste se entrega un informe detallado de sistema construido para cumplir con los objetivos planteados anteriormente.

3.1. Desafíos de diseño de la aplicación

El desafío principal de esta aplicación es obtener un grafo, previa consulta de una base de datos de genealogía; y usar un algoritmo de visualización (*layouter*) en dos dimensiones para visualizarlo y interactuar con el mismo, todo en tiempo real usando una única interfaz.

Todo esto sujeto a que la aplicación puede ser extensible y no quede sujeta a un propósito particular ni tenga funciones muy limitadas. De ahí algunos objetivos que se pueden enumerar serían los siguientes:

Tener una interfaz simple para el usuario : Tener una aplicación tal que las consultas y manipulaciones de los resultados de éstas sean simples e intuitivas tanto como sea posible para el usuario, con tal de reducir el esfuerzo para obtener información de calidad.

Permitir al usuario interactuar con los resultados de la consulta : Permitir al usuario que interactuar con la la visualización obtenida de la información para que éste obtenga un mejor entendimiento de su consulta.

Permitir que el usuario vea detalles del resultado de la consulta : Obtener información con más detalles de la consulta, especialmente a los nodos del grafo.

Permitir al usuario hacer preguntas específicas a la base de datos : Permitir preguntas específicas con el fin obtener el grafo de su interés. Es decir introducción de restricciones.

Soporte para la extensión para más tipos de trazado de grafos El programa debe ser modular, en cuanto a la futura adición de algoritmos de trazado de grafos.

La interfaz debe estar integrada a datasources : Para un acceso eficiente a los datos, pero que la misma no calcule las consultas.

Tener una aplicación que no sea específica a este problema : Una aplicación de visualización de propósito más o menos general.

Herramienta multiplataforma : La mejor manera es que sea una aplicación que opere como plugin o incrustada a un *navegador web* que tenga soporte multiplataforma.

3.2. Arquitectura de la aplicación

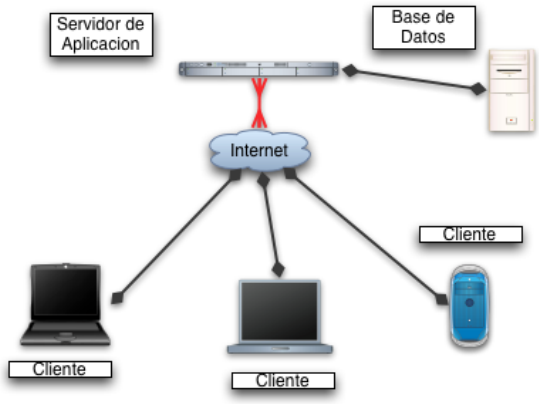
La aplicación a desarrollar está en un entorno servidor-cliente. A la primera componente se le llamará servidor de aplicación. Éste será el responsable de reponder a la consultas que generen los clientes por medio de la red; consiste en un servidor de aplicaciones J2EE. Sobre este servidor se montó un servicio web estilo Rest[11], Transferencia de Estado Representacional, para que sea independiente de la naturaleza del cliente sea éste: una aplicación de escritorio, otro servicio web, etc. El estándard de intercambio de resultados serán respuestas en formato un formato independiente construído para este propósito, similar a GraphML[12], con ello se uniformiza el formato independiente de la aplicación que procese el resultado, aunque la aplicación esté diseñada para poder responder en otro formato dependiendo del solicitado por la aplicación cliente.

Por el lado cliente se generan las consultas al servidor de aplicación, el cual procesa y despliega el resultado de la consulta utilizando una forma de visualización elegida por el usuario. Es aquí donde se visualizan los datos y se generan consultas hacia el servidor, dando la posiblidad al usuario de manipular los resultados localmente (ver figura 3.1(a) para una ilustración de la situación). Con esto se espera tener una forma ágil y amena de mostrar información de interés para el usuario.

3.3. Almacenamiento de la información

Como se dijo anteriormente, la información de la base de datos de la Federación será la piedra angular de este sistema. Para ello se ha definido que el sistema se alimente de información de forma periódica, el motivo es por eficiencia y complejidad de las consultas.

Los datos almacenados en la base de datos de Federación de Criadores del Caballo Chileno, se puede resumir en las siguientes entidades bien definidas: caballo, criadero, y desempeño de los caballos en distintas pruebas. La primera de ellas tienen información de los sujetos de las consultas, los caballos de *raza chilena* y las otras entidades sirven para cálcular índices de calidad de los mismos, lo que llamaremos **figuración**. Tales índices serán



(a) Arquitectura de la aplicación.

calculados una vez y son responsabilidad de los expertos, dado que éstas representan algunas de las características que se quieren mostrar o encontrar.

Dentro de algunos de los atributos de estas entidades se tiene:

Caballo : run (identificador único del caballo), run padre, run madre, fecha de nacimiento, run de su criadero, sexo, color.

Criadero : run del criadero, rut del propietario , nombre del criadero de caballos.

Desempeño : La distintas mediciones de desempeño de un caballo (figuraciones), que son el pilar fundamental para definir las consultas sobre los caballos y sus distintas generaciones.

Considerando lo anterior un gran desafío para completar el trabajo, fue ubicar una estructura, sistema gestor de base de datos donde almacenar la información contenida en un gran grafo, tal que ésta sea eficiente y rápida para resolver consultas sobre la misma. Este un tema no menor, ya que comercialmente existen varias alternativas de productos en el mercado y cada una de ellas con objetivos y aproximaciones distintas al mismo problema.

Como requerimiento escencial para esta parte de la solución la base de datos debía permitir almacenamiento y recuperación información de tipo estructurada.

Un posible candidato a evualuar son las base de datos orientadas a objetos, que permiten almacenar, consultar, recuperar información representable en el paradigma de programación orientado a objetos, obteniendose una representación lo más fiel a lo que sería el mismo grafo almacenado en la memoria volátil del computador. La potencia de la misma es que puede hacer persistente todo tipo de estructuración de los datos, como el usuario de la misma desee, con lo cual se puede ahorrar mucho tiempo en la recuperación de datos. Como producto se evaluó DB4O [13]. Otro posible solución al problema es la utilización de un gestor de base de datos relacional, por su amplia aceptación en el mercado y por que se puede modelar la

estructuración de información en paradigma relacional. Esta alternativa presenta ventajas como los es que orientada al dato y no a la relación que hace más fácil por ejemplo el cálculo de ciertos indicadores que se quieran de los mismos. Así como también provee interfaces de trabajo independiente de las aplicaciones o lenguajes utilizados para manejar el acceso a la información.

Para ello se pueden tomar varias aproximaciones en lo referente a la elección del gestor de base de datos. Se consideraron:

- Base de Datos Orientada a Objetos (OODBMS)
- Un Sistema Gestor de Base de Datos Relacional (RDBMS).

3.3.1. Base de Datos orientada a objetos (OODBMS)

Esta aproximación se podría resumir como toda la potencia de la orientación al objeto unida a la perisistencia de los datos encapsulada en objetos dentro de una base de datos.

Esta aproximación la podemos justificar, como lo ya expuesto anteriormente en los siguientes antecedentes:

1. Si bien el modelo relacional para las bases de datos está extendido en la industria no es el único disponible.
2. Dada la gran proliferación del paradigma de la orientación a objetos en el mundo del desarrollo y modelamiento de software, y el frecuente uso que estas aplicaciones hacen de las bases de datos, es natural pensar en un tipo de base de datos que también considerase esta forma de modelar la información.

Y la comparación natural de las aproximaciones relacional y orientada a objetos:

1. ¿Por qué tener que hacer grandes consultas para obtener la información de un sólo objeto?.
2. En un OODBMS cada objeto es almacenado independientemente, lo que hace mucho más ágil su recuperación, así como también la de aquellos relacionados con el mismo.
3. La aproximación coloquial para el lector sería: *¿Al guardar un automóvil en un garage, alguien lo desmantela para dejar todas sus partes 'ordenadas'?*. Ver comparación en la figura 3.1.

Dentro de las características que provee en este caso DB4O el OODBMS examinado ([14]) podemos mencionar: agregación de objetos, encapsulamiento, herencia, versinamiento de objetos y un acceso transparente de la información.

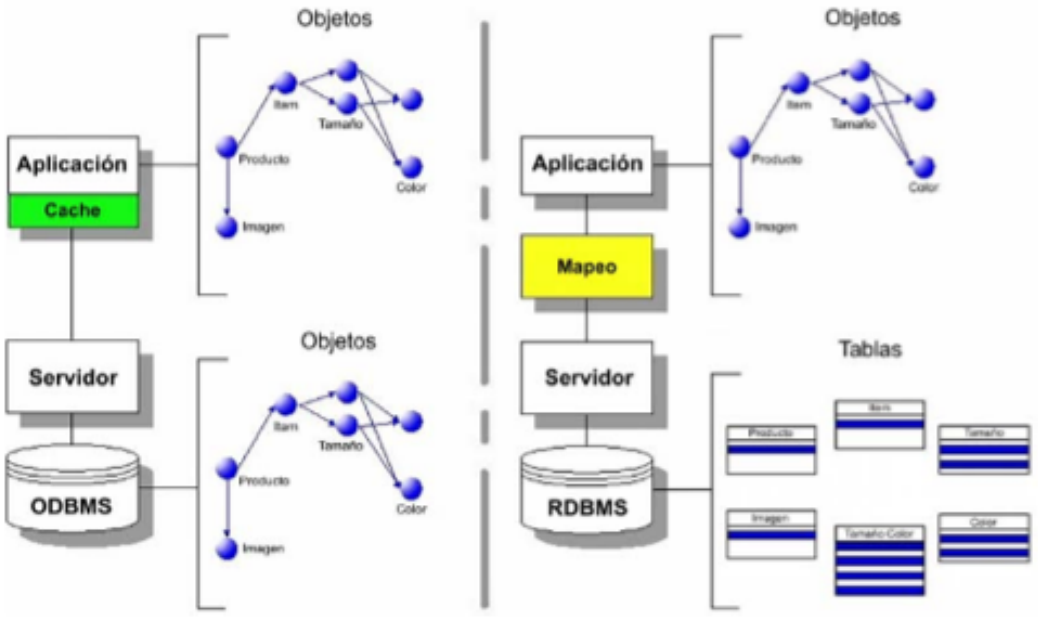


Figura 3.1: Almacenamiento directo en objetos v/s a un modelo relacional

Con todo ello el modelo de datos del problema en un OODBMS sería tentativamente, concordando exactamente con el modelo de clases del mismo, ver figura 3.2.

3.3.2. Sistema Gestor de Base de Datos Relacional

Debido a la acepatación de estos sistemas en el mercado y uso en la industria (no se detallarán los detalles técnicos de la misma). Pero para el problema a solucionar es la alternativa más apropiada desde el punto de mantenibilidad y extensión futura del modelo propuesto. La alternativa del OODBMS evaluado haría que el servidor de aplicación debiese estar construído en el mismo lenguaje (JAVA) que el gestor de base de datos en el cual se realizarían las consultas, teniéndose menos extensibilidad de la aplicación en una futura modificación.

Dentro de la elección del producto se optó por PostgreSQL [15], un gestor de base de datos relacional de código abierto de altas prestaciones y el cual cuenta con características avanzadas que fueron de utilidad para la realización de este proyecto. Entre ellas podemos mencionar:

Soporte de datos multidimensionales : El uso de arreglos multidimensionales permitió realizar una materialización de una consulta muy recurrente dentro del sistema. Esta consulta *dado un caballo obtener todos sus hijos*, era un acceso o consulta bastante costosa en tiempo de proceso (acceso a disco). La cual materializando la descendencia del caballo en un campo multivariado fue de enorme ayuda para mejorar el desempeño,

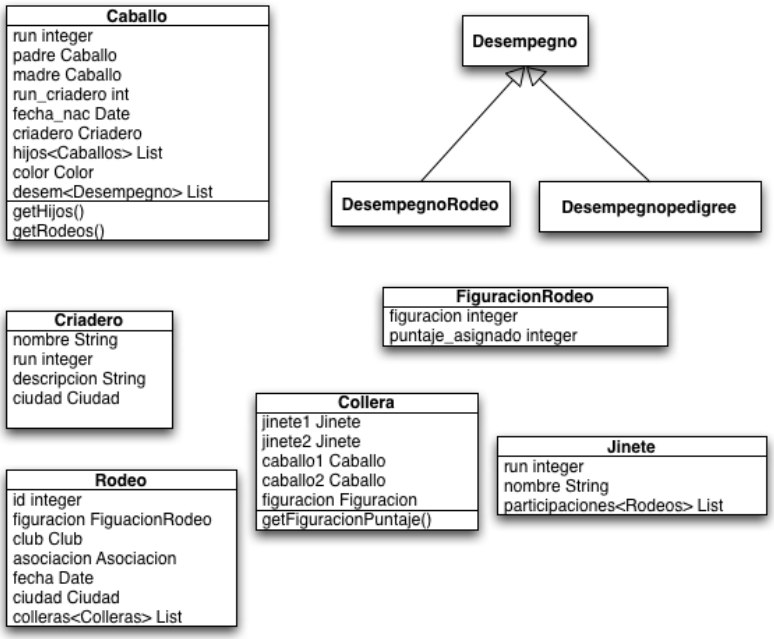


Figura 3.2: Modelo del problema en OODBMS DB40.

también lo fue la simplificación de las consultas al gestor de base de datos. El *trade-off* de lo mismo es que este el dato materializado de la descendencia de un caballo se debe precalcular cada vez que se altere la un de dato y por lo menos una vez tiene que hacerse el proceso completo para todos los ejemplares.

