Sistemas Informáticos
Curso 2009 – 2010

Proyecto Krowdix 2.0:

Sistema de simulación y análisis de redes sociales

Fernando Burillo López
Alberto Jiménez Ramiro
Ignacio Rico Teixeira

Dirigido por:

Diego Blanco Moreno

Facultad de Informática
Universidad Complutense de Madrid

Krowdix 2.0

Proyecto de Sistemas Informáticos
Facultad de Informática

Universidad Complutense de Madrid

Autores:

Fernando Burillo López
Alberto Jiménez Ramiro
Ignacio Rico Teixeira

Profesor director:

Diego Blanco Moreno

Curso 2009 / 2010

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

En Madrid, a 28 de junio de 2010

Fernando Burillo López Alberto Jiménez Ramiro Ignacio Rico Teixeira

Resumen

Krowdix 2.0 es una herramienta de simulación de redes sociales basada en agentes inteligentes para visualizar y analizar la evolución de redes sociales en tiempo real. La aplicación es capaz de crear distintas redes sociales de hasta 10.000 usuarios y simular el comportamiento de cada uno de los usuarios en la red. Una vez creadas las redes se puede visualizar cada uno de los instantes de evolución mediante las distintas vistas disponibles en el sistema. Es posible configurar los distintos perfiles de los usuarios de la red social, logrando distintas evoluciones de las redes según haya más usuarios de un perfil o de otro. Estos perfiles son editables, pudiendo crear nuevos perfil según nuestras necesidades.

Abstract

Krowdix 2.0 is an undergoing social network simulator tool based on intelligent agents to visualize and analyze social network evolution in real time. This software is able to create several social networks up to 10.000 users, and simulate each one of the users behaviour inside the network. Once the networks are built, each moment of their evolution can be visualized by the different views supplied by Krowdix. It is possible to set the different users profiles, achieving varied evolutions in the social network depending whether there are more profiles of a kind or another. Those profiles are editable so it is posible to make new profiles according to our needs.

INDICE DE CONTENIDOS

1 INTRODUCCIÓN

1.1 SOBRE KROWDIX 2.0

Krowdix 2.0 es una evolución del software de simulación y análisis de redes sociales del mismo nombre desarrollado en la asignatura de Sistemas Informáticos el curso 2008/2009. Manteniendo la idea original hemos desarrollado un sistema de simulación de redes sociales con un diseño de la arquitectura y un motor de ejecución totalmente nuevo y diferente.

El principal objetivo del nuevo diseño ha sido llegar a simular redes sociales con un número de usuarios de en torno a 10000. Además se ha tratado de dotar a cada usuario de la red social, a partir de ahora URS, de un comportamiento acorde con el entorno que le rodea, es decir, la red y evitar una simulación que se limite a hacer crecer la red aleatoriamente de forma que, al final, obtenemos una red en la que todos están relacionados con todos.

Una red social se compone de los usuarios que la forman y de las relaciones entre ellos. Podemos llegar a la conclusión de que estos dos elementos son la estructura que va a sostener una red social, ya que ésta no podría existir sin usuarios así como sin una serie de interconexiones entre ellos que produzca un intercambio de información. A esta información le podemos llamar “contenidos”, en una red social los usuarios están continuamente mostrando a sus contactos una serie de contenidos, ya sean mensajes, fotos, comentarios en contenidos de otros usuarios. Además las relaciones entre usuarios van a hacer formase grupos en la red social. Estos grupos suelen ser más o menos homogéneos en cuando a las aficiones de los usuarios. Además las relaciones no tienen el mismo grado de importancia según los usuarios que están relacionados. Una relación será más fuerte conforme se intercambien más contenidos los usuarios que une esa relación.

De esta forma vamos a modelar en nuestro sistema redes sociales en los elementos principales serán:

- Usuarios de la red social (URS)
- Relaciones entre ellos (Amistades)
- Contenidos de la red social (CRS)
- Grupos de la red social

Además cada URS va a tener asociada una afición que va a ser un factor importante a la hora de entablar nuevas relaciones con los demás URS. Y cada relación tendrá una propiedad llamada afinidad. Esta propiedad marcará la importancia de la relación en la red social. La afinidad estará marcada por el intercambio de contenidos entre los URS, cuanto más intercambio más crecerá la afinidad pero si ese intercambio decrece la afinidad se verá afectada negativamente pudiendo llegar un momento en que la relación desaparezca si la afinidad es suficientemente baja.

Acabamos de describir los elementos que forman el modelo de nuestra red social.

Ahora vamos a hablar de los componentes que harán evolucionar nuestra red. Cuando decimos evolucionar nos referimos a tres cosas, hacer crecer la red con nuevos usuarios, entablar nuevas relaciones entre usuarios o modificar la afinidad entre ellas, o eliminar relaciones o usuarios de la red social. Estos componen van a ser dos, los perfiles y las acciones.

Los perfiles están asociados a los usuarios de la red. Van a estar predeterminados antes de empezar a hacer andar la simulación de nuestra red social. El perfil de cada usuario va a guiar el comportamiento de éste en cada instante en que le toque llevar a cabo una acción. Por ejemplo, si un usuario tiene un perfil lector, se va a dedicar la mayor parte del tiempo a leer contenidos de otros usuarios, mientras que si tiene un perfil amistoso, dedicará la mayor parte del tiempo a entablar nuevas relaciones (amistades) con los demás usuarios de la red.

Las acciones van a ser lo que verdaderamente moverá nuestra red social. Vamos a tener un número predeterminado de acciones que se encargarán de hacer crecer la red, entablar nuevas relaciones entre usuarios, crear nuevos contenidos, crear nuevos grupos, o añadir usuarios a algún grupo. Se ha intentado plasmar el comportamiento en las redes sociales reales en estas acciones. Pensando en la posibilidades que tiene un usuario real de una red social hemos creados todas las acciones para que la simulación de una red social en el sistema Krowdix 2.0 sea realista.

Los perfiles y las acciones están directamente relacionados de forma que cada perfil establece una ponderación de cada acción. Según esto las acciones que tienen mayor peso en el perfil se ejecutarán más veces que las demás.

Hasta aquí tendríamos el modelo de datos de nuestra red social y el motor de ejecución. Necesitamos un controlador que nos permita configurar los parámetros de simulación de una red social. En la versión a la que hemos sido capaces de llegar debemos configurar todos estos parámetros antes de echar a andar la red. Cuando creamos una red social nueva, hay que configurar el número de usuarios iniciales, y establecer los perfiles que se van a usar en la simulación, así como el porcentaje de usuarios que va tener cada perfil en la nueva red. De esta forma podemos modelar por ejemplo, una red con usuarios con perfiles lectores y amistosos y que el 50% de los usuarios tenga un perfil lector y los demás un perfil amistados. Lamentablemente en esta versión de Krowdix no se pueden cambiar estos parámetros una vez lanzada la simulación.

Nuestro sistema nos permite parar la simulación y tiene una serie de vistas de los datos generados en esta simulación. Las diferentes vistas van mostrando los cambios realizados en la red para cada instante de tiempo, pudiendo diferenciar fácilmente la evolución de la red en una línea temporal.

Las vistas nos sirven para mostrar una serie de estadísticas y medidas utilizadas en el análisis de redes sociales. También hemos realizado gráficas con datos concretos de nuestras redes como son las aficiones de los usuarios o una representación de los agrupamientos llevados a cabo en la red.

1.2 SOBRE ESTE DOCUMENTO

En este documento vamos a ilustrar tanto el diseño de la aplicación como el manejo de la misma por parte del usuario final. El propósito es facilitar la comprensión del trabajo realizado por nuestra parte en este proyecto a toda aquella persona que esté interesada en conocerlo o ampliarlo.

Hemos querido dejar claros los requisitos del sistema con los que hemos trabajado, de forma que sean fácilmente comprobable si se ha cumplido su consecución. También se trata de mostrar el diseño de la implementación del sistema de tal forma que se conozcan los paquetes y las clases que hemos realizado así como los métodos más importantes a los que se deben llamar para la utilización de los módulos implementados para poder hacer una ampliación de funcionalidades del sistema, o mejorar su rendimiento.

Todo lo mostrado en este documento se puede obtener en la plataforma *SourceForge* en la página http://sourceforge.net/projects/krowdix/ de tal forma que quien este interesado en él puede bajar el código fuente completo. Si se desea contribuir al desarrollo de Krowdix 2.0 se puede unir al equipo de desarrolladores, permitiendo el acceso al repositorio SVN. Se puede utilizar el sistema, o ampliarlo ya que está disponible bajo licencia GPL.

2 ESTADO ACTUAL

El análisis de redes sociales, Social Network Analisys (SNA) se remonta al menos en la década de 1940 (posiblemente a fines del siglo 19), y el análisis de redes sociales (ARS) como un campo formal ha sido bien establecido en la sociología. El análisis de redes sociales prevé descriptores formales, y por medio de medidas cuantitativas permite realizar pruebas de modelos estadísticos sobre las relaciones y estructuras.

El análisis de redes sociales se originó como un medio para la aplicación de la computación científica para el análisis sociológico cuantitativo, y sigue siendo de gran importancia allí. Se ve un uso intensivo en los estudios de modelos de negocio y estrategias, pero también se aplica intensamente a asuntos militares, y los estudios criminológicos antiterroristas. También ve el servicio a través de una notable variedad de campos, demasiado numerosa para resumir, incluso, incluyendo las ciudades de origen humano / interacción de los animales, como la agricultura y, junto con otros métodos científicos basados en la informática, tales como el modelado basado en agentes y el análisis fractal, el bienestar animal.

Actualmente existen una serie de programas de análisis de redes social tanto de tipo comercial, freeware, y de código libre o abierto. En general el software SNA posee una serie de características comunes.

1. Introducción de datos y manipulación de datos.
2. Técnicas de visualización
3. Rutinas de análisis de redes sociales, divididos en tres tipos de métodos:
    a. Métodos descriptivos para calcular (simple) estadísticas de la red (por ejemplo, la centralidad.
    b. Análisis basados en procedimientos basados en más algoritmos complejos (iterativos) por ejemplo, análisis de clusters
    c. Modelos estadísticos basados en las distribuciones de probabilidad.

Todos estos programas ofrecen una serie de métricas sobre las redes. Estas métricas se extraen de una serie fórmulas y metodologías que otorga el análisis de redes sociales. Usando software de análisis de redes se pueden obtener respuesta a preguntas como:

- ¿En qué medida está conectada una entidad a una red?
- ¿Cuál es la importancia real de una entidad en una red?
- ¿Cómo de central es una entidad en una red?
- ¿Cómo fluye la información dentro de una red?

Los programas de análisis de redes sociales, tratan una red como un conjunto de nodos y un conjunto de aristas que relacionan estos nodos. Además usan funciones de la teoría de grafos para extraer la información antes descrita de la red. Además de todas estas métricas usan diferentes vistas gráficas de la red social, representada como un grafo y utilizando distintas maneras de disponer todos estos nodos y sus aristas, como por ejemplo circularmente, jerárquicamente, etc.