Soporte para índices : Con ello se tiene que se pueden bajar el tiempo empleado por ciertas consultas cuando se realizan por ciertos campos de la tabla en cuestión, dependiendo de varios factores entre ellos: si son muy consultados, la distribución de los datos, el tamaño de la tabla y del índice. Considerando que la base de datos es de sólo consulta en la práctica.

El modelo de datos relacional para el problema es el siguiente.

Caballo : Tabla (entidad) principal del modelo que mantiene aproximadamente toda la historia del registro genealógico del Caballo Chileno de la Federación. Dentro de sus atributos más importantes estan: **id** (run) del caballo, el cual es identificador único del caballo (siendo llave primaria de la tabla), **idm**: identificador o *run* de la madre del caballo, **idp** identificador o *run* del padre del caballo, y la materialización o cálculo de los hijos del caballo dentro de un arreglo de *runs (ids)*. El lector se puede preguntar que este dato no es átomico, no cumple con la primera formal normal, pero cabe destacar que dentro del estándar `ANSI SQL SQL-1999` aprobado por ISO (International Organization for Standardization) si está definido y PostgreSQL presenta soporte para ello. Dentro de los otros campos importantes que se pueden mencionar están año de nacimiento, **color**, dato importante, **sexo**.

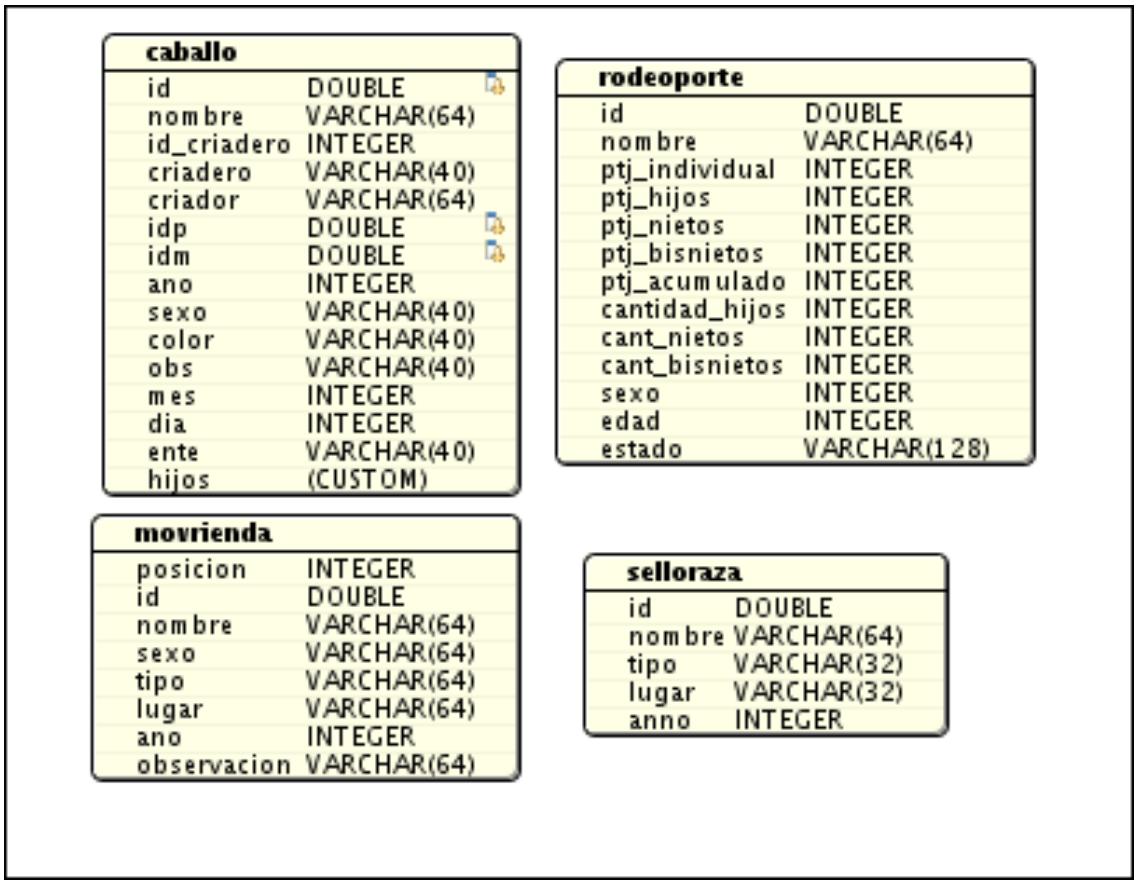


Figura 3.3: Modelo relacional del problema

Selloraza : Tabla (entidad) que contiene información sobre los premios de raza *pedigree* de los lugares que han obtenido los caballos catastrados en el registro. Dentro de su atributo más importante está el **nombre** que identifica el concurso, **id** : *run* del caballo con cierto lugar y **lugar** posición que obtuvo el ejemplar en ese concurso, por mencionar las principales. Principalmente estás pruebas miden morfología del ejemplar, de acuerdo a criterios propios definidos por la Federación.

Movrienda : Tabla (entidad) que contiene información sobre los premios obtenidos en pruebas ecuestres (lugares) de los caballos catastrados en conjunto con el jinete que realizó la prueba.

Rodeoporte : Tabla (entidad) que contiene información sobre los premios obtenidos en los rodeos organizados en el país.

3.4. Servidor de aplicación

Para el servidor de aplicacón se optó un servicio que soporte varios clientes consultando concurrentemente. Para ello se eligió por un *webservice*, que se refiere a un conjunto de protocolos y/o estándares para intecambiar datos entre aplicaciones, dada la naturaleza estándar abierto, si las aplicaciones respetan el protocolo, éstas pueden intercambiar datos sin mayores problemas independiente de la naturaleza de cada una ellas.

A continuación se presentan algunas de las ventajas de los servicios web.

3.4.1. Algunas ventajas de los servicios web

Interoperabilidad: Entre las aplicaciones se software con las que intercambian datos y más aún sobre las plataformas donde se éstas se instalen.

Modo texto: Fomentan en protocolos de comunicación de modo texto, que hace más fácil entender como funcionan y acceder al contenido, tanto como para máquinas y para humanos.

3.4.2. Algunas desventajas de los servicios web

Sin soporte para transacciones: No definen una forma de soporte de transacciones nativas, por lo cual no son naturalmente transaccionales, pero este comportamiento se puede simular en la lóagica de la aplicación. Para este trabajo cada consulta es una transacción, con ello se tuvo que esta limitante no era tal.

Bajo rendimiento: Por ejemplo si se compara con otros modelos de computación distribuida, RMI, CORBA, DCOM entre otros. Es una desventaja derivada de utilizar trans-

misión basada en texto y mayormente de la forma como se establece la conexión entre los 2 puntos mediante la transacción y transimisión de datos.

3.4.3. Implementación

Como se dijo anteriormente el servidor de consultas y/o recuperación de información, es de arquitectura Rest. Donde se optó por una forma sencilla de comunicación HTTP y XML, comparado por ejemplo con toda al abstracción servicios web SOAP.

Para la implementación se utilizó el framework RESTLET[16] que es para servicios web REST, que está construido en lenguaje de programación JAVA y que intenta ser la implementación de la especificación JAX-RS 311 : *The Java API for RESTful Web Services*[17].

La arquitectura REST propone la existencia de recursos que son accedidos utilizando un identificador único de recurso, *uri*, para la cual se expone por medio de HTTP el recurso, y donde los clientes hacen operaciones bien definidas sobre el recurso: actualizar, crear, modificar y/o borrar. Ellas son la analogía del uso de los *verbos http* : POST, DELETE, GET, PUT.

Con el uso del framework RESTLET[16], se construyó el servicio web, el cual en este caso consta de un solo recurso *caballo*. La idea del funcionamiento es muy simple, el cliente accede a un recurso, de la forma habitual REST, esto mediante el run o identificador único del caballo en la base de datos de genealogía.

$$\underbrace{/\text{recurso}}_{\text{caballo}}\underbrace{/\text{uri}}_{\text{run}}\underbrace{/?\text{accion=genealogia\&gen=2}}_{\text{parámetros de consulta}}$$

Donde *caballo* identifica al recurso, *run* identifica al caballo, y lo restante son los parámetros que recibe la acción a consultar del recurso. En la siguiente sección del capítulo se abordará cuales y cómo son estas consultas, y los *parámetros de la consulta*.

3.4.4. Descripción de la implementación del servidor

Como se mencionó anteriormente la implementación del servidor se realizó utilizando el lenguaje JAVA. Para ello se modeló la aplicación siguiendo, patrones y convenciones utilizadas por el framework RESTLET[16]. Se puede separar en partes la implementación una referente a proceso y petición del recurso, sobre todo lo que concierne a REST, otra el proceso mismo del la consulta tal como es acceso a base de datos, algoritmos de búsqueda y procesamiento de la respuesta hacia el cliente (aplicación de usuario).

El siguiente un diagrama de clases de la aplicación servidor:

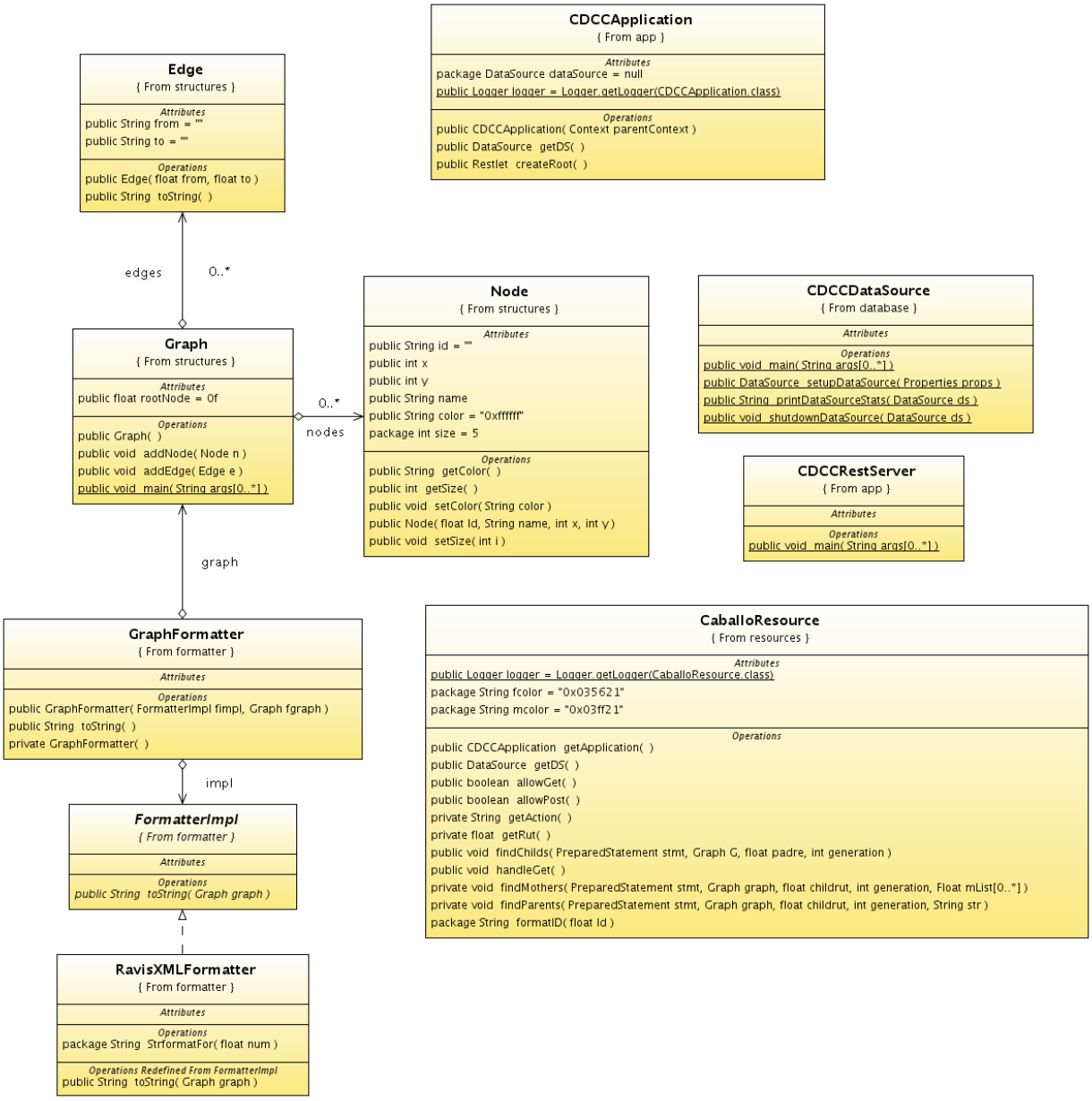


Figura 3.4: Diagrama de clases de la aplicación servidor.

Debido a la estrucuturación del código se optó por separarlo en partes bien definidas usando las propiedades de JAVA en cuanto a los llamados *packages*. Para ello se estructuró de la siguente forma:

app : Todo lo que tiene que ver con la configuración propia del framework restlet: routing, acceso a recursos, configuración de datasource e inicio del servidor de aplicación.

database : Configuración del datasource. En este *package* se tiene la configuración del acceso a la base de datos. La conexión es hecha usando un pool de conexiones usando Apache commons DBCP Datasource[18], que es una forma eficiente de conectarse a gestores de base de datos relaciones, en este caso PostgreSQL. Esta forma de conexión permite un uso eficiente de la base de datos en lo referente al *pooling*, adiministrando eficientemente el acceso al recurso mencionado.

resource : Clases que representan implementan el servicio REST propiamente tal. La clase **CaballoResource**, representa el recurso principal del servicio, la cual maneja y procesa las consultas. Es ahí donde se aplican los algoritmos de búsqueda, construcción y representación del grafo en estructuras de datos.

structures : Clases que representan el grafo como estructura de datos, entre las clases están Node, representado todo lo que es un nodo (caballo en este caso), Edge a los arcos dirigidos (relaciones de parentezco), y Graph que consiste el conjunto de nodos y arcos.

formatter : Clases que construyen representación de hipermedia portable, entre plataformas, del grafo generado por una consulta. Para ello se uso un patrón de tipo *strategy* [19], esto permite tener múltiples comportamientos, en lo que se refiere a una reprensentación multiplataforma para el grafo.

3.5. Búsqueda de la información

La búsqueda en este problema se traduce a encontrar el subgrafo conectado, tal que dentro grafo de la familia (genealogía) de caballos, dado un reproductor (padre o madre), entregue todos los hijos que cumplen con una cierta *figuración* dada. Se entiende por figuración a un indicador propio del caballo estudiado: por ejemplo *lugar obtenido un en un rodeo*, *número de champions ganados*, *número finales logradas por el ejemplar*, *ganador de prueba de morfología*, es decir sus *figuraciones*.

Se define grado k como el número de generaciones desde un padre (o madre, técnicamente un nodo) hasta un descendiente, en su misma línea sanguínea. Por ejemplo la distancia entre un *nieto* y su *abuelo* es dos.

Aplicado a este caso, las consultas se pueden resumir en términos del grado como: *Entregue el subgrafo conectado desde un nodo C_i (caballo), a sus parientes (descendientes o ascendientes) hasta un grado k que cumplan con cierta figuración.*

Definido una vez lo que es la generación o grado k se ilustran tres de los tipos de consultas soportadas por el sistema.

Descendencia : Dado un caballo, su *run* obtener su genealogía, hasta un grado k, es decir k-generaciones hacia adelante, todos los descendientes hasta la k-generación.

Algoritmo 1 Cálculo de la descendencia.

```
1: r ← run padre
2: K ← número de generaciones
3: V ← ∅ Conjunto de Nodos
4: E ← ∅ Conjunto de Arcos
5: G ← (E, V) Grafo descendencia
6: BuscarHijos(G, K, r)
7: if K = 0 then
8:    return
9: end if
10: H ← Buscar descendientes el padre r
11: h ← ∅
12: for i = 0 to H.size() do
13:    run_hijo ← H(i).run Obtener rut del hijo i
14:    V ← new Nodo (run_hijo)
15:    E ← new Arco (r, run_hijo)
16:    BuscarHijos(G, K − 1, run_hijo)
17: end for
```

Línea materna : Dado un caballo obtener la línea de materna del mismo, que consiste dado el *run* de un equino obtener las k madres anteriores desde su generación, y por cada

una de esas madres obtener, las k_1 generaciones de hijos de la misma. Acá el grado total de consulta en el peor caso sería, $k + k_1$.

Algoritmo 2 Cálculo de Línea materna.

```
1: r ← run de consulta
2: K ← número de generaciones de madres hacia atrás
3: K_1 ← número de generaciones de para la búsqueda de hijos
4: V ← ∅ Conjunto de Nodos
5: E ← ∅ Conjunto de Arcos
6: G ← (E, V) Grafo de línea materna
7: LineaMaterna(G, K, K_1, r)
8: if K = 0 then
9:    return
10: end if
11: Madres ← BuscarMadre(K, r)
12: for i = 0 to Madres.size() do
13:    run_madre ← Madres(i).run Obtener rut de madre i
14:    BuscarHijos(K_1, run_madre)
15: end for
```

Algoritmo 3 Cálculo línea de madres K generaciones atrás.

```
1: r ← run de consulta
2: M ← ∅ Conjunto de madres
3: K ← número de generaciones de madres hacia atrás
4: BuscarMadre(K, r)
5: if K = 0 then
6:    return
7: end if
8: m ← Buscar madre r
9: run_madre = m.run
10: M ← m
11: BuscarMadre(K − 1, r_madre)
```

Genealogía : Dando un *run* de un caballo obtener sus antepasados k generaciones hacia atrás.

Algoritmo 4 Cálculo de la genealogía

1: $r \leftarrow$ run de consulta
2: $K \leftarrow$ número de generaciones de madres hacia atrás
3: $V \leftarrow \emptyset$ Conjunto de Nodos
4: $E \leftarrow \emptyset$ Conjunto de Arcos
5: $G \leftarrow (E, V)$ Grafo de la genealogía
6: GENEALOGIA(G, K, r)
7: **if** $K = 0$ **then**
8: **return**
9: **else**
10: $c \leftarrow$ Buscar caballo de run r
11: $V \leftarrow$ **new Nodo** $(c.run)$
12: $run_{padre} \leftarrow c.run_padre$
13: $run_{madre} \leftarrow c.run_madre$
14: $E \leftarrow$ **new Arco** $(c.run, run_{madre})$
15: GENEALOGIA$(G, K - 1, run_{madre})$
16: $E \leftarrow$ **new Arco** $(c.run, run_{padre})$
17: GENEALOGIA$(G, K - 1, run_{padre})$
18: **end if**

Para todos los algoritmos expuestos anteriormente se tiene que *buscar* se refiere a un acceso a la base de datos (coloquialmente una consulta tipo SELECT en lenguaje SQL), ya que como se indicó anteriormente, las relaciones **hijo de** están ya previamente calculadas.

Así cada búsqueda realizada retorna el el objeto $G(E, V)$ o grafo resultante, el cual después es procesado para generar la reprensentación en modo texto, XML, del mismo. Esta representación puede ser pedida como un parámetro al recurso, y esta misma puede ser solicitada en distintos formatos. Para lograr esta capacidad se optó por usar una clase construida sobre un patrón de diseño de comportamiento [19]. Donde cada formato es implementado de forma independiente manteniendo una interfaz común, entre los distintas implementaciones de dicho formateador. En este caso sólo se implementó sólo un tipo, la interfaz *FormatterImpl* tiene los métodos a implementar para extender a futuros formatos de salida.

3.6. Aplicación cliente

Una vez ya calculada la consulta en el servidor, la respuesta en entregada a través de la red, Internet, al cliente. He ahí donde comienza la parte de proceso en el cliente procesa el grafo: lo contruye en memoria y calcula una visualización determinado a priori según el tipo de consulta, es decir aplica un *layouter*.

La idea básica de como interactúa el cliente con el servidor, se resume bien en los siguientes pasos:

1. Cliente realiza consulta al recurso del servicio web utilizando una *uri* como la revisada anteriormente. En el cliente donde define el caballo que desea consultar, algunos parámetros propios de la consulta y especificar la búsqueda propiamente tal.
2. Servidor calcula la respuesta de acuerdo al tipo de información que se pida. Para ello accede a la base de datos y obtiene resultado, el cual sería un grafo conectado con la información resultante del la consulta del usuario. Servidor convierte ese grafo a un formato interoperable con la aplicación cliente.
3. Aplicación cliente recibe el resultado proveniente del servidor, reconstruye el grafo, calcula el trazado del mismo y lo despliega de forma visual.

Es necesario recalcar que el formato en que es enviado el resultado de las consultas desde el servidor al cliente es un documento XML de factura propia para el problema, debido a que se evaluaron diferentes alternativas:

1. Pajek [20] es un formato de la aplicación con el mismo nombre, que está diseñada para el tratamiento de grandes redes. Pero que para este caso estaba limitado por que no soportaba definición de nodos con más de un atributo, y además la intención de la aplicación que lo procesa es para el análisis cuantitativo de la red obtenida y no principalmente para visualización. Otro alcance es que la herramienta Pajek, se perfila para el uso de científicos y usuarios técnicos que analizan redes desde distintas características de la misma y no para usuarios finales, dicho de una manera coloquial.
2. GraphML : Formato de intercambio de grafos en XML, quizás la mejor alternativa, pero se optó por una solución propia debido a que de todas formas había que extenderlo para tener nodos con muchos atributos.

Un ejemplo de grafo en XML retornado por el servidor es :

Listing 3.1: Grafo en XML generado por la consulta.

```
<?xml version="1.0" encoding="UTF-8"?>
<cdccml>
 <root>8859.00</root>
 <n i="0" N="MAQUINITA" r="8859.00" f="1" c="1" />
 <n i="1" N="VEGUILLITA" r="11698.00" f="1" />
 <n i="2" N="SOPANDA" r="13346.00" f="1" />
 <n i="3" N="ZANGANO" r="22606.00" f="1" />
 <n i="4" N="MANICERA" r="23924.00" f="1" />
 <n i="5" N="COILERA" r="36096.00" f="1" />
 <n i="6" N="AROMO" r="37550.00" f="1" />
 <n i="7" N="BANDOLERA" r="16419.00" f="1" />
 <n i="8" N="CARPETA" r="22520.00" f="1" />
 <n i="9" N="CHCHUFLETA" r="30550.00" f="1" />
 <n i="10" N="CASTELLANA" r="33302.00" f="1" />
 <n i="16" N="CUESTA" r="25932.00" f="1" />
 <n i="17" N="ODIADA" r="37289.00" f="1" />
 <n i="18" N="PASCUALITO" r="61939.00" f="1" />
 <n i="19" N="ENDEMONIADA" r="66989.00" f="1" />
 <n i="20" N="PARCELITA" r="73953.00" f="1" />
 <e s="0" t="1" />
 <e s="0" t="2" />
 <e s="2" t="3" />
 <e s="2" t="4" />
 <e s="2" t="5" />
 <e s="2" t="6" />
 <e s="0" t="7" />
 <e s="7" t="8" />
 <e s="8" t="9" />
 <e s="8" t="10" />
 <e s="7" t="16" />
 <e s="7" t="17" />
 <e s="17" t="18" />
 <e s="17" t="19" />
 <e s="17" t="20" />
</cdccml>
```

En la anterior representación los nodos están representados por el elemento **n** y los arcos en este caso dirigidos por el elemento **e**. Para el elemento **n** y sus atributos **i,N,r,f,c** representan el identificador único del nodo, nombre del caballo, run del caballo, **f**, si cumple con la figuración pedida, **c** representa un color en la visualización, respectivamente. El elemento **e** y sus atributos **s** y **t**, representan el nodo de origen del arco, y el nodo de destino del arco, respectivamente identificados por el identificador del nodo **i**.