Krowdix 2.0 como software de análisis de red sociales debe satisfacer todas estas características pero además incluye la potencia del modelado y simulación de las redes

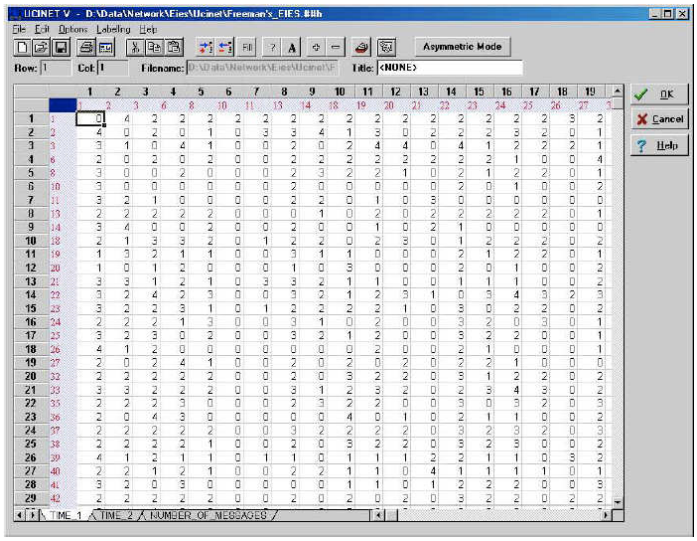
sociales. En nuestro sistema podemos modelar la “forma” de una red social mediante la asignación de perfiles a cada entidad y hacerla evolucionar mediante la simulación para posteriormente pasar al análisis de la red creada.

Vamos a ver tres programas actuales de análisis de red sociales, UCInet, Pajek y SocNetV.

2.1 UCInet

UCINET 6.0 (versión 6.05; Borgatti, Everett y Freeman, 2002) es un amplio programa para el análisis de redes sociales y otros datos de proximidad. Es probablemente el más conocido y más utilizado software para el análisis de sociales de redes de datos y contiene un gran número de rutinas de análisis de red. El programa es un producto comercial, existe una versión de evaluación gratuita, que puede ser ejecutada durante 30 días sin necesidad de registrarse. El manual consta de dos partes: una guía del usuario (gestión de datos y la manipulación) y una guía de referencia (el análisis de redes).

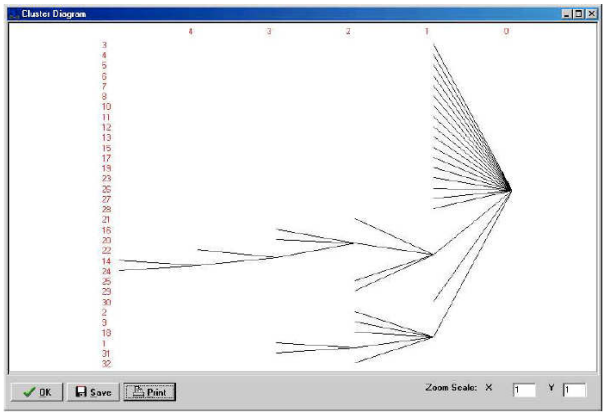
UCINET es un programa de Windows basado en menús, y, como los desarrolladores mismos dicen, “se construye para la velocidad, no para la comodidad” (Borgatti, Everett y Freeman, 1999). La elección de los procedimientos de los menús suele dar lugar a la apertura de un parámetro de formulario donde la entrada para los algoritmos se especifica. Se generan dos tipos de salida se generan: textual de salida, salvo en el diario y les muestra en la pantalla (véase la figura para un ejemplo), y conjuntos de datos que se pueden utilizar como entrada para otros procedimientos. Puede manejar del orden de 30.000 nodos.



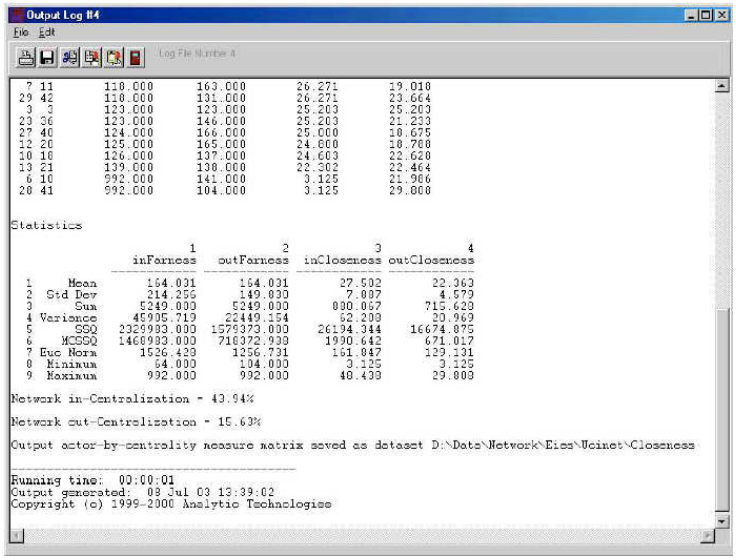
UCINET provee un gran número de herramientas de gestión de datos y la transformación como subconjuntos de la selección, la fusión de los conjuntos de datos, permutando, transposición, o recodificación de datos.
Tiene un lenguaje de álgebra de matrices con todas las características, puede manejar dos modalidades de datos, así como obtener los datos de un modo establece a partir de datos de dos modos. Hay una opción para introducir los datos de atributos y para especificar los valores que faltan. Cabe señalar, sin embargo, que sólo unos pocos

procedimientos pueden manejar adecuadamente los valores que faltan. UCINET se distribuye con un gran número de datos de ejemplo conjuntos, incluidos los datos de Freeman EIES.

UCINET contiene herramientas gráficas para elaborar diagramas de dispersión y diagramas de árbol, que se pueden guardar como mapa de bits les (BMP). El programa en sí no contiene los procedimientos gráficos para visualizar las redes, pero tiene un speedbutton para ejecutar el programa NetDraw (Borgatti, 2002), que lee archivos UCINET de forma nativa. NetDraw, es un software desarrollado para la visualización de redes. Además de exportar funciones y Pajek Mago, los datos pueden ser exportados para su visualización en KrackPlot



El programa contiene una gran cantidad de rutinas de análisis de red para la detección de subgrupos cohesivos (pandillas, clanes, plexos) y regiones (componentes, núcleos), por análisis de centralidad, para el análisis del ego de la red, y para el análisis de los agujeros estructurales. Como un ejemplo, el resultado de un análisis de centralidad se presenta en la siguiente figura. Para cada nodo, contiene la lejanía de entrada y salida (la suma de las longitudes de las geodésicas de y de todos los demás nodos), y la centralidad de éstos.



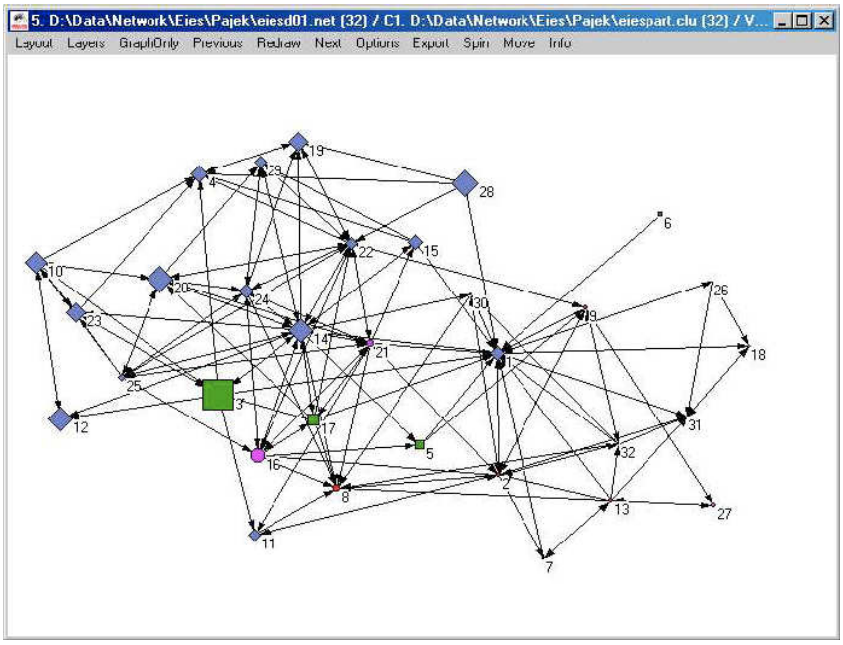
2.2 Pajek

Pajek es un programa de análisis y visualización de redes, específicamente diseñado para manejar grandes conjuntos de datos. Los principales objetivos del diseño de Pajek son:

1. Facilitar la reducción de una red grande en varias redes más pequeñas que se pueden tratar más con métodos más sofisticados.
2. Proporcionar al usuario herramientas de visualización de gran alcance.
3. Aplicar una selección eficiente de algoritmos de redes.

El programa puede descargarse de forma gratuita, y sus desarrolladores están continuamente actualizándolo. No hay ayuda en línea y la documentación disponible no es suficientemente detallada para los usuarios que no son expertos en el análisis de redes.

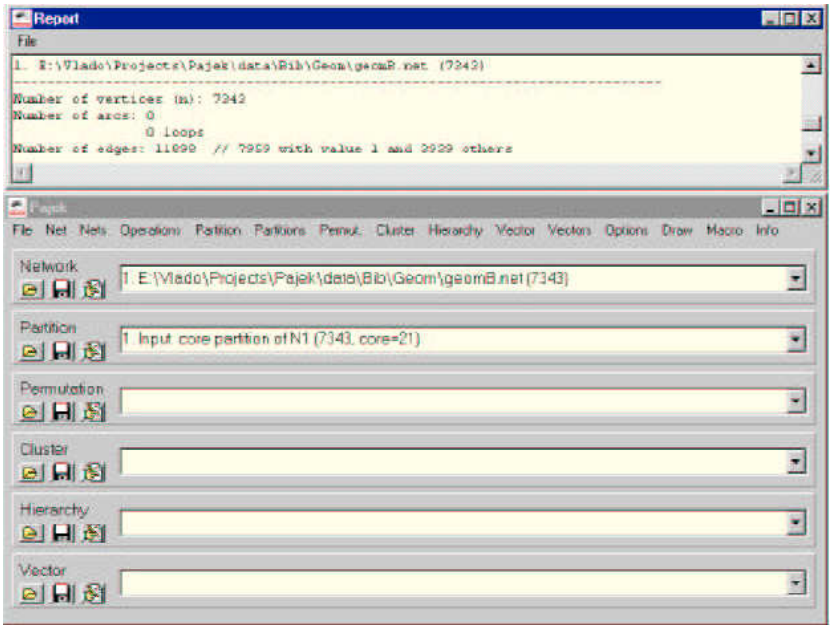
Pajek puede manejar múltiples redes simultáneamente y redes con eventos en el tiempo. Los eventos en el tiempo resumen el desarrollo o evolución de las redes a través del tiempo (usando unas diferencias horarias como indicadores). En Pajek se pueden analizar redes muy grandes, con más de un millón de nodos, la disposición de memoria en el equipo establece el límite real. Para ahorrar memoria, los nombres y etiquetas de los nodos no se mantienen las redes muy grandes, pero estos se pueden adjuntar más adelante a las pequeñas subredes.