3.6.1. Implementación aplicación cliente

La implementación de la aplicación cliente se hizo en el lenguaje JAVA por diversas razones:

1. Por contar con un Toolkit gráfico desarrollado, documentado y maduro como es Swing, que además incorpora elementos de diseño que facilitan la construcción de aplicaciones con interfaz gráfica.
2. Capacidad de conexión a red incorporadas en el lenguaje, así como también capacidad de interoperabilidad con plataformas de distinta naturaleza.
3. Bibliotecas de manipulacíon de información estructurada como red o grafo, en este caso JUNG *Java Universal Network/Graph Framework* [21], la cual tiene años de desarrollo y probada reputación en diferentes campos de aplicación.

Es necesario señalar que la mayoria de la interfaz cliente está construida con un asistente integrado en el Entorno de programción Netbeans de Sun MicroSystems©. La cual con los asistentes de generación de código y herramientas gráficas de generación de interfaces acelera la creación de aplicaciones.

Lo más importante en el cliente es la visualización del grafo generado por el servidor, previa consulta, empleó JUNG[21], siendo más que una biblioteca de software es un framework para la manipulación y exploración de dato estructurados como grafo. Para ello provee varias caraterísticas:

1. Implementaciónes génericas y abstractas de lo que es un grafo (uso de Generics lenguaje JAVA). El propio usuario o desarrollador puede crear un tipo de grafo acorde a su necesidad.
2. Implementación eficiente para distintos tipos de trazado de grafos, tanto como para grafos generales y también como para tipos específicos como árboles.
3. Implementación de renders génericos para el despliegue de los grafos, con la potencia de definir como se quiera la forma de los vértices, tamaños de los arcos, color, transformaciones, etc.
4. Soporte para manipulación del grafo una vez construido: capacidad para agregar arcos, vértices, generar subgrafos e hipergrafos.

En la implementación se trató de separar lo más posible la parte de transacciones hacia el servidor de consultas y la interfaz gráfica. Donde la mayoría de los módulos (clases hablando programáticamente) sólo se ocupan de una tarea específica, por ejemplo:

1. Conexión al servidor de consulta: para envío y/o procesamiento de las consultas del usuario, tanto en la búsqueda de ejemplares como en las de índole genealógica.
2. Procesamiento de los grafos, cálculo de layouts y render de la representación del trazado del mismo.
3. Propias interfaz gráfica: definición de controles, eventos de usuario, diálogos de texto, etc.

El siguiente un diagrama de clases de la aplicación cliente:

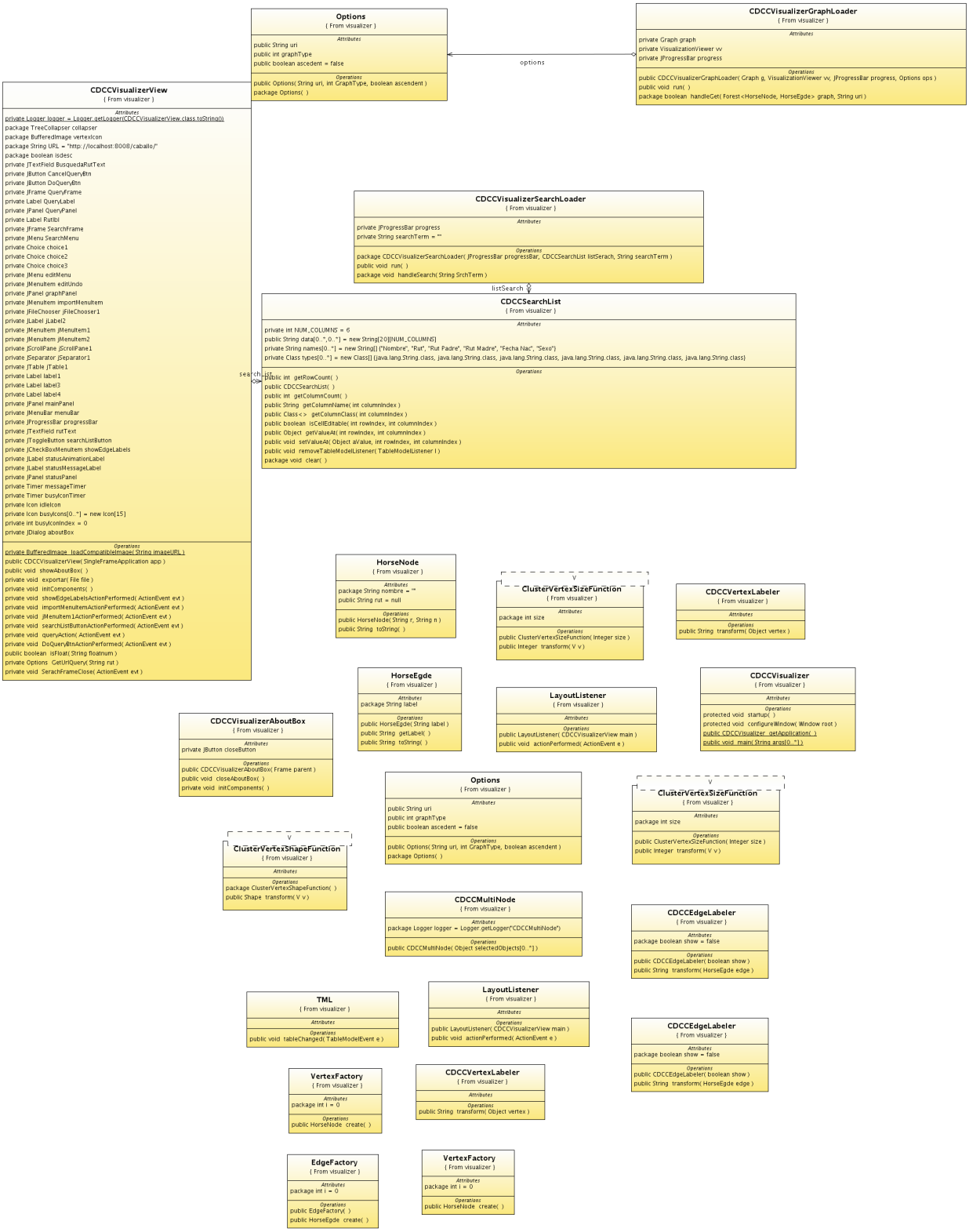


Figura 3.5: Diagrama de clases de la aplicación cliente.

Debido a la estrucuturación del código se optó por separarlo en clases bien definidas. Para ello se estructuró de la siguente forma, resaltando las más relevantes:

Interfaz : CDCCVisualizer encargada de la creación de dialogos de texto, ventanas de trabajo para el usuario. CDCCSearchList construye el área de trabajo para realizar búsquedas. Y otras clases anexas usadas por el render para decorar el grafo, labelers de nodo y arcos, render de nodos y arcos, etc.

Comunicación de datos : CDCCVisualizerGraphLoader y CDCCVisualizerSearchLoader, envían las consultas genealogicas y de búsqueda de ejemplar al servidor, respectivamente. CDCCVisualizerGraphLoader procesa los resultados y envia al Layouter el grafo generado para su trazado.

Representación del grafo : Se ocupó implementaciones propias arcos y nodos (HorseEdge y HorseNode) que son creados por la capa de comunicación y agregados al grafo, una estrucutura génerica de la biblioteca JUNG, a la cual se le aplican los métodos de trazado (layouters).

3.6.2. Vistas de la aplicación cliente

Ventana principal

Esta es el área de visualización del trazado del grafo. Es donde se puede manipular el grafo zoom y paneo. También es la ventana principal a la cual se le anexaron menús con distintas acciones:

- Barra de menú Acción : Donde se puede acceder a la ventana de consulta y ventana de búsqueda.
- Barra de menú Acción : Inicio y control de la aplicación en este menú se puede exportar el layout obtenido a una imagen de la visualización del trazado obtenido de la consulta.
- En el menú efectos se puede alterar el comportamiento de los etiquetados de los arcos.

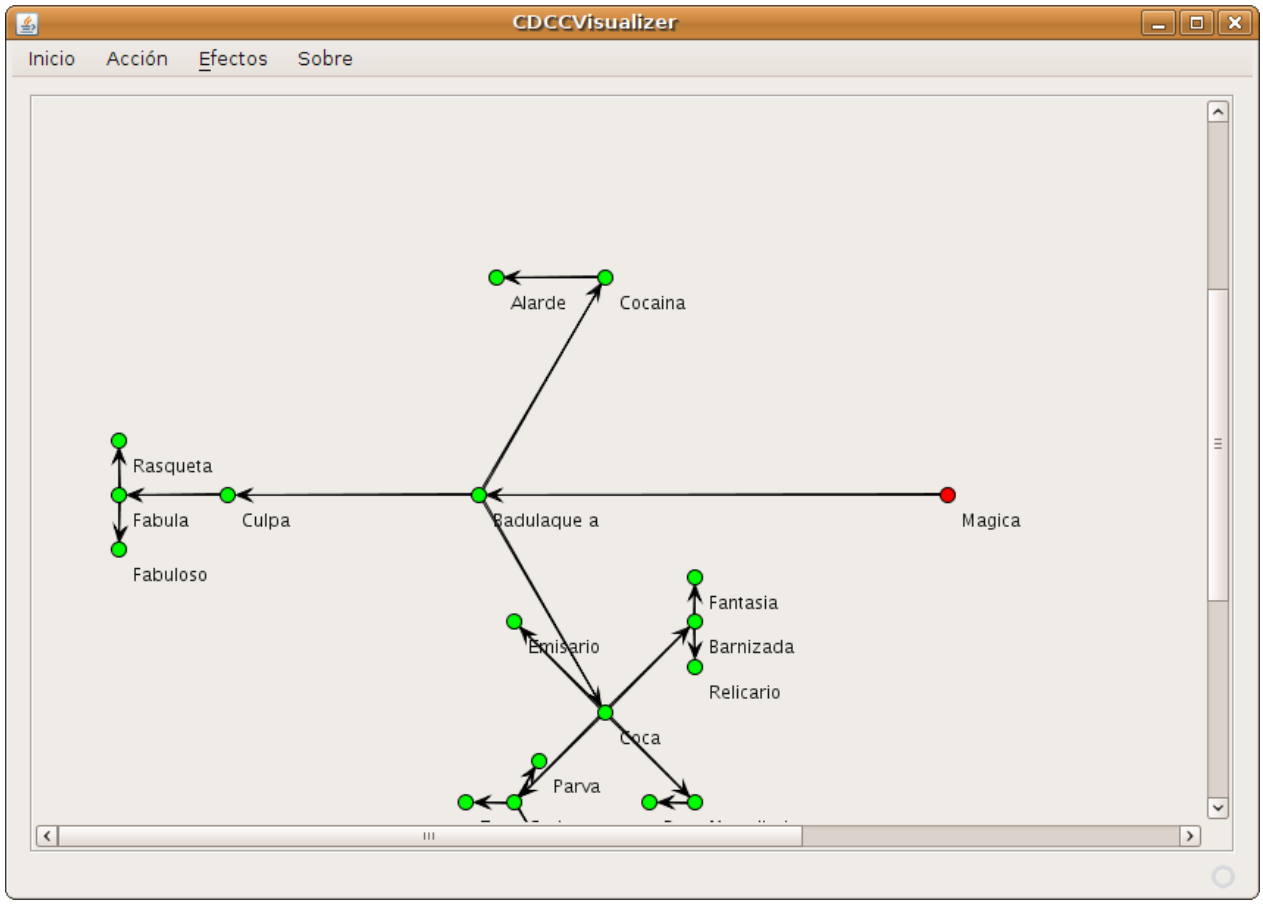


Figura 3.6: Ventana principal

Ventana de consulta

Ventana que permite ingresar los parámetros de consulta de tipo genealógica como : generaciones, filtros y son el tipo de consulta. Es aquí donde enviando la consulta se obtiene el grafo en formato XML desde el servidor. Dentro de las opciones de la ventana se tiene:

- Rut de equino : Es donde se ingresa el identificador único del caballo para consultar dentro del registro.
- Tipo de consulta : Es el tipo de consulta pedida por el usuario; Descendencia, línea materna o genealogía.
- Generaciones : Define el grado k, definido anteriormente, para la consulta elegida en la opción anterior.
- Filtrar por : Implica que tipo de filtro el usuario desee aplicar al ejemplar de su consulta: finalista de rodeo, si llegó a clasificatorio, o si obtuvo al menos dos victorias en rodeos de nivel nacional.