Además de sus propios formatos de entrada, Pajek soporta varios formatos: UCINET DL, genealógicas GED, y algunos formatos molecular: BS (Ball y Stick), Mac (Mac Molecule) y MOL (MDL archivo MOL).
Al ejecutar Pajek obtendrá la ventana principal, Pajek es organizado como una "calculadora" para los datos de la red:

1. Network: objeto principal (vértices y líneas);
2. Partition: a la que pertenece un vértice de racimo;
3. Vector: valores de vértices;
4. Permutation: reordenación de los vértices;
5. Cluster: subconjunto de vértices (por ejemplo, un clúster de la partición);

6. Hierarchy: agrupaciones jerárquicamente ordenadas y vértices.



Pajek contiene opciones de manipulación para todas sus estructuras de datos. Por ejemplo, las redes pueden ser traspuestas, grafos dirigidos cambiados a grafos no dirigidos y viceversa al contrario, las aristas pueden ser añadidas o eliminadas, o la red puede ser reducida por la disminución de clases o extracción de las piezas. El programa también contiene las operaciones básicas de red como recodificación o dicotomización. Por otra parte, las transformaciones amplias para los atributos y las opciones para crear otros datos objetos sobre la base de los atributos (jerarquías, en racimos).

Las propiedades gráficas de Pajek son avanzadas. La ventana de dibujar da al usuario muchas opciones para manipular los gráficos (diseño, tamaño, color, rotación, etc.) Por otra parte, tiene representaciones gráficas de las particiones, los vectores, y combinaciones de particiones y vectores. El dibujo de la red se basa en el principio de que las distancias entre los nodos deben revelar el patrón estructural de la red. Además de diseños sencillos (círculo, al azar), Pajek tiene varios procedimientos automáticos para diseños óptimos. Tiene un algoritmo que supone que los nodos están conectados por medio de muelles, cuya tensión se quiere minimizar.

2.3 SocNetV

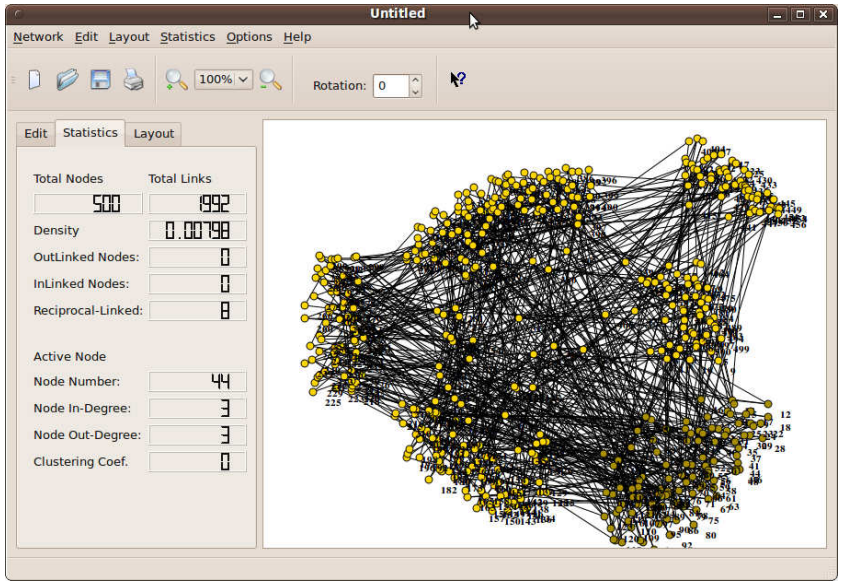
Social Networks Visualizer (SocNetV) es una herramienta multi-plataforma, para el análisis y la visualización de las redes sociales. SocNetV le permite construir redes sociales con unos pocos clicks en un lienzo virtual o cargar redes en diferentes formatos (GraphML, GraphViz, Adyacencia, Pajek, UCINET, etc.) y modificarlos para adaptarlos a sus necesidades.

La aplicación puede calcular todas las propiedades de red de base, tales como el diámetro de grafos, y las distancias (la más corta longitud de ruta de acceso), así como las estadísticas estructurales más avanzados, tales como centralidades los

nodos y de la red (es decir, cercanía, betweeness, grafos, etc.), coeficiente de agrupamiento , etc.

Dispone de varios algoritmos de diseño para las visualizaciones de sus redes. Además, se pueden crear con unos pocos clicks unas redes aleatorias (Erdos-Renyi, Watts-Strogatz, la red del anillo, etc.).

SocNetV es un trabajo en curso y está siendo desarrollado en C++ y Qt, un conjunto de herramientas de código abierto de desarrollo GUI de Nokia. Nuestro objetivo principal es la plataforma Linux, pero se puede compilar y ejecutar SocNetV en OS X y Windows, siempre y cuando cuenten con las bibliotecas Qt4 instalado.



3 ESPECIFICACIÓN DE REQUISITOS

3.1 INTRODUCCIÓN

Krowdix pretende ser una herramienta innovadora para el análisis del comportamiento y la evolución de una red social. Existen diversas herramientas que tratan parcialmente este aspecto, pero no son integrables entre sí, y la elección de las bondades de una implica el rechazo de ventajas del resto que la elegida no trae. Por ello, cuando se diseña Krowdix se intenta pensar en todas las buenas características de las herramientas existentes, además de nuevos aspectos no encontrados en ellas.

El objetivo es conseguir una herramienta flexible para poder reproducir todas las condiciones tanto fijas como variables que proporciona cualquier red social de hoy en día. Además, se pretende dar un aspecto totalmente amigable e intuitivo para facilitar el uso de la aplicación y la comprensión de los datos obtenidos.

En la especificación de requisitos se plasma el comportamiento del sistema Krowdix 2.0, no se hará alusión es este documento a requisitos más bien subjetivos, de distinta interpretación por parte del usuario, como por ejemplo, el uso de un interfaz intuitiva ya que eso puede muy diferente por parte de cada persona.

El propósito es poder definir una serie de metas que nuestro sistema debe cumplir y una vez finalizada la implementación poder comprobar fácilmente si se han llegado a cumplir. Así como facilitar las etapas de la fase de implementación de tal forma que sirva de guía para completar todas las funcionalidades y requisitos de diseño del sistema. Además especificar concretamente todos los aspectos desarrollados en el sistema de tal forma que sea fácilmente comprobable su realización

3.2 REQUISITOS

3.2.1 Interfaz

Código Requisito	Prioridad	Realización	Descripción
REQ001	Alta		Krowdix permitirá la definición de distintos tipos de usuarios desde la interfaz
REQ002	Alta		Se podrán asignar distintas políticas de creación de usuarios en la red desde la interfaz
REQ003	Alta		Se podrán incluir nuevos usuarios con perfiles concretos desde la interfaz
REQ004	Alta		La interfaz se representará como distintas paletas flotantes que nos permitirán monitorizar el estado de la red: Influencias por amistades. Influencias por aficiones.
REQ005	Alta		El sistema permitirá definir reglas concretas de los parámetros a monitorizar en el sistema:

			– Valores independientes de los nodos – Expresiones regulares sobre ciertos valores de los nodos – Características de subredes dentro de la red
REQ006	Media		Utilizar JavaFX para el desarrollo de la interfaz gráfica
REQ007	Baja		La gráfica 3D agrupará nodos que tengan una serie de atributos comunes de manera automática, permitiendo la navegación en modo drill-down hacia los nodos inferiores

3.2.2 Modelo de Red Social

Código Requisito	Prioridad	Realización	Descripción
REQ008	Alta	Representación	Una red social es un compendio de: • Usuarios de la red social • Vínculos establecidos entre usuarios • Información asociada a los usuarios • Dominios de usuarios (Grupos o cualquier otra agrupación)
REQ009	Alta	Representación	La red social lleva asociado un “indicadores de estado de red”, un conjunto de valores absolutos que nos permite identificar y monitorizar cómo evoluciona la red a lo largo del tiempo.

3.2.3 Modelo de Usuario Red Social

Código Requisito	Prioridad	Realización	Descripción
REQ010	Alta	Representación	Un URS tendrá asociado un perfil.
REQ011	Alta	Representación	Un perfil establece cual es el comportamiento del URS dentro de la red social, qué acciones puede suele llevar a cabo.
REQ012	Alta		Una acción describe qué puede hacer un usuario dentro de la red social. Ejemplos: hacer un nuevo amigo, enviar un mensaje, buscar información o aceptar la solicitud de amistad de un tercero.
REQ013	Alta		Cada acción tiene un coste de ejecución asociado. Para cada unidad de tiempo de ejecución de la RS, instante, se podrán ejecutar acciones mientras el URS no haya agotado sus unidades de ejecución por instante. Si con las unidades de ejecución pendientes no se puede ejecutar una acción, se decrementa al coste de ejecución las unidades de ejecución disponibles.
REQ014			Acciones de crecimiento de la RS Acciones de evolución de la RS Acciones de decrecimiento de la RS

REQ015	Media		Un URS tendrá asociado un Estado de Red Social (EURS). El estado tendrá en cuenta los siguientes elementos: • Dominios creados • Dominios en los que participo • Amigos • Posibles amigos (Amigos de amigos) • Contenidos propios

3.2.4 Afinidad de URSs y grupos

Código Requisito	Prioridad	Realización	Descripción
REQ016	Alta	Representación	Entre cualesquiera dos URSs asociados a nivel de red social existirá una relación de afinidad entre ellos bidireccional.
REQ017	Alta		Entre cualquier usuario y los grupos a los que pertenece existirá una relación de afinidad
REQ018	Alta		La relación de afinidad tendrá un valor entero.
REQ019	Alta		El valor de la afinidad puede ser diferente de un usuario hacia el otro.
REQ020	Alta		El valor de afinidad se incrementa cuando un URS realiza una acción hacia otro URS o grupo. El valor a incrementar dependerá de la acción a desarrollar.
REQ021	Alta		El sistema monitorizará la última vez que se modificó la afinidad entre cualesquiera dos elementos del sistema. Cuando hayan transcurrido más de un límite máximo de instantes sin actividad, el sistema decrementará el valor de la afinidad en un valor constante. Este valor indica la degradación de las relaciones en la red.

3.2.5 Acción de red social

Código Requisito	Prioridad	Realización	Descripción
REQ022	Media		Cada acción de red social tiene un valor de afinidad que indica cuanto se incrementa la afinidad entre el origen y el destino de la acción.
REQ023	Alta	Representación	**Evaluación de ejecución de acción:** **Precondición:** El URS dispone de algún punto de acción disponible. **Acción:** El gestor de RS verifica que el URS tiene puntos de acción suficientes para ejecutar la acción Se decrementan los puntos de acción disponibles del URS necesarios para ejecutar la acción Se ejecuta la acción que invoca el caso de uso. **Poscondición:** El URS tiene actualizados sus puntos de acción

			disponibles. **Estado URS:** La acción se ha ejecutado o si la acción no se ejecuta, se actualizan los puntos de acción necesarios para ejecutarla. **Estado de Red Social: No aplica**
REQ024	Alta	Representación	**Enviar un mensaje a un conocido:** **Precondición:** El URS Origen dispone de amigos a los que enviar mensajes. **Acción:** Seleccionar a un amigo de la lista de amigos existentes. Si esta acción se desencadena a raíz de un mensaje en mi lista de entrada, el destinatario es el emisor. Crear un nuevo contenido Asignar el contenido enviado al emisor Asignar el contenido enviado al receptor **Poscondición:** El URS Destino recibe el mensaje enviado. **Estado URS:** El valor del enlace entre URS emisor y receptor se incrementa (está activo) **Estado de Red Social:** Existe un nuevo contenido que es el mensaje enviado.
REQ025	Alta	Datos	**Crear tipo de grupo "grupo temático":** **Precondición:** Ser URS de la RS **Acción:** El URS crea un grupo temático, asociado con un determinado tema, al que otros URS podrán solicitar participación y enviar mensajes que serán leídos por todos los participantes. Los URS que piden participación tienen entre sus aficiones el tema del grupo. Solo se puede enviar mensajes si el URS figura como participante. **Poscondición:** Hay una nueva instancia de grupo temático en el sistema, asociado al URS que ejecuta la acción, con una relación de afinidad. **Estado URS:** Tiene asociado el grupo temático nuevo como dueño. **Estado RS:** Existe un grupo nuevo en la red social.
REQ026	Alta	Datos	**Crear tipo de grupo "Foro":** **Precondición:** Ser URS de la RS **Acción:** El URS crea un foro, al que otros URS pueden pedir participación. El foro no está asociado a ningún tema concreto, por lo que los URS que piden participación pueden tener diversas aficiones. Solo los participantes pueden enviar Posts al foro y leerlos. **Poscondición:** Hay una nueva instancia de grupo foro en el sistema, asociado al URS que ejecuta la acción, con una relación de afinidad.

			Estado URS: Tiene asociado un grupo foro nuevo como dueño. **Estado RS:** Existe un grupo nuevo en la red social.
REQ027	Alta	Representación	**Crear tipo de grupo "blog":** **Precondición:** Ser URS de la RS **Acción:** El URS crea un tipo de grupo blog. En el blog sólo puede escribir entradas él, y éstas son legibles por todos sus amigos. **Poscondición:** Se ha creado una nueva instancia de blog asociado al URS que ejecuta la acción, con su relación de afinidad del usuario al blog. **Estado URS:** El usuario tiene asociado un nuevo blog. **Estado RS:** Existe un blog nuevo en la red social.
REQ028	Alta	Representación	**Petición de amistad a un usuario:** **Precondición:** Ser URS de la RS **Acción:** El URS que ejecuta la acción busca a otro URS para hacerle una petición de amistad. **Poscondición:** El URS elegido tiene una nueva petición de amistad por parte del URS que ejecuta la acción. **Estado URS:** El estado del URS no varía. **Estado RS:** Existe una nueva petición de amistad pendiente de atenderse.
REQ029	Alta	Representación	**Procesar petición de amistad:** **Precondición:** El URS tiene peticiones de amistad pendientes de evaluar. **Acción:** Se coge una petición de amistad y se decide si se acepta o no. En caso de que se acepte, se crea la nueva relación de amistad. **Poscondición:** Hay una nueva relación entre los dos URS implicados con las afinidades al valor inicial. **Estado URS:** El usuario tiene una petición de amistad menos pendiente. **Estado RS:** En caso de aceptarse, la red tiene una relación de amistad más.
REQ030	Alta	Representación	**Enviar mensaje a grupo temático:** **Precondición:** El URS es creador o participante de algún grupo temático. **Acción:** El URS envía un mensaje nuevo a uno de los grupos temáticos en los que participa, o bien de los que es creador. **Poscondición:** Hay un nuevo mensaje en el grupo temático elegido, escrito por el URS que ejecuta la acción, y la afinidad entre el URS y el

			grupo ha aumentado. **Estado URS:** El estado del URS no se altera. **Estado RS:** Hay un nuevo mensaje en el grupo temático elegido.
REQ031	Alta	Representación	**Solicitar participación en grupo temático** **Precondición:** El URS no es creador ni participante del grupo en el que va a pedir participar. **Acción:** El URS envía una solicitud de participación a un determinado grupo temático del que no es creador. Sólo si se acepta podrá participar enviando mensajes. **Poscondición:** Hay una petición de participación del URS en el grupo elegido, que deberá ser procesada. **Estado URS:** El estado del URS no varía. **Estado RS:** Hay una nueva petición de participación en el grupo temático.
REQ032	Alta	Representación	**Enviar post a foro** **Precondición:** El URS es creador o participante de algún foro. **Acción:** El URS envía algún post a uno de los foros en los que participa, o ha sido creado por él. **Poscondición:** Hay un nuevo post en el foro elegido, escrito por el URS que ejecuta la acción, y la afinidad entre el URS y el foro ha aumentado. **Estado URS:** Tiene un nuevo contenido creado. **Estado RS:** Existe un nuevo post en el foro.
REQ033	Alta	Representación	**Solicitar participación en foro:** **Precondición:** el URS no es creador ni participante del foro en el que pide participar. **Acción:** El URS envía una solicitud de participación a un determinado foro, del que él no es creador. Sólo si se acepta podrá participar mandando posts. **Poscondición:** Hay una petición de participación del URS en el foro elegido, que deberá ser procesada. **Estado URS:** El URS ha hecho una nueva petición de participación en foro. **Estado RS:** Existe una nueva petición de participación en foro pendiente de atenderse.
REQ034	Alta	Representación	**Enviar entrada a blog:** **Precondición:** El URS es creador de al menos un blog. **Acción:** El URS envía una entrada a uno de sus blogs. Lo podrán leer sus amigos. **Poscondición:** Hay una nueva entrada en el blog elegido, escrito por el URS que ejecuta la

			acción, y la afinidad entre el URS y el blog aumenta. **Estado URS:** El URS ha creado un nuevo contenido. **Estado RS:** El blog tiene un nuevo contenido, creado por el URS.
REQ035	Alta	Representación	**Comentar un comentable:** **Precondición:** El URS debe tener acceso a un elemento comentable. **Acción:** El URS envía un comentario sobre un elemento comentable, que puede ser un amigo, o un grupo o contenido en los que participa o ha creado. **Poscondición:** Se ha creado un nuevo comentario, dirigido a un elemento comentable. **Estado URS:** El URS ha creado un nuevo contenido de tipo comentario. **Estado RS:** Existe un nuevo comentario.
REQ036	Alta	Representación	**Buscar a otro usuario de la red social:** **Precondición:** El URS que realiza la búsqueda debe pertenecer a la red social. La red social contiene un URS más aparte del que realiza la búsqueda. **Acción:** Un URS hace una búsqueda en la red social de otro URS que le interese, con el que quiere interactuar, ya sea para enviar mensajes, solicitar amistad, u otro tipo de acción descrito anteriormente que requiera de la búsqueda. **Estado URS:** El estado del URS no varía. **Estado RS:** El estado de la red social tampoco varía.
REQ037	Alta	Representación	**Buscar a otro URS de la red social que pertenezca a un mismo grupo:** **Precondición:** El URS que realiza la búsqueda debe pertenecer a la red social. Los grupos a los que pertenece el URS actual tienen más usuarios aparte de él mismo. **Acción:** Un URS hace una búsqueda en la red social de otro que pertenece a uno de los grupos del primero. **Estado URS:** El estado del URS no varía. **Estado RS:** El estado de la red social no varía.
REQ038	Alta	Representación	**Buscar a otro URS de la red social según una ontología** **Precondición:** El URS que realiza la búsqueda debe pertenecer a la red social. El URS debe tener un término o más en su ontología. **Acción:** Un URS hace una búsqueda en la red social de otro con el que comparte algún término de la ontología,

			Estado URS: El estado del URS no varía. **Estado RS:** El estado de la red social no varía.
REQ039	Alta	Representación	**Buscar a otro URS de la red social que sea amigo de amigo:** **Precondición:** El URS que realiza la búsqueda debe pertenecer a la red social, y debe tener al menos un amigo. **Acción:** Un URS hace una búsqueda en la red social de otro con el que comparte al menos un amigo. **Estado URS:** El estado del URS no varía. **Estado RS:** El estado de la red social no varía.

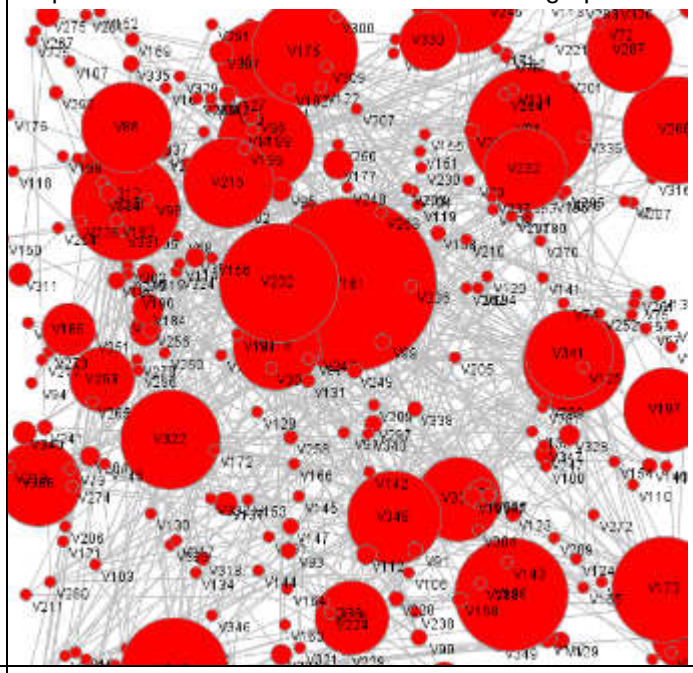
3.2.6 Contenido de red social

Código Requisito	Prioridad	Realización	Descripción
REQ040	Alta	Representación	Los contenidos tendrán asociado un URS que es el que ha creado este contenido para la red social.
REQ041	Baja	Representación	Los contenidos estarán caracterizados con distintos atributos: • Privacidad • Valor Objetivo de red (lo que se valora un contenido) • Valor Subjetivo de red (lo que los involucrados en el contenido lo valoran)

3.2.7 Tiempo de red social

Código Requisito	Prioridad	Realización	Descripción
REQ042	Alta	Representación	El tiempo en la red social evoluciona por Instantes. Un instante es una unidad de tiempo "ficticia" que se puede hacer corresponder con unidades de tiempo real.
REQ043	Alta	Representación	Durante cada Instante, la red evoluciona de alguna forma porque los URS llevan a cabo acciones. Cada URS tiene asociado un número de unidades de ejecución. El número de unidades de ejecución indican cuantas acciones se pueden hacer por Instante. Un usuario en cada Instante, a medida que ejecuta acciones va consumiendo unidades de ejecución hasta agotarlas. Si con las unidades disponibles no puede ejecutar una acción, el coste de ejecutar la acción en el siguiente Instante se decrementa en tantas unidades como se disponga a la hora de ejecutarla.