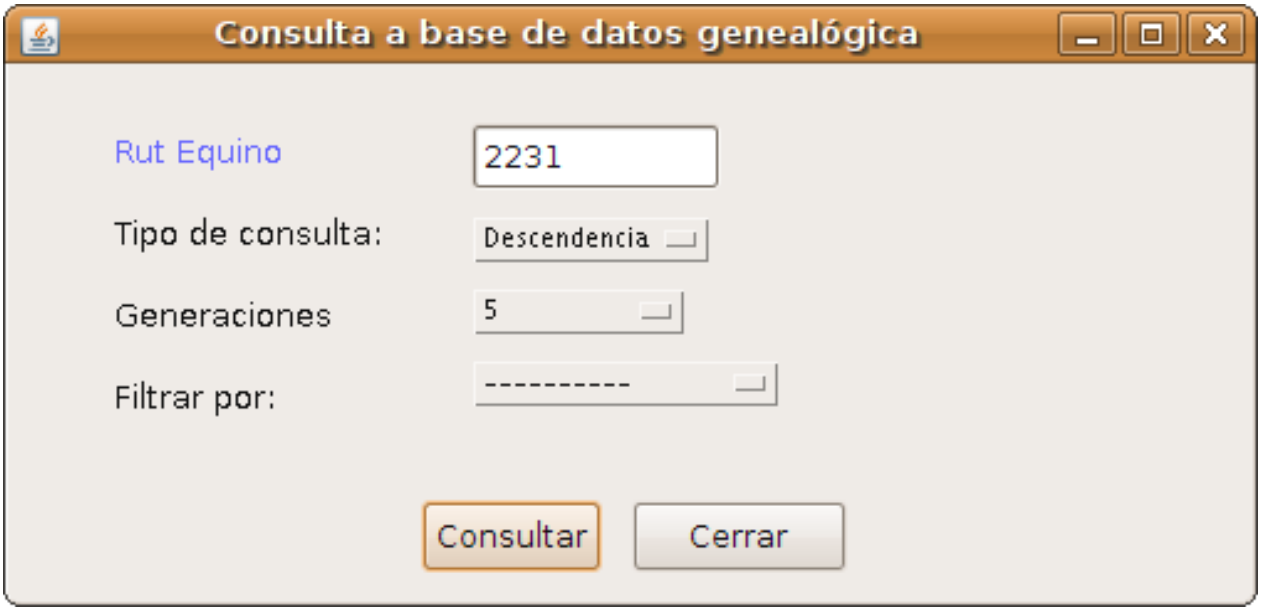


Figura 3.7: Ventana consulta

Ventana búsqueda de ejemplar

Buscar por nombre un ejemplar y recuperar ,si existe en el servidor, su run para poder hacer una consulta de índole genealógica.

El usuario debe ingresar un término el cual será usado para hacer una consulta por nombre de en la aplicación servidor, pulsando sobre uno de los resultados, el usuario obtendrá el *run* para la consulta usando la interfaz anterior.

Figura 3.8: Ventana búsqueda de ejemplar

3.6.3. Trazados de grafos utilizados en la aplicación

Claramente la relación *es padre de* o *es hijo de* convierte a todos los resultados de consultas exitosas en un tipo definido de grafo jerárquico de tipo árbol.

En la aplicación cliente se optó por el la utilización de trazados de grafos fijos dependiendo de la consulta asociada se utilizaron para la descendencia, genealogía y línea de materna respectivamente son los siguientes *layouters*:

1. Polar, BalloonLayout.
2. Jerárquico.

Polar (BalloonLayout)

En esta disposición se puede lograr mostrar de forma ecuiánime líneas cosanguíneas de la descendencia, construyendo un *spanning-tree* desde cada descendiente. Tiene la potencialidad de agrupar descendientes tales que pueden mostrar diferencias siginificativas en cuanto a características, en este caso emphfiguraciones. Ver 3.9. Variantes o mejoras de la misma se pueden encontrar en [22].

El algoritmo trabaja de la siguiente forma:

1. Primer paso toma la raíz del árbol y la ubica como centro del trazado. Para la misma define una restricción en el trazado el padre está al centro y los hijos alrededor en (coordenadas polares) ubicados a un distancia radial equivalente para cada arco, e imponiendo un estética de dibujo de resolución angular, donde el ángulo de incidencia de cada hijo es el mismo al padre, por ejemplo, para cuatro hijos el ángulo es de 90 grados. Un ejercicio para entender la resolución angular es que si se unen los arcos de hijo a hijo se formaría un polígono regular de n aristas, maximizando el ángulo mínimo entre dos arcos incidentes al mismo nodo.
2. El cálculo se repite recursivamente para cada hijo con sus hijos hasta que, el hijo tomado como raíz ya no tenga más hijos, por cada iteración del algoritmo se define un radio para disponerlo alrededor del padre.
3. El radio al que se sitúa el hijo se escoge de manera de minimizar el cruce entre arcos y minimizar el área del trazado.

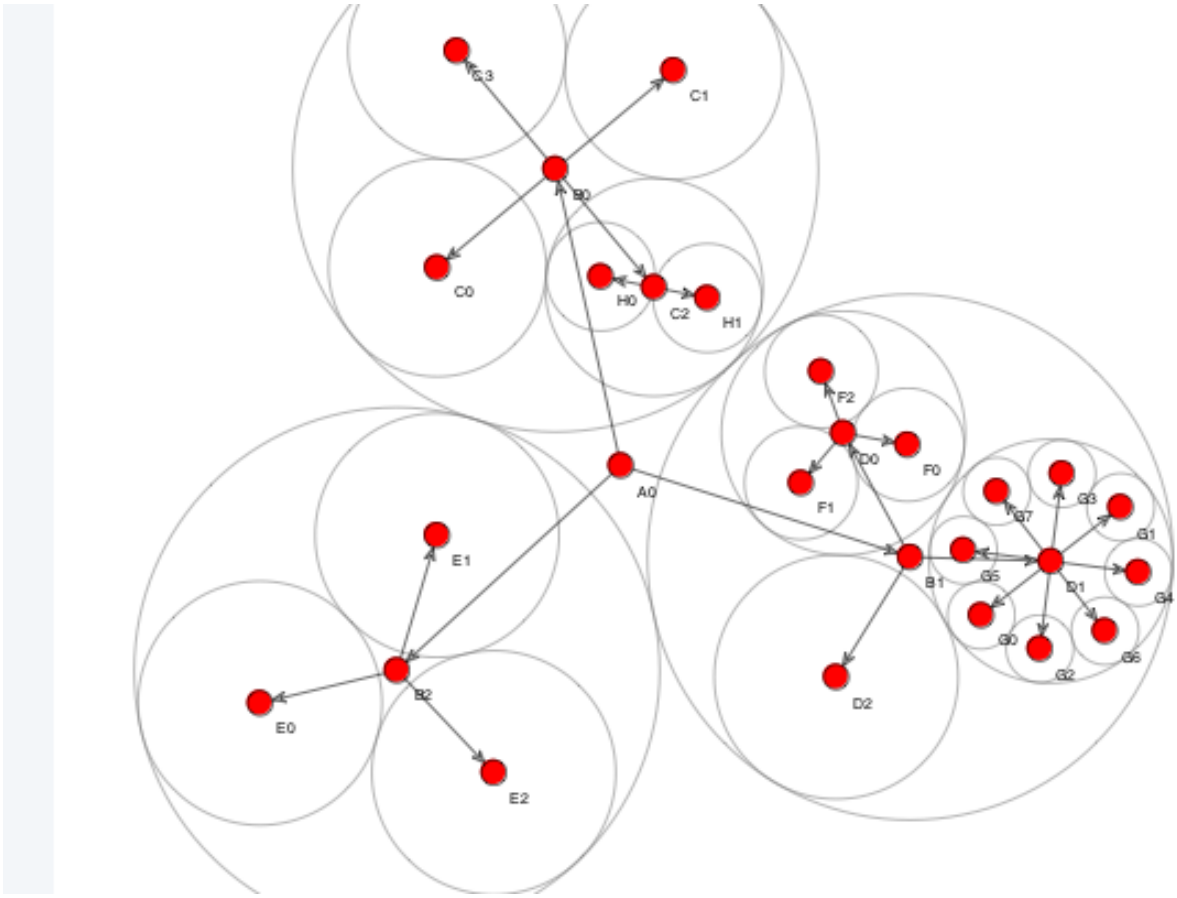


Figura 3.9: Disposición BallonLayout

Jerárquico

Con esta disposición se puede lograr mostrar de forma jerárquica la información dejando en claro: la generación y las líneas consanguíneas de la descendencia o genealogía dependiendo de la consulta. Ver la figura 3.10, sólo como referencia.

Las convenciones del trazado son las mismas mencionadas que en el capítulo anterior, por ejemplo:

1. La raíz del árbol queda al centro.
2. La altura considerada de un nodo en el mismo nivel que otro es la misma para todos. Los hijos están a la misma altura del padre.
3. Los nodos padres están centrados sobre sus hijos.
4. Los nodos en cada nivel horizontal deben ser dispuestos para minimizar la separación entre los mismos pero no deben sobreponerse, en lo posible.
5. Los arcos no deben cruzarse, el dibujo debe representar la planaridad del árbol.

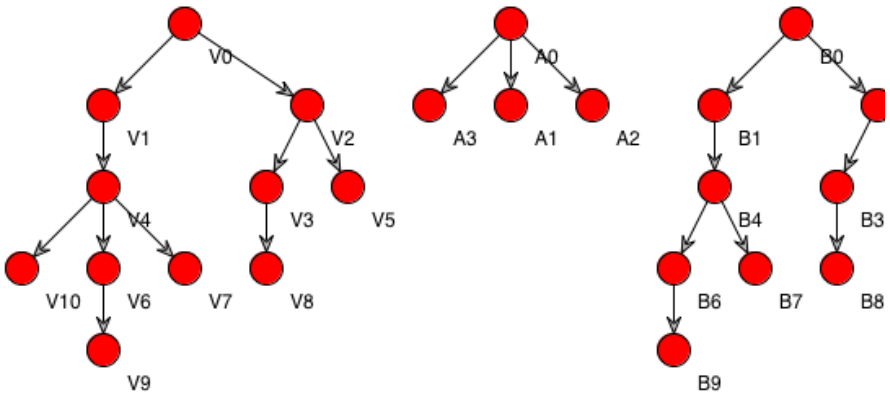


Figura 3.10: Disposición jerárquica

Capítulo 4

Resultados y Discusión

En el capítulo anterior, se planteó implementación del sistema que responde a los requerimientos planteados para el problema. En este capítulo se entregan los resultados de la utilización de la aplicación contrastando en algunos casos con ideas de bibliografía existente.

4.1. Evaluación de aplicación servidor

Para poder medir la efectividad del servidor procesando consultas se midió frente a la consulta más compleja en cálculo, que es *descendencia*. Se tiene que la consulta *descendencia* es complicada por que se vió que para caballos machos reproductores, esta retorna en grafos en cuanto a la cantidad de nodos generados y por consiguiente un gran número de descendientes. Recordemos esta consulta calcula toda la descendencia de un caballo de un sujeto hasta un grafo k, es decir, k-generaciones.

Para el experimento se utilizó una consulta para la cual experimentalmente se consigue un gran grafo, para esto se utilizó un ejemplar macho con mucha descendencia, que genera un grafo con muchos arcos y nodos, lo que se traduce a un reproductor con muchos hijos desde la primera generación y consiguientemente más descendientes en las generaciones que le preceden. Como se había visto la aplicación servidor que retorna las consultas del usuario, se impuso como requerimiento que ésta sea multitusuario, es decir de acceso de concurrente entre varios clientes. Para simular esta situación se simuló un cliente que realiza una 5 consultas concurrentes sobre el ejemplar **Morocho**, que tiene como identificador $run = 43789$, dentro del registro genealógico. Recordando la consulta etilo REST se tiene que para consultar por su descendencia con diez generaciones, se tiene la siguiente consulta.

$$\underbrace{/\text{caballo}}_{\text{recurso}}\underbrace{/\mathbf{43789}}_{\text{run}}\underbrace{?\text{action=childs\&gen=10}}_{\text{descendencia, } k=10}$$

El experimento se simuló mediante un *benchmark* utilizando el programa Apache Benchmark, herramienta que sirve para simular accesos concurrentes de clientes a un servidor HTTP. Todo esto con el fin de mostrar que el servicio es capaz de procesar consultas concurrentemente y evaluar su uso frente alta demanda.