3.2.8 Representación

Código Requisito	Prioridad	Realización	Descripción
REQ044	Alta	Representación	Krowdix 2.0 permitirá visualizar dinámicamente la evolución de la red social.
REQ045	Alta	Representación	Krowdix 2.0 se basará en la mejores librerías existentes para representación de la red en un espacio bi o tridimensional http://en.wikipedia.org/wiki/Social_network_analysis_software
REQ046	Alta	Representación	Grupos de nodos con atributos comunes se agruparán 
REQ047	Alta	Representación	Se podrán resaltar atributos específicos dentro del modelo para entender cómo evoluciona cada uno de ellos en el sistema (similar a los diagramas de voronoi pero a partir de nodos)

3.2.9 Modelo de usuario

Código Requisito	Prioridad	Realización	Descripción
REQ048	Alta	Representación	Cada Nodo de la red en Krowdix 2.0 es un usuario de una potencial red social y tendrá asociado uno o varios perfiles que identifica sus posibles comportamientos.
REQ049	Alta	Datos	Cada perfil de usuario establece el subconjunto de acciones que tiene este usuario con la red social. Las acciones estarán encaminadas a cambiar el estado de la red social y a afectar a los objetivos que se establezcan para el usuario. De esta forma, si un usuario tiene como objetivo "mirón de red" sus acciones se encaminarán a acceder a la información de un tercer usuario. Si por el contra uno de sus objetivos es localizar

			gente con afinidad (parámetros dentro del rango de afinidad entre miembros) el hecho de conseguir amigos afines ser establecerá como relevante.

3.2.10 Rendimiento

Código Requisito	Prioridad	Realización	Descripción
REQ050	Alta	Rendimiento	Krowdix 2.0 podrá soportar la ejecución de hasta 10.000 nodos en un único equipo estándar.
REQ051	Alta	Datos	Krowdix 2.0 soportará la escalabilidad horizontal permitiendo soportar redes de más de 10.000 nodos utilizando la capacidad de varios equipos en red
REQ052	Alta	Datos	El sistema gestionará el tiempo internamente a través de un generador de "tics" cada uno representando una unidad de tiempo en que los distintos elementos del sistema lleva a cabo una acción
REQ053	Alta	Datos	Todas las acciones del sistema quedarán registradas en un archivo de logs que nos permitirá analizar el comportamiento que se lleva a cabo en cada uno de los momentos del sistema y entender cómo evoluciona el mismo.

3.3 OBSERVACIONES

3.3.1 REQUISITOS INTERFAZ USUARIO

La interfaz de usuario nos debe permitir crear redes, nombrarla y guardarla. El proceso de creación se debe hacer a través de un asistente en los que se lleven a cabo los pasos para la creación. Lo primero que debe hacer el asistente es pedir al usuario un nombre para la red. El asistente debe permitir elegir el número de URS inicial de la red, cada uno de estos URS iniciales debe tener un perfil asociado. El asistente de permitir determinar qué el número de URS va a tener asociado cada perfil. Una vez hecha la asignación de perfiles se puede cerrar la creación de la red.

La interfaz debe tener una funcionalidad para crear perfiles. Esto se hace igualmente con un asistente en el que indicamos el nombre del perfil, y damos un porcentaje de ejecución a cada acción. El asistente no nos debe permitir terminar la creación del perfil si los porcentajes asignados a las acciones no sumas exactamente el 100%.

Otra acción que nos permite la interfaz es cargar una red social previamente creada. Nos debe permitir elegir entre las redes que se encuentran en el sistema y cargar esa red con los datos de la última ejecución que se realizó.

La interfaz debe permitirnos mostrar las diferentes vistas de la red social. Cada vista se debe mostrar por separado, es decir, no podemos mostrar dos vistas a la vez. Y

esta vista se abrirá en una ventana distinta a la de la interfaz de usuario principal. La ventana no debería poder ser movida por el usuario ya que una vez abierta se determinan sus coordenadas en la pantalla para hacer las capturas de pantalla y posteriormente generar el vídeo de evolución de la red según la vista escogida.

3.3.2 REQUISITOS FUNCIONALES

3.3.2.1 Múltiples redes sociales

Puesto que uno de los objetivos de nuestro programa es el estudio de redes sociales, se hace imprescindible el poder guardar diversas simulaciones para poder establecer comparaciones y bifurcar la evolución de una red social en varias en función de uno o varios parámetros.

Por tanto, se establece que la aplicación debe permitir detener y guardar una simulación, para poder reanudarla en cualquier otro momento desde el mismo instante en que se detuvo, sin perder ningún dato. Las redes sociales deben tener suficiente persistencia como para poder cerrar el programa y volverlo a abrir sin perder las redes sociales guardadas.

3.3.2.2 Múltiples visualizaciones

Krowdix debe permitir mostrar una o varias visualizaciones a la vez, tanto si se está desarrollando una simulación como si la simulación se encuentra detenida. Dichas visualizaciones deben mostrar la evolución de características o aspectos concretos de la red social, como nodos, relaciones, distintas métricas... de forma que ayude al usuario a comprender de forma gráfica cómo se está desarrollando la simulación, y hacia dónde va la red social. Cada visualización permitirá la grabación de un vídeo con información del momento temporal, de forma que se pueda analizar posteriormente, e incluso comparar con el resultado de otras visualizaciones, o de la misma visualización de una red social distinta.

3.3.2.3 Gestión de perfiles de usuario

Con la finalidad de poder crear usuarios que se comporten de distinta manera, se debe dotar a la aplicación de un sistema de perfiles de usuario configurable. Estos perfiles definirán de alguna manera las tendencias de acción de los usuarios. Se debe incorporar, además, un gestor gráfico para poder crear, borrar y configurar estos perfiles.

3.3.2.4 Asistente de creación de redes

La aplicación debe contar con un asistente detallado e intuitivo desde el cual se permita crear y configurar una red desde cero, con la elección del número de

usuarios, perfiles de estos, y distribución de las relaciones.

3.3.3 REQUISITOS NO FUNCIONALES

3.3.3.1 Rendimiento vs. tamaño de redes sociales

Este requisito fue uno de los primeros que se planteó, debido al pobre desempeño que daba la versión anterior del programa, y que hacía que los datos no fuesen demasiado reales.
Debido a la propia naturaleza del programa, a mayor número de usuarios simulados, menor rendimiento del programa. Había que decidir un punto intermedio que proporcionase una simulación real sin sacrificar la usabilidad de la herramienta. Después de varias discusiones se llegó a la decisión de garantizar el buen funcionamiento para simulaciones del orden de 10.000 usuarios.

3.3.3.2 Interfaz gráfica intuitiva

La interfaz gráfica es muy importante, ya que supone el contacto entre el usuario y el programa. Un programa muy bueno y potente puede fracasar si no es capaz de atraer a sus usuarios a través de su interfaz, y transmitirles las posibilidades que éste le brinda. Es por ello que se fijan una serie de objetivos para la interfaz de Krowdix:

- Debe ser suficientemente intuitiva como para que el usuario pueda manejar todos los aspectos del programa sin tener que investigar demasiado.
- Debe ser atractiva para que el usuario no se aburra de su uso, y no pierda interés por la aplicación.
- Debe ser lo suficientemente potente como para poder aprovechar todas las ventajas que ofrece el núcleo de la aplicación, sin perder calidad respecto a los dos factores anteriormente mencionados.

En una versión anterior se escogió el Framework JUNG, pero la interfaz obtenida no resultó demasiado atractiva, siendo además excesivamente estática. Esta vez se abordarán otras tecnologías además, como Java3D o JavaFX.

3.3.3.3 Portabilidad

Nuestro sistema funciona para Windows, Linux y Mac OS X, que tengan instalada la máquina virtual java 1.6, así como un servidor MySql.

4 DISEÑO DEL SISTEMA

4.1 INTRODUCCIÓN

En esta sección se hablará de las distintas decisiones que se han tomado para formar el diseño de Krowdix. Algunas de esas decisiones se han tomado basándose en errores vistos en la anterior versión, y otras son simplemente para dar más potencia a la aplicación, lo que se traducirá en nuevas posibilidades abiertas tanto para el desarrollo presente como para futuras ampliaciones.

En términos generales, se ha usado un nuevo modelo de datos donde se distingue entre usuarios (URS), perfiles de usuario, acciones, grupos y contenidos (CRS). Cada usuario tiene un perfil determinado, que marca la preferencia del usuario para hacer unas acciones u otras. Estas acciones implican crear grupos, contenidos, establecer relaciones etc. Cada acción tiene un coste en puntos.

Por otro lado, un motor de tiempos simulará instantes de tiempo. En cada instante, todos los usuarios tendrán un crédito de puntos que gastarán realizando acciones de una lista interna. La formación de esta lista dependerá de los parámetros de su perfil. Se va a usar un nuevo entorno de ejecución basado múltiples threads ligeros que se van a encargar de la ejecución de las acciones de cada URS.

Para la interfaz de usuario íbamos a utilizar la tecnología java FX que permite desarrollar un entorno mucho más vistoso, pero por falta de tiempo y desconocimiento de la tecnología, se ha implementado una interfaz en swing en la versión final.

4.1.1 PROPÓSITO DEL SISTEMA

El sistema va a permitir simular el comportamiento de una red social. Es decir, cómo se comportan los usuarios y cómo son sus relaciones en una red cualquiera como pueden ser facebook, tuenti o twitter. Para ello debe de ser capaz de generar un buen número de URS (agentes) que automáticamente creen relaciones entre ellos, de una forma no aleatoria según sus perfiles y afinidades y se pueda observar la evolución de la red social. Los perfiles de cada URS van a marcar el tipo de acciones más importantes que cada agente va a realizar, de forma que todos los URS pueden ejecutar todas las acciones del sistema pero el perfil marca el porcentaje de acciones que se van a usar en cada instante de tiempo.
Esta evolución deber ser guardada por el sistema para poder representar mediante mapas y diagramas los cambios acontecidos a lo largo del tiempo. También se podrá introducir cambios en la red manualmente con la intención de alterar el normal desarrollo de ésta, y poder apreciar cómo cambiaría la evolución con estos cambios.