Se realizó el siguiente experimento, simula 100 conexiones con una concurrencia de 5, a la misma consulta.

Listing 4.1: Ejecución Simulación sintética de 5 peticiones concurrentes servidor

```
ab -n 100 -c 5 http://www.caballoyrodeo.cl/caballo/43789?action=childs&gen=10
```

Listing 4.2: Simulación sintética de 5 peticiones concurrentes servidor

```
Document Path:          /caballo/43789?action=childs&gen=10
Document Length:        819524 bytes

Concurrency Level:      5
Time taken for tests:   52.466424 seconds
Complete requests:      100
Total transferred:      81984800 bytes
HTML transferred:       81952400 bytes
Requests per second:    1.91 [#/sec] (mean)
Time per request:       2623.321 [ms] (mean)
Transfer rate:          1525.99 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   29  17.1      26      96
Processing:  1189 2565 730.3    2545    5170
Waiting:      661  841 139.6     800    1404
Total:       1236 2594 726.9    2574    5199
```

Para una consulta normal al servicio se eligió un ejemplar hembra, el cual representa también un tipo de consulta de interés para el usuario. Bajo las mismas condiciones 5 conexiones simultáneas al servicio, 20 veces lo que hace un total de 100 peticiones, pero con una descendencia de 4 generaciones . Se probó con el ejemplar **Maquinita** $run = 8859$, un ejemplar antiguo si se considera el año de nacimiento. El grafo generado es pequeño 38 nodos y 37 arcos.

Listing 4.3: Simulación sintética de consulta normal.

```
Document Path:          /caballo/8859?action=childs&gen=4
Document Length:        2573 bytes

Concurrency Level:      5
Time taken for tests:   0.863608 seconds
Complete requests:      100
Total transferred:      278100 bytes
HTML transferred:       257300 bytes
Requests per second:    115.79 [#/sec] (mean)
Time per request:       43.180 [ms] (mean)
Transfer rate:          313.80 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     8   41  29.5     35     139
Waiting:        8   40  29.4     35     139
Total:          8   41  29.5     35     139
```

4.1.1. Discusión de resultados evaluación servicio de consulta

En el test (4.2) se observa (Total transferred) alta transferencia de datos donde cada consulta genera más o menos 780 *kilobytes* de información, lo que corresponde a un grafo con 3782 nodos y 3781 arcos, con lo que se hace patente que es una consulta con gran cantidad de resultados. Debido al uso de XML, como formato de salida, se ve un alto costo en la transferencia de datos hacia el cliente, lo que se traduce en alto uso de ese recurso. Este es una de las consultas más grandes e intensivas en procesamiento y uso recursos, pero como ejemplo de *peor caso*, el resultado fue satisfactorio, promedio 5 segundos en procesar una respuesta concurrente.

Con ello queda probada la eficacia del servicio de consulta, aunque las pruebas son sintéticas, son útiles para probar una gran demanda de consultas concurrentes con uso intensivo de recursos por ambos extremos, tanto como para el cliente como para la aplicación servidor.

4.2. Evaluación de aplicación cliente

Para la situación de la evaluación de la aplicación cliente la consideración de buenos resultados se puede considerar de carácter cualitativo y cuantitativo. Para los criterios de evaluación cualitativo se tiene en este caso el uso de trazado de grafos (uso de *layouters*) y como éstos son aplicados a los grafos generados frente a las consultas de los usuarios: considerando ésteticas y restricciones de trazado impuestas para cada tipo de consultas, legiblidad del trazado. Para los criterios cuantitativos se tiene por ejemplo el tiempo y responsibidad al usuario el procesamiento del trazado del grafo.

4.2.1. Tipos de consultas

Como ya se mencionó anteriormente se tienen tres tipos de consultas bien definidas:

Descendencia : Es el cálculo del grafo de hijos de un equino desde un ejemplar dado hasta las k-generaciones que le prosigan. Para ello se escogió una disposición de trazado *polar* para el grafo, para dar una restricción estética de que las líneas de descendencia cosanguíneas queden ubicadas *cerca* cuando la relación *es padre de* o *es hijo de* se cumpla. Imponiendo como restricción estética que el emplar de consulta esté al centro del trazado.

Genealogía : Es el cálculo del grafo de antepasados de un ejemplar, es decir, sería un grafo de tipo *árbol*, donde cada rama define al padre o la madre y así consecutivamente k generaciones hacia atrás.Imponiendo como restricción estética que el emplar de consulta esté al centro del trazado.

Línea Materna : Es el cálculo del grafo de los antepasados hembras de un ejemplar, k generaciones hacia atrás, escogiendo como en genealogía sólo a la madre. Además por cada madre en cada generacíon se obtiene la descendencia k_1 generaciones. Imponiendo como restricción estética que el emplar de consulta esté individualizado en el trazado, de tipo *árbol*n , y que a su vez se muestre a la línea materna generada.

Para las mismas se definió a priori tres tipos de restricciones, *figuraciones*, en la consulta, todos de tipo binario, es decir, si cumplen o no con cierto criterio:

Clasificatorio : Indica si el ejemplar llegó a una etapa clasificatoria dentro de los rodeos más importantes del país.

Finalista : Indica si el ejemplar fue finalista en un rodeo de índole nacional.

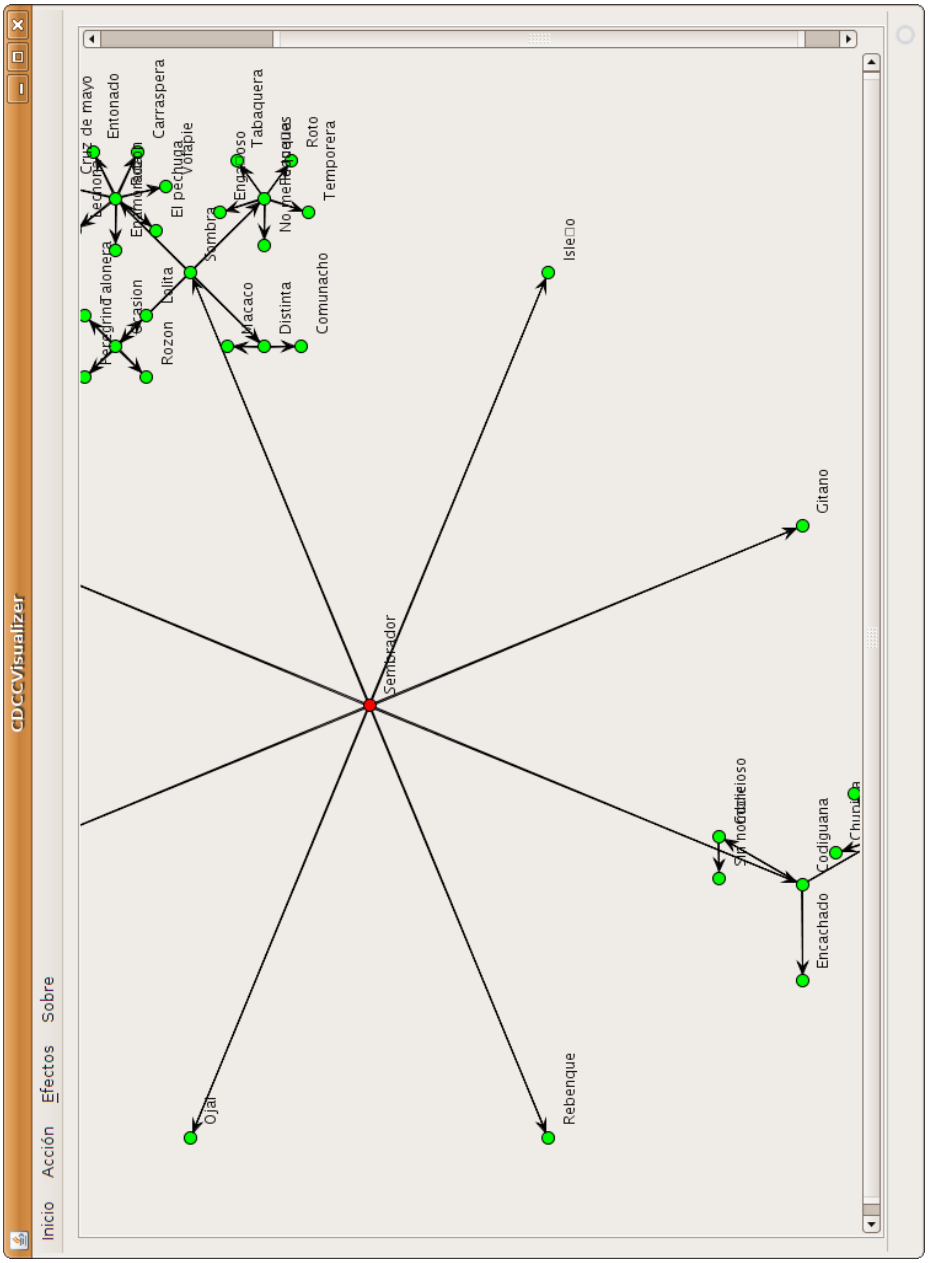
Ganador más de una vez : Indica si el ejemplar ganó más de una vez una final de rodeo de índole nacional.

4.2.2. Consulta de descendencia

Ya descrita la consulta de descendencia se procederá a mostrar los resultados para ejemplares de distinto sexo, ya que esto es un factor determinante observado en el desempeño de la aplicación. Y también es un hecho registrado en la base de datos genealógica.

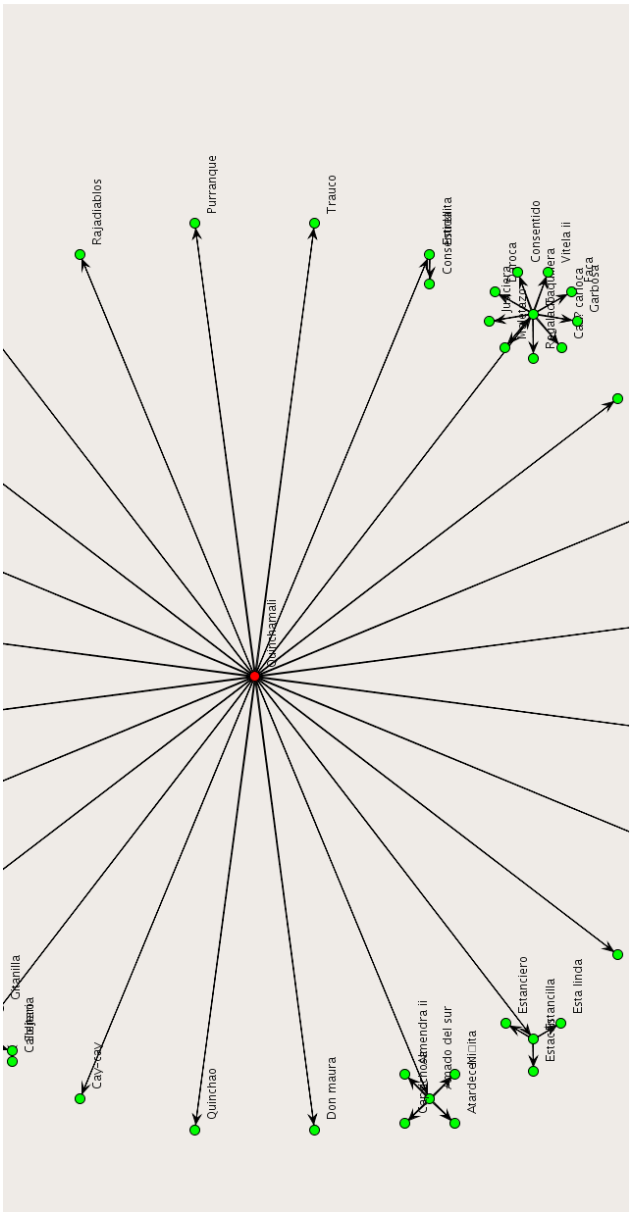
Descendencia ejemplar reproductor

. En este ejemplo se aplicará la consulta sobre una macho, **Sembrador** ($run = 52454$). Como se puede ver en la figura adjunta tomando cuatro generaciones; Clara mente se puede observar que dada la disposición del trazado polar es fácil identificar las líneas de más prolíficas de este reproductor como son : **Codiguana** ($run = 58796$), y **Sombra** ($run = 58794$). Este ejemplo demuestra un reproductor con poca descendencia, ver figura (4.1(a)).