4.1.2 OBJETIVO DEL DISEÑO

Nuestro diseño debe permitir la ejecución de hasta 10.000 URS sin que su comportamiento se vea afectado. Para ello utilizaremos un pool de threads ligeros que permiten ejecutar un buen número de agentes sin saturar el sistema.
Un objetivo del diseño es también conseguir una evolución no aleatoria de la red social, es decir, conseguir una red con un buen número de usuarios entre los que exista una lógica entre sus relaciones y que el sistema no acabe llegando a una red enorme en la que todos los usuarios están relacionados con todos. Para conseguir este objetivo se han realizado los perfiles. El usuario, antes de lanzar una red social nueva, va a poder crear una serie de perfiles. Cada perfil tiene asociado un porcentaje de cada acción, este porcentaje va a marcar el tipo de acciones que va a realizar cada perfil. Por ejemplo si creamos un perfil y le asignamos un porcentaje del cincuenta por ciento a la acción petición de amistad, los URS que tengan este perfil, se van a dedicar la mitad del tiempo que les corresponde ejecutar acciones a realizar la acción petición de amistad.
Otro objetivo es marcar los pasos de la evolución de la red social. Para esto se ha pensado en los instantes de tiempo. Cada URS tiene en su estado un instante de tiempo, de forma que puede realizar una serie de acciones en ese instante. Se le ha adjudicado una puntuación a cada acción de forma que cada URS tiene una puntuación restante y al llevar a cabo una acción se le resta a la puntuación restante de usuario la puntuación de la acción. De esta forma, cuando por la carencia de puntación restante, no puede ejecutar ninguna acción más se incrementa el instante de tiempo del URS y se reinicia su puntuación restante.
También debe de ser capaz el sistema de almacenar la evolución de la red social a lo largo de un periodo de tiempo. Para ello vamos a utilizar una base de datos MySQL. Así como representar varias vistas de esta evolución reflejando distintos datos, como la agrupación de los URS según sus características, la forma de la red social o el número de acciones realizado por cada URS en un instante de tiempo. Todas estas vistas deben ser fácilmente ampliadas con distintas medidas o visualizaciones de la red social que puedan surgir en un futuro.

4.2 ARQUITECTURA DE SOFTWARE

4.2.1 PANORAMA

En lo referente a la arquitectura, se ha intentado separar la gestión de la interfaz, de la gestión de la persistencia y la lógica en 3 capas distintas, de modo que sea sencillo afrontar cambios o ampliaciones en el diseño de cada aspecto.
A continuación se explicará con detalle cada una de estas capas, junto a las decisiones de diseño involucradas.

La arquitectura del sistema va a ser Modelo Vista Controlador (MVC) ya que tenemos el modelo de la red social, donde modelamos su comportamiento. Un controlador para introducir cambios en el desarrollo de la red. Y por último las distintas vistas de la evolución del sistema.

4.2.2 DESCOMPOSICIÓN EN SUBSISTEMAS

Podemos diferenciar varios subsistemas o módulos en la aplicación. El módulo más importante, el que podríamos considerar el motor de nuestra aplicación, sería el entorno de ejecución de la red social. Otro módulo usado sería el encargado de llevar a cabo las interacciones con la base de datos MySQL. Tenemos también el módulo encargado de crear las distintas visualizaciones de la red social. Y por último la interfaz gráfica que se va a encargar de mostrar las visualizaciones y demás operaciones para crear redes, perfiles y asignarles a estos las acciones.

El entorno de ejecución o núcleo de la aplicación se encarga de crear la red social y de que cada URS lleve a cabo sus acciones de forma que vaya evolucionando la red social.

4.2.2.1 Núcleo de la aplicación

Esta es, quizá, la parte más importante de Krowdix, ya que es la que define todas las posibilidades del programa. En esta parte se implementan cuestiones tan importantes como el motor de tiempos, el comportamiento de los usuarios y las acciones, grupos y contenidos con los que los usuarios pueden interactuar.

Acciones
Para representar lo que un usuario puede hacer en la red social, se utiliza el concepto de acción. Todas las acciones tienen una estructura similar, por lo que se han utilizado interfaces para favorecer la generalización y facilitar la creación de nuevas acciones en un futuro. Cada acción implementada es diferente al resto, y tiene efectos distintos. En una red social real, algunas acciones tienen un coste en esfuerzo mayor que otras. Por ejemplo, acciones como dejar un comentario o escribir un mensaje tienen menor coste que compartir un vídeo o crear un blog. Para reflejar esa diferencia, cada acción tiene un coste específico de puntos, mayor cuanto mayor esfuerzo requiera.

Diversidad de comportamientos. Perfiles.
Para reflejar la diversidad de usuarios en una red social, se decide implementar perfiles. Cada perfil define la preferencia de unas acciones por encima de otras. Para ello, se asocia a cada perfil las acciones que se prefieran, y se priorizan en forma de porcentajes. De esta manera, el perfil determinará la proporción entre las acciones que el usuario intentará realizar en la simulación.
Se piensa también en las posibilidades de un perfil dinámico, que se adapte a lo que le va ocurriendo a su usuario. Sin embargo, esto es algo que se decide dejar para futuras versiones, para estudiarlo con detenimiento.

Usuarios
Para representar un usuario se usará una entidad llamada URS. Cada usuario estará asociado a un único perfil. Además, tendrá una lista de acciones, que se rellenará con acciones al azar atendiendo a los porcentajes que marca su perfil. En la simulación, el usuario consumirá y ejecutará acciones de esta lista de la forma en que se explicará más adelante. Los usuarios podrán establecer relaciones de amistad entre sí, y relaciones de pertenencia y creación con grupos y contenidos.
En un primer momento se decide utilizar una infraestructura basada en agentes, de manera que cada agente fuera un usuario. Se piensa, más concretamente, en la

librería DIET para esto. Pero tras implantarlo en Krowdix, se descubre que no cubre bien las necesidades del sistema, dando más problemas que alegrías. Es por esto que se decide echar marcha atrás e implantar otras soluciones de nuestro propio diseño.

Motor de tiempos y tabla de usuarios

Esta parte es muy importante, ya que es la que dirige la simulación. Después de muchas ideas, se decide implementar esta parte en forma de bucle infinito. Cada vuelta del bucle representa un instante de tiempo distinto. En cada instante, se forma una lista con todos los usuarios existentes en ese momento, se les da un crédito de puntos para gastar y se manda a cada uno ejecutar acciones de su lista hasta agotar el crédito, o hasta llegar a una acción cuyo coste es mayor al crédito que le queda. Todo esto lo realiza un único thread destinado a gestionar este motor de tiempos.
Para poder acceder rápidamente a todos los usuarios, se mantiene una estructura en forma de tabla hash donde se almacenan los usuarios existentes en el sistema.
En un futuro se trabajará para modularizar mejor esta parte.

Paralelizar la simulación. Pool de threads

Debido a la cantidad de usuarios que pueden llegar a crearse, el bucle del motor de tiempos puede llegar a ser muy lento. Para solucionar esto, se llega a la conclusión de que se debe paralelizar la simulación, y dejar la ejecución de acciones de cada usuario en cada instante a cargo de threads independientes. Esto hace pensar que se pueden llegar a necesitar muchos threads. Los threads en java no se pueden reciclar una vez han terminado su ejecución, por lo que es necesario alcanzar una solución a través de la cual se evite el coste en tiempo y recursos de crear y arrancar un thread cada vez que un usuario tiene que ejecutar sus acciones.
Para gestionar todo esto se comienza a diseñar un pool mixto de threads y tareas. A estos threads se les pueden asignar tareas a ejecutar. En caso de no tener tareas que ejecutar, los threads se mantienen a la espera de tareas. Al recibir una tarea, el thread la ejecuta y vuelve a quedar como disponible. El pool es el encargado de recibir tareas y almacenarlas en el pool. En caso de que haya threads disponibles, va asignándoles las tareas que hay pendientes de ejecución. En caso de ser necesario, se crean nuevos threads hasta un límite máximo.
Con esta solución, conseguimos ganar flexibilidad y velocidad de la ejecución. Cada vez que un usuario tiene que ejecutar las acciones de un instante, se crea una tarea con ello y se manda al pool para que se ejecute cuando haya recursos disponibles para ello. Además, en caso de saturación, el sistema automáticamente se adapta para liberar el cuello de botella.

Múltiples redes sociales

Para poder dar soporte a diversas redes sociales, se piensa en varias alternativas. Algunas pasaban por tener varias copias de las estructuras necesarias para una red, de forma que cada copia representase una red social lista para reanudar la simulación. Al final se toma la decisión de mantener una sola copia global de las estructuras necesarias, y actualizarlas cada vez que se quiere cambiar de red.
Sin embargo, es posible que en un futuro esta parte del diseño se revise con el fin de evitar algunos problemas de sincronización.

El módulo de interacción con la base de datos se encarga de crear el modelo de datos de la red social según va recibiendo la información desde el entorno de ejecución. Debido a la penalización que suponen todas las operaciones de entrada salida, este

módulo se puede dividir en dos partes, una de recepción de peticiones por parte del entorno de ejecución y otra de comunicación con la base de datos en sí misma. El envío de las peticiones por parte del entorno de ejecución es asíncrono, de forma que éste envía la petición y sigue con su ejecución sin esperar a que esta finalice, de eso se encarga el módulo.

4.2.2.2 Módulo interacción con la base de datos

Mediante esta capa, se pretende independizar completamente la gestión de la base de datos del resto de la aplicación. Esto facilita enormemente las tareas de mantenimiento de la base de datos, al tener delimitadas las clases que interactúan con la misma.

DAOs
Debido a la complejidad del esquema de base de datos, se opta por implementar un DAO para mantener cada tabla. Un DAO será en adelante una clase que implemente todas y cada una de las consultas, actualizaciones y modificaciones de la tabla de la base de datos a la que se asocia. Mediante esta estrategia se consigue ganar flexibilidad y comodidad en caso de que haya que modificar las tablas, debido a que el código afectado está perfectamente localizado y acotado al DAO correspondiente a la tabla modificada.

Punto de entrada común. GestorDB
La gran cantidad de tablas necesarias en la base de datos, y por consiguiente, de DAOs para administrarlas, hace complejo conocer en cada momento el DAO necesario para gestionar los datos, ya que implica conocer la estructura de tablas para desarrollar algo a nivel de la lógica de negocio. Debido a que esto contradice las intenciones iniciales de mantener la gestión de la base de datos lo más independiente posible, se toma la decisión de implementar un único punto de interfaz entre la lógica de negocio y núcleo de la aplicación y la gestión de la base de datos. Este punto de acceso será la clase GestorDB, que contendrá todas las funciones necesarias desde las capas superiores, dejando la responsabilidad de conocer la estructura de tablas y DAOs al programador de estas funciones.
Esta estructura hace posible poder dividir el equipo de desarrollo, de forma que se pueda destinar un subgrupo de personas al desarrollo, mantenimiento y ampliación de esta capa, incluyendo GestorDB, y otros equipos al desarrollo del resto de la aplicación, sin necesidad de que estos conozcan los detalles de implementación tanto de la estructura de la base de datos, como su gestión y código de mantenimiento. Lo único que necesitan estos equipos es que las funciones de GestorDB estén muy bien documentadas, por lo que se ha de poner especial cuidado en este aspecto.

Retardo de conexiones. ConnectionPool
Para hacer una consulta o query a la base de datos, antes hay que abrir una conexión por la que mandarla. Esto es significativamente más lento que la propia consulta, por lo que el proceso entero se ralentiza. Por ello, en aplicaciones con mucha carga de consultas como es Krowdix, esta lentitud pasa factura en cuestiones de rendimiento.
Por ello, se decide implementar un pool de conexiones abiertas, llamado ConnectionPool. La misión del pool es abrir una cantidad suficiente de conexiones al arrancar el programa, y servirlas bajo demanda conforme se vayan necesitando,

devolviéndose al pool cuando haya terminado de usarse. De esta manera, las conexiones permanecen siempre abiertas y se pueden mandar a través de ellas las consultas sin perder tiempo en operaciones redundantes, como la apertura y cierre de las mismas.

Retardo de E/S. Pool de queries

Las escrituras en base de datos implican el uso de E/S, lo que es varios órdenes de magnitud más lento que las lecturas y escrituras en memoria. Esto hace que cada vez que un thread del programa necesita escribir algún cambio en base de datos, bloquee su ejecución hasta que la operación termina. Debido a esto, la aplicación es más lenta de lo deseado, ya que todos los threads entran en bloqueo muchas veces por las abundantes operaciones de E/S.

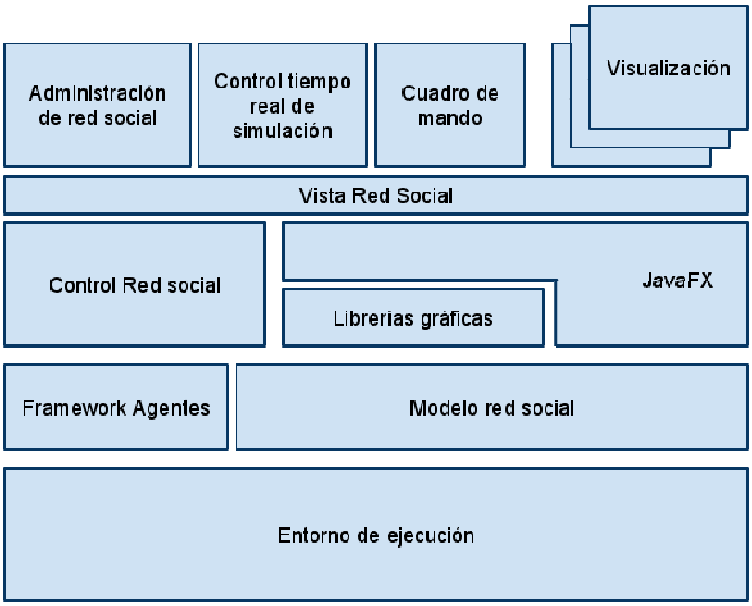
Para intentar paliar este efecto, se piensa en una solución que separe las escrituras en base de datos de la ejecución normal. Para ello, se implementa un módulo consistente en una serie de colas que almacenan queries de escritura pendientes de ejecutarse. Este módulo tiene su propio juego de threads cuya labor es exclusivamente extraer queries de estas colas y ejecutarlas. Mediante esta solución, se desacoplan las operaciones lentas, como son las escrituras en base de datos, que recaen sobre otro conjunto de threads distinto al que realiza la ejecución del programa y la simulación. Esto ayuda a que la ejecución tanto de la simulación como de la aplicación en general sea mucho más ágil y ligera.

Hay que decir, no obstante, que este módulo no está del todo integrado con la capa de gestión de base de datos, y se pierde en parte la independencia de la misma. Esto es un punto a mejorar en un futuro que, de seguro, se tendrá en cuenta.

4.2.2.3 Módulo de visualizaciones de la red social

Por último tenemos el módulo de vistas que se encarga de hacer un modelo de las distintas visualizaciones al que accederá la interfaz gráfica para pintarlas.

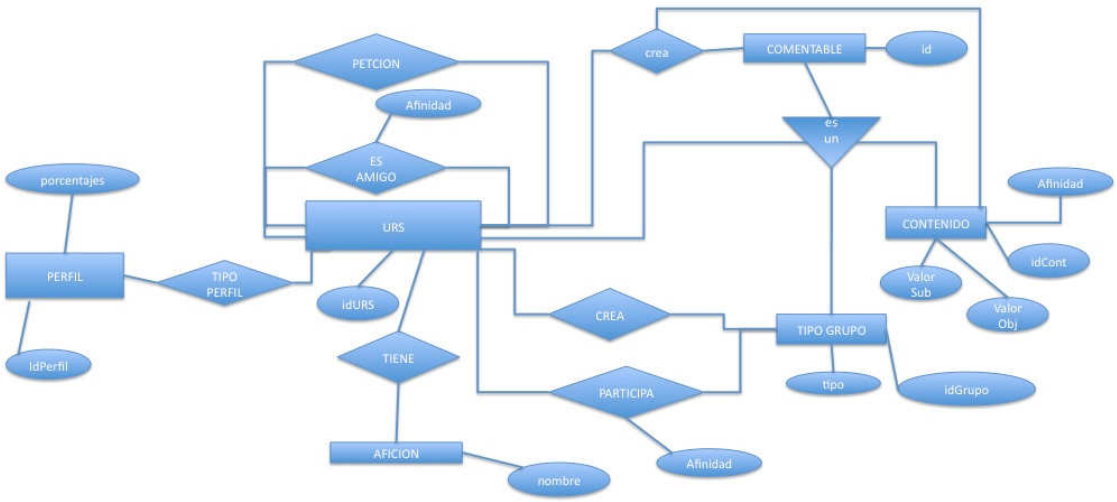
Este módulo es el encargado de, una vez activado por parte del usuario, mostrar para cada instante de tiempo una “foto” que indique el estado de la red social en ese momento. Hay varios tipos de visualizaciones, hemos utilizado grafos, gráficas y otros diseños para reflejar las distintas perspectivas con las que podemos mirar el estado de la red social.



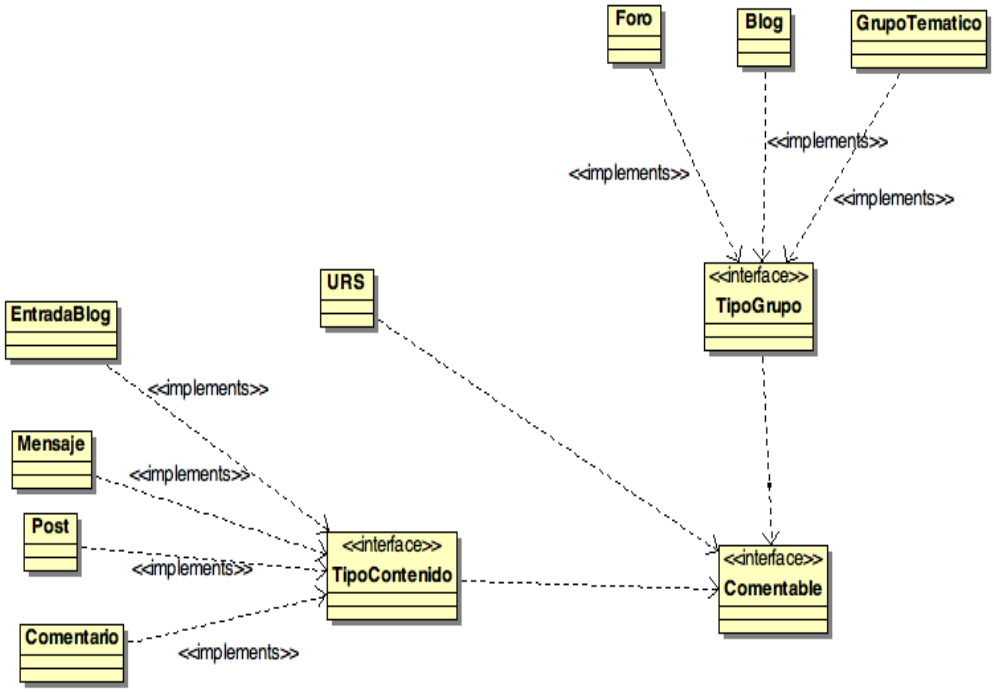
4.2.3 ADMINISTRACIÓN DE DATOS PERSISTENTES

Para la persistencia de datos vamos a utilizar una base de datos en MySQL. En ella vamos a guardar los URS y sus perfiles. Así como las distintas relaciones entre los URS y las acciones que éstos realizan. Para guardar la evolución de la red debemos marcar cada acción de los URS con el instante de tiempo en el que se producen.

Diagrama E-R de la base de datos:



4.2.4 DIAGRAMAS DE CLASES



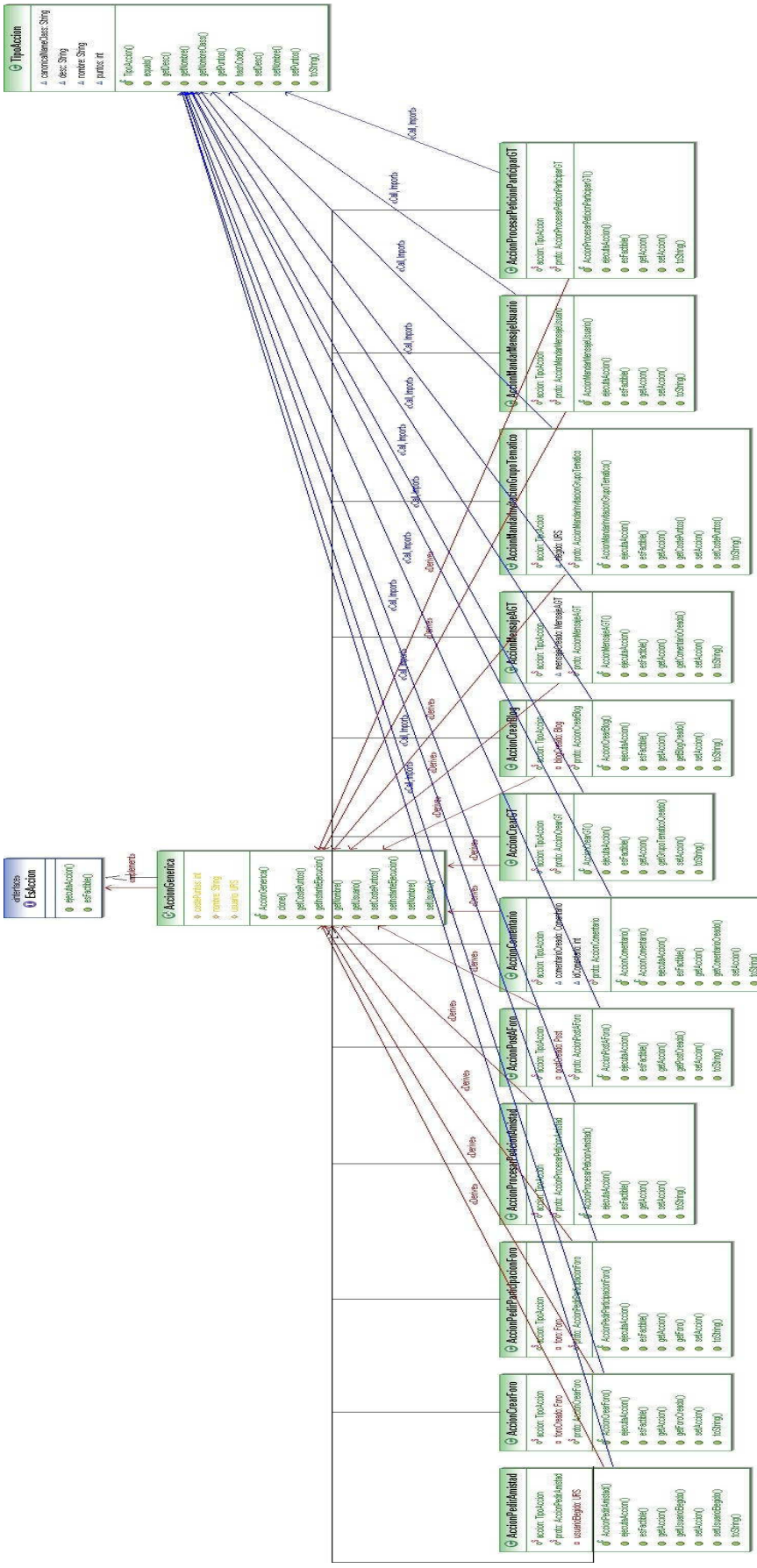
Tipos de Contenido:

- Entrada Blog: afecta al tipo grupo Blog.
- Post: afecta al tipo grupo Foro.
- Mensaje: afecta al tipo URS.
- Comentario: afecta a otro Contenido.