(a) Descendencia reproductor, ejemplar Sembrador $run = 52454$ reproductor.

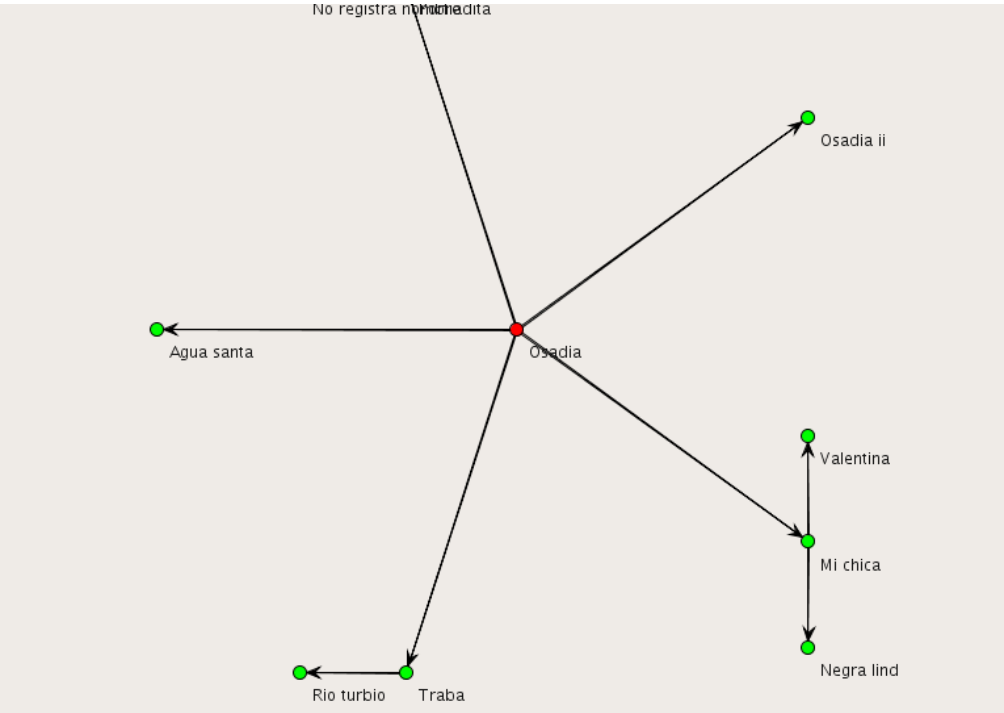
En este ejemplo se aplicará la consulta sobre una macho, **Quinchamalí** ($run = 66689$), tomando cuatro generaciones, donde se puede apreciar claramente que la línea de descendiente machos generalmente engendra más hijos que las descendientes hembras, ya sea en primera o segunda generación. Ver figura (4.1(b)).



(b) Descendencia reproductor, ejemplar Quinchamalí ($run = 66689$)
.

Descendencia para una ejemplar hembra

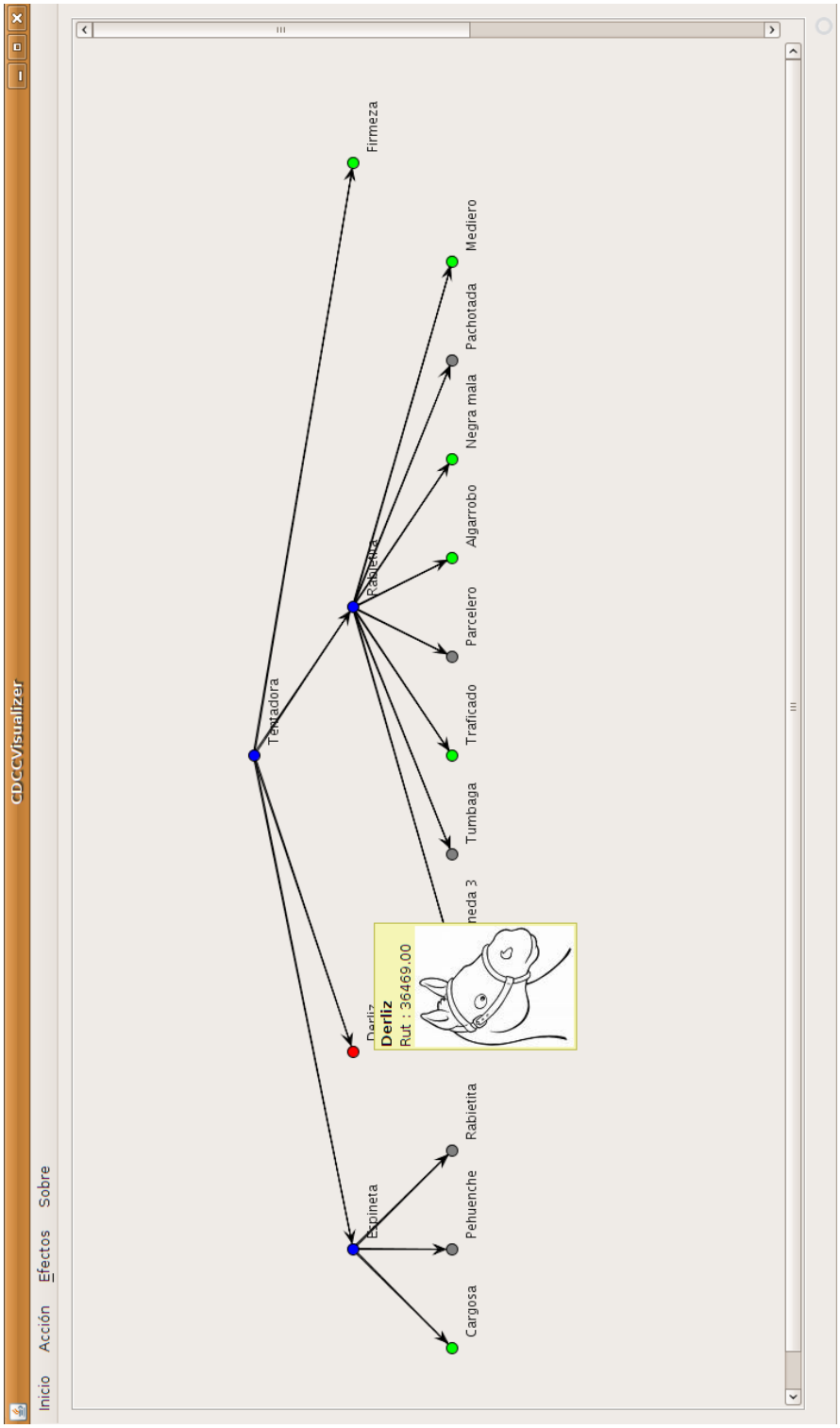
Para la consulta sobre descedencia para una hembra tomando cinco generaciones, como se mencionó anteriormente, se ve que los descendientes con más hijos son los machos, el cual hace patente un hecho que ocurre en la naturaleza. Ver figura (4.1(c)).



(c) Descendencia hembra, ejemplar Osadía $run = 79879$

4.2.3. Consulta de línea materna

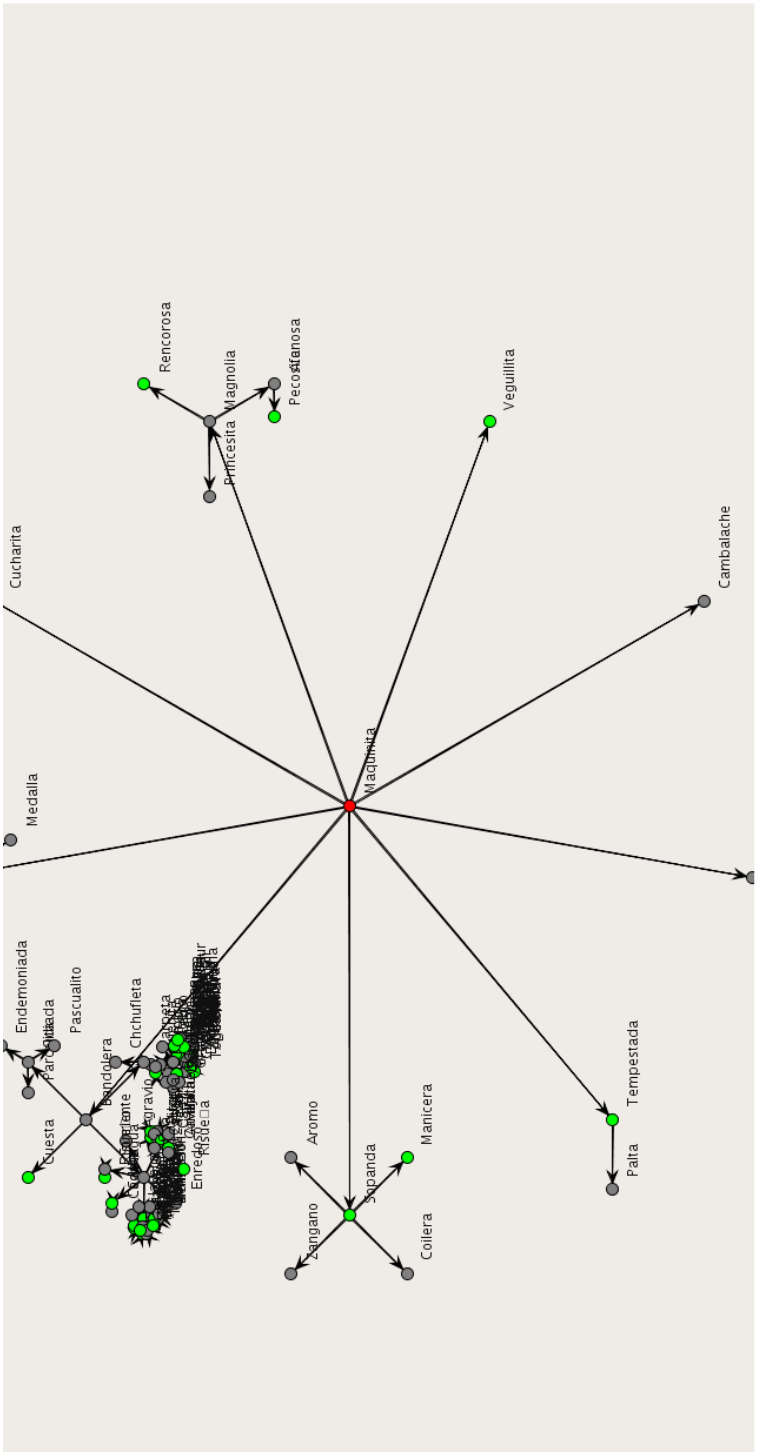
Para la siguiente consulta una de las que genera grafos de mayor cantidad de nodos. Se tiene una disposición de árbol, la línea materna se caracteriza por los nodos tachados de azul, y el sujeto de consulta en rojo. Los restantes en gris son los ejemplares que para la consulta no poseían la figuración requerida en la consulta, en este caso *clasificatorio*. Búsqueda madre *Derliz run* = 36469. Ver figura (4.1(d)).



(d) Línea materna para tres geneaciones, Derliz $run = 36469$.

Al igual que la consulta de descendencia ésta tiene el mismo problema que en los resultados obtenidos en la consulta de descendencia, se puede obtener un grafo resultante muy grande para las capacidades de la aplicación. Esas capacidades se refieren a la confusión visual que produce en el usuario, al disponer la visualización del trazado del grafo en la pantalla (dispositivo de despliegue), tomando en cuenta las dimensiones de la misma (ancho y alto). También se tiene que el uso de recursos en la máquina del cliente son intensivos en CPU y memoria gráfica utilizada por la aplicación en el momento de cálculo y despliegue del trazado.

Para ilustrar lo anterior se muestra la consulta de descendencia para ilustrar el caso, con el ejemplar **Maquinita** $run = 8859$, que tiene como hijo al ejemplar **Quinchamalí** ($run = 66689$) en quinta generación, para la consulta $k = 6$ y filtro *clasificatorio*. Cuando se visualiza el trazado del grafo, como se ve en la figura (4.1(e)), claramente el tamaño del dispositivo (pantalla importa) y un poco de confusión visual aparece, para salvar esta situación se implementó una característica que el usuario pueda colapsar esos nodos de alta incidencia de arcos en uno solo, como medida para evitar despliegues innecesarios y que puedan confundir la exploración de su visualización.

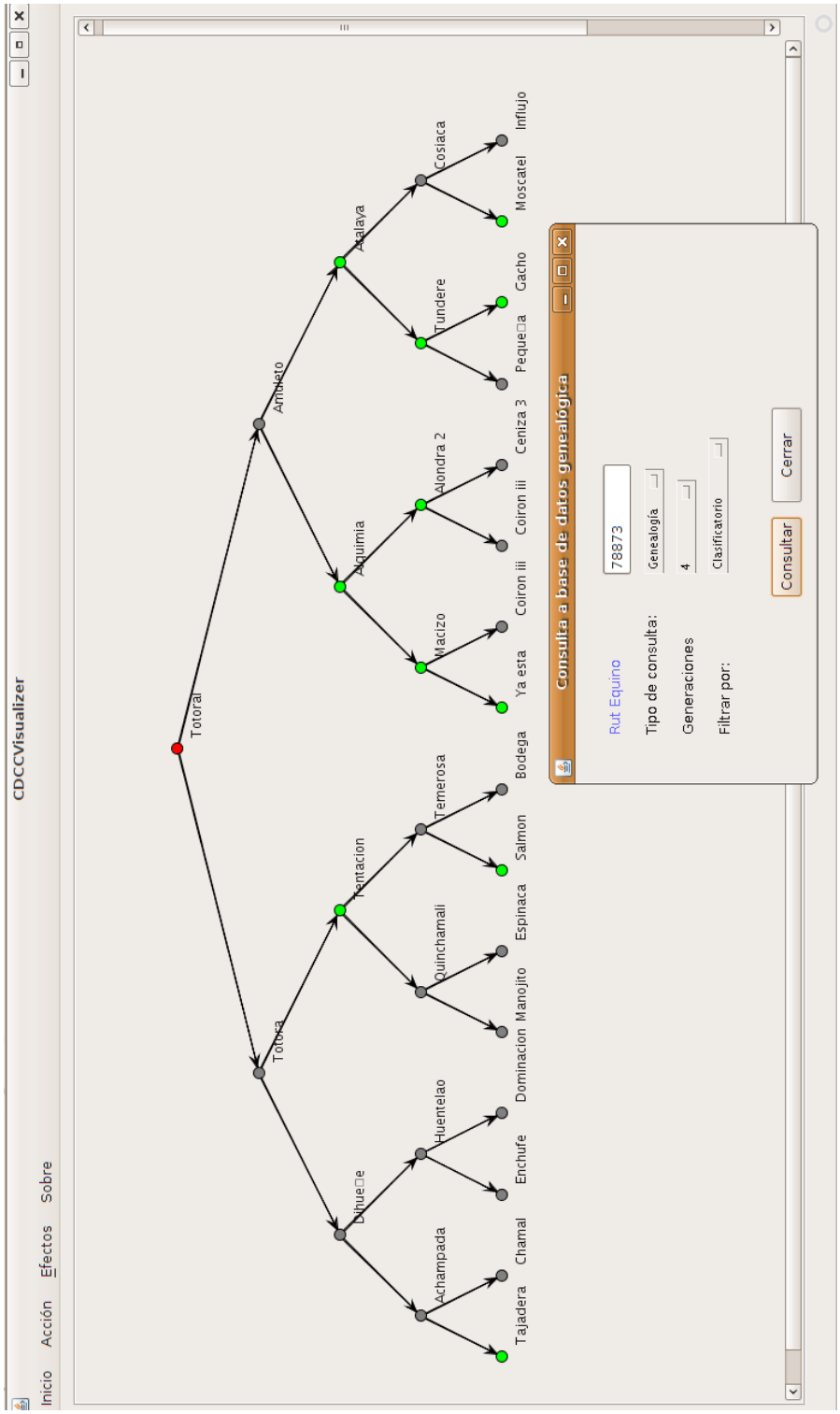


(e) Descendencia para ejemplar , Maquinita $run = 8859$, $k = 6$.

4.2.4. Consulta de genealogía

Para la consulta de genealogía se obtuvieron resultados satisfactorios, ya que se esperaba obtener árboles binarios balanceados y bien formados, como se ve en la figura (4.1(f)). Lo cual fue posible gracias a que se construyeron los grafos generados para que así sea. La razón principal fue que la poligamia de los equinos es patente, de hecho en varios de los resultados hay ejemplares que se representan como nodos distintos pero que son el mismo ejemplar, generalmente machos reproductores. Aplicada esta salvedad se pudo obtener grafos que para el *layouter* son fáciles de tratar, por ejemplo para una consulta con grado $k = 4$, el máximo número de nodos esperados es $2^k - 1$ y los arcos son del mismo orden, con lo cual se pueden procesar y desplegar hasta $k = 7$ generaciones sin mayores problemas.

Otra característica interesante de esta visualización es que se puede ver gráficamente como ejemplares machos y hembras tienen mezclas cosanguíneas dentro de la misma familia. Por ejemplo un padre tiene una cría hembra con la que después engendró otra cría o también cruces entre ejemplares *hermanos*.



(f) Genealogía ejemplar, Totoral $run = 8873$.

Capítulo 5

Conclusiones

Concluyendo con la arquitectura de una aplicación que podemos catalogar de tipo *computación distribuida* se tienen dos componentes bien definidas: el servidor y la aplicación cliente. La aplicación en el servidor maneja la información genealógica es capaz de responder a las consultas eficientemente y de forma robusta: hace lo pedido y entrega resultados razonables en tiempos relativamente pequeños, respetando su naturaleza de servicio a través de Internet. El diseño e implementación de esta parte hace soportante la infraestructura para extensiones posteriores y a su vez prueba que es una buena aproximación para el problema de entrega de información.

La segunda componente, la aplicación cliente, es capaz de generar y procesar las consultas emitidas al servidor de forma robusta y rápida pero hay costos asociadas a ella por la tecnología empleada. No obstante, para interoperabilidad y mantención de la misma se puede justificar su uso.

Debido a la alta complejidad computacional del problema de trazado de grafos, generalmente problemas que son *NP-Duros*, el trazado de grafos de gran tamaño y con gran cantidad de nodos y de arcos es costoso. Por esta razón el trazado sólo se empleó grafos *pequeños* para este problema particular. En el tratamiento de grafos por medio esta aplicación se consideró que debían que ser manejados por personas, y en consecuencia éstos tenían que ser de tamaño reducido (de nodos y arcos), para enriquecer y dar calidad a la información requerida por el usuario. Otro punto considerado es el de la usabilidad por personas no expertas en el tratamiento de redes o grafos, que sólo lo pueden ver como una representación de la información de su interés. Lo anterior condujo a diseñar una interfaz sencilla y con funcionalidades específicas para el tipo de problema.

Dadas algunas particularidades propias de la información genealógica equina, por ejemplo que la monogamia, casi no existe, hacía que ciertos supuestos tomados a priori antes de examinar los resultados, por ejemplo se esperaba obtener árboles genealógicos completos tal como en el caso de los humanos, no se cumplieron. También estas hipótesis influyeron en el desempeño de los algoritmos de trazados de los grafos, influyendo en las restricciones estéticas impuestas para los tipos de visualización definidos en los trazados de los grafos. Por último

se puede rescatar que alguna de esta información muestra algunas características propias de la genealogía equina como son la presencia patente en la descendencia de un reproductor en una misma familia, *endogamia*, por ejemplo en las consultas de descendencia y genealogía, que no son patentes hasta su visualización, y que agregan valor al uso de la aplicación.

El mayor reto que puede dejar este trabajo es el tratamiento de grandes grafos, con cientos de nodos o inclusive llegar a los miles. Se observó que es un desafío en cuanto al tiempo de los algoritmos usados en el visualizador y así como también para la visualización del mismo. Para lo primero hay algoritmos propuestos que poseen escalabilidad para el trazado de grandes grafos, pero pierden calidad en la visualización. Quizás el trazado de grafos en tres dimensiones pueda ayudar. La introducción de otra dimensión, al contrario de otros problemas en ciencias de la computación, podría *relajar* alguna de las restricciones impuestas en dos dimensiones.

Como trabajo futuro para la extensión de este trabajo u otros con fines específicos basados en la misma problemática se podrían introducir, a juicio del autor, las siguientes:

Soporte para consultas dinámicas : En una consulta ya generada en el servidor tener la posibilidad de agregar dinámicamente un subgrafo al grafo de dicha consulta. Por ejemplo si se quiere tener la consulta genealógica de un ejemplar hasta el grado k, y después extender dicha consulta pero para un grado $k + n$, poder generar sólo la información asociada al subgrafo de grafo n que uniendose al gafro de la consulta original retorna el resultado deseado. Y en términos de usabilidad el usuario no debiese reprocesar en su *mapa mental* ([23], cap. 9), una nueva visualización que sólo es una extensión de una más pequeña.

Reconocimiento de posibles nodos problemáticos : Posible reconomiento de nodos que tengan alta incidencia de arcos en el grafo para poder colapsarlos a priori. Pero resaltando que hay más información para ese nodo en particular. Todo esto para ayudar a la legibilidad y rápida exploración de la visualización de la información por parte del usuario.

Posibilidad de definir atributos extras en los ejemplares : Permitir que el usuario pueda introducir nuevos atributos a un ejemplar o ejemplares de su interés y que éstos puedan ser usados por otros para enriquecer el uso de la información del registro genealógico. Si bien la Federación de Criadores del Caballo Chileno ha hecho un esfuerzo por concentrar y parametrizar información completa de una historia con más de cien años, usuarios expertos podrían enriquecerla aún más con datos específicos que pueden ayudar a un mejor uso de la misma.

Uno de los objetivos implícitos de esta memoria es enriquecer el uso de la información genealógica para usuarios expertos, aficionados en el tema, para criadores y seguidores del rodeo chileno. Se espera que con este trabajo se abran nuevas puertas para la evolución y uso de tecnologías de la información en este tipo de actividades como ésta.

Bibliografía

[1] M. Scott Marshall Ivan Herman, Guy Melancon. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, pages 24–43, 2000.

[2] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[3] Helen C. Purchase, Robert F. Cohen, and Murray James. Validating graph drawing aesthetics. In *GD '95: Proceedings of the Symposium on Graph Drawing*, pages 435–446, London, UK, 1996. Springer-Verlag.

[4] Peter Eades and Xuemin Lin. Spring algorithms and symmetry. *Theoretical Computer Science*, 240(2):379–405, 2000.

[5] Corey Kosak, Joe Marks, and Stuart Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Trans. Systems, Man and Cybernetics*, 24(3):440–454, 1994.

[6] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, 18(1):61–79, 1988.

[7] Kenneth J. Supowit and Edward M. Reingold. The complexity of drawing trees nicely. *Acta Inf.*, 18:377–392, 1982.

[8] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Trans. Softw. Eng.*, 7(2):223–228, 1981.

[9] P. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 14(4):10–36, 1992.

[10] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *j-IEEE-TRANS-SYST-MAN-CYBERN*, SMC-11(2):109–125, feb 1981.

[11] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. *Doctoral dissertation*, 2000.

[12] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. Marshall. Graphml progress report: Structural layer proposal, 2002.

[13] Db4o : base de datos orientada a objetos - http://www.db4o.com.

[14] Jim Paterson, Stefan Edlich, Henrik Hörning, and Reidar Hörning. *The Definitive Guide to db4o*. Apress, Berkely, CA, USA, 2006.

[15] Postgresql : Relational database opensource system - http://www.postgresql.org/.

[16] Jax-rs: 311 the java api for restful web services - http://jcp.org/en/jsr/detail?id=311.

[17] Restlet : Lightweight rest framework for java - http://www.restlet.org/.

[18] Dbcp : Apache commons pool system - http://commons.apache.org/dbcp/.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse in object-oriented designs. In O.Ñierstrasz, editor, *Proceedings of ECOOP'93*, Berlin, 1993. Springer-Verlag.

[20] Wouter deÑooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory Social Network Analysis with Pajek (Structural Analysis in the Social Sciences)*. Cambridge University Press, January 2005.

[21] Jung: Java universal network/graph framework - http://jung.sf.net.

[22] Christopher Homan, Andrew Pavlo, and Jonathan Schull. Smoother transitions between breadth-first-spanning-tree-based drawings. In *Graph Drawing*, pages 442–445, 2006.

[23] Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999)*, volume 2025 of *Lecture Notes in Computer Science*. Springer, 2001.