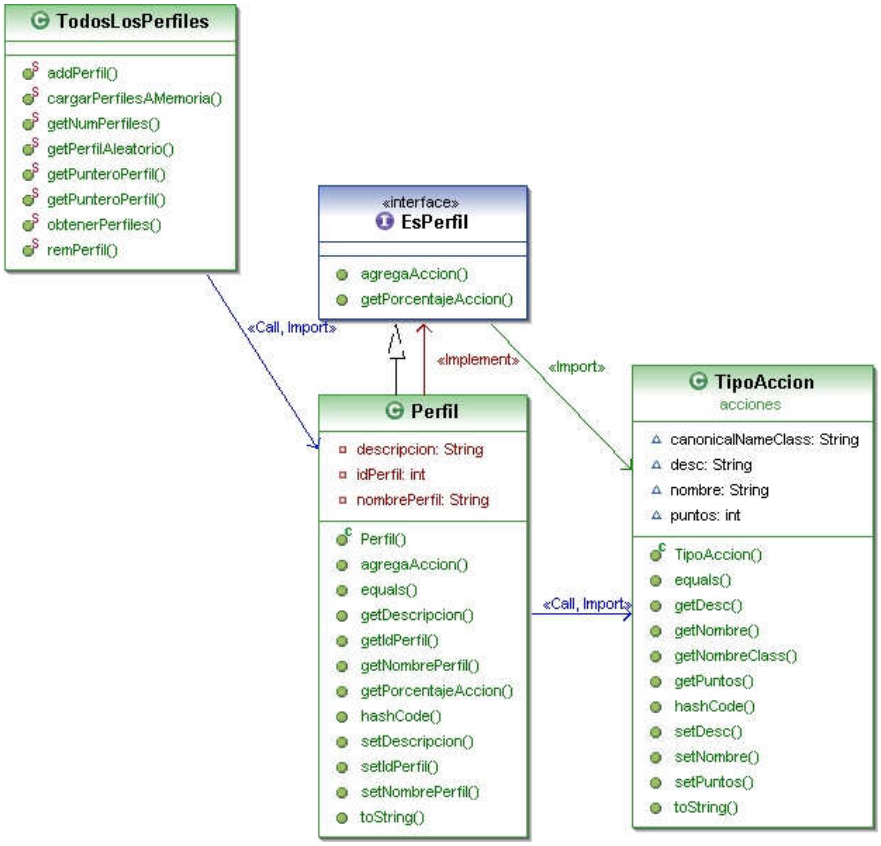
Interfaces:

- La interfaz *Comentable* aglutina todos los elementos sobre los que se puede generar un comentario: usuarios, otros contenidos y grupos.
- La interfaz *TipoGrupo* sirve para representar los distintos tipos de grupos que existen: foros, grupos temáticos, blogs.
- La interfaz *TipoContenido* representa los distintos contenidos que puede generar un usuario de la red social.

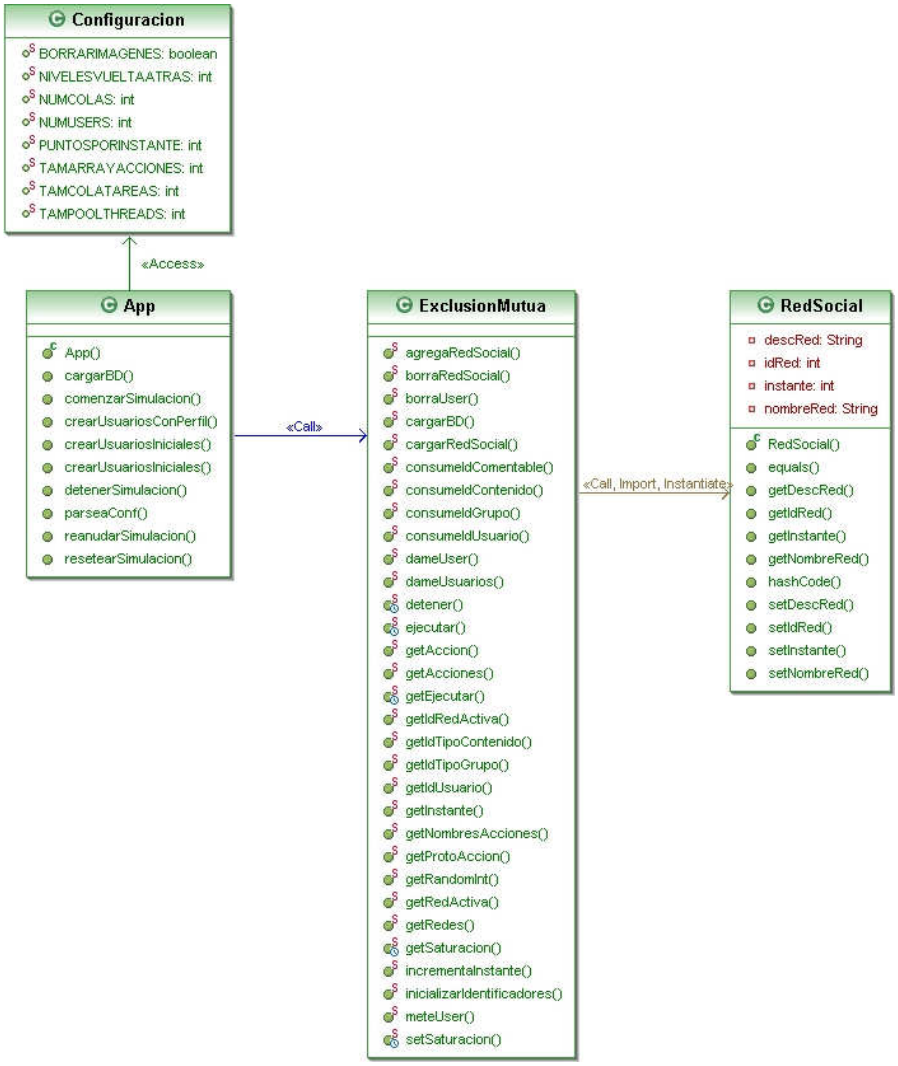
ACCIONES



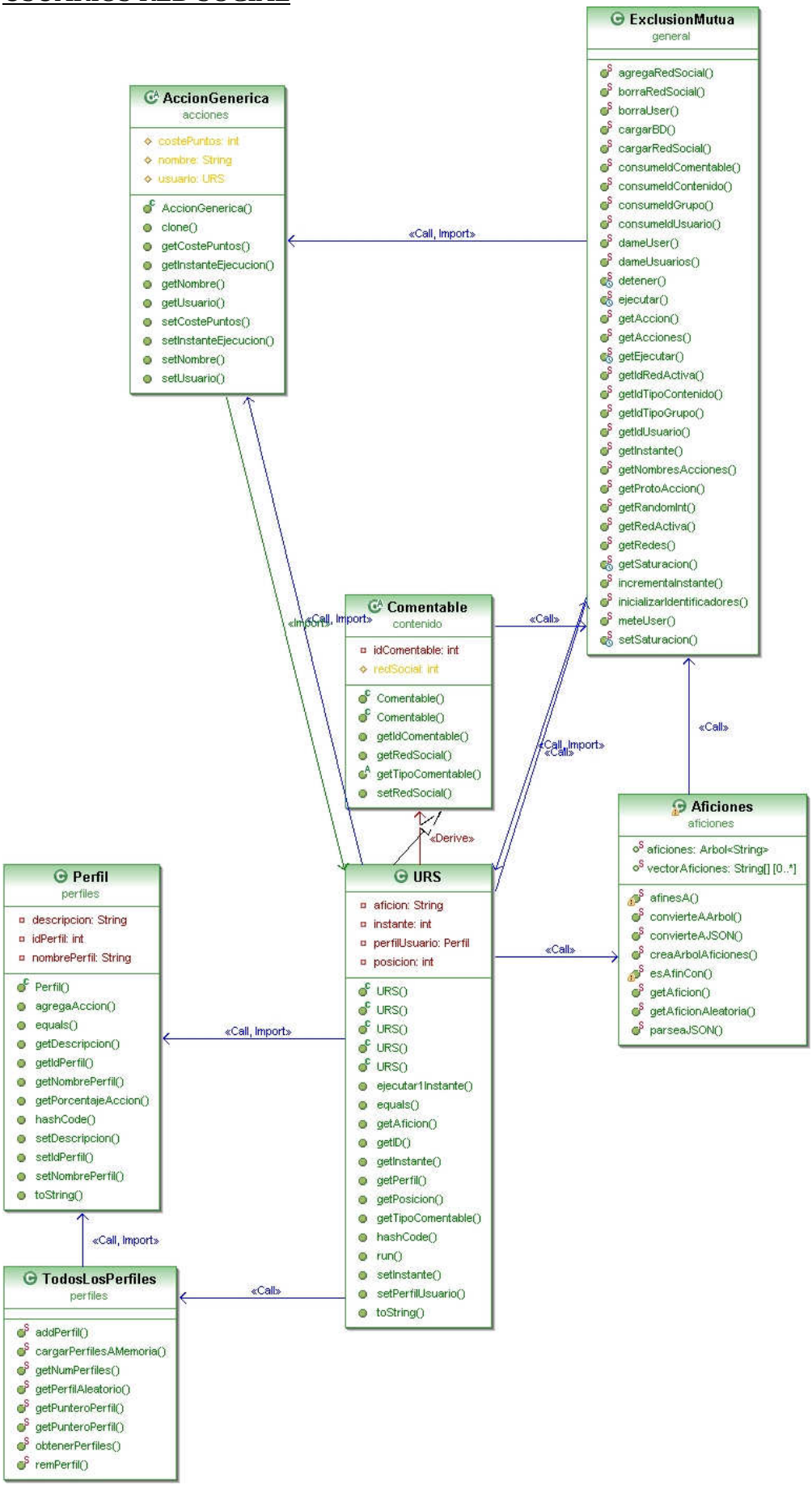
PERFILES


TodosLosPerfiles
addPerfil()
cargarPerfilesAMemoria()
getNumPerfiles()
getPerfilAleatorio()
getPunteroPerfil()
getPunteroPerfil()
obtenerPerfiles()
remPerfil()
«interface»
EsPerfil
agregaAccion()
getPorcentajeAccion()
«Call, Import»
«Implement»
«Import»
Perfil
descripcion: String
idPerfil: int
nombrePerfil: String
Perfil()
agregaAccion()
equals()
getDescripcion()
getIdPerfil()
getNombrePerfil()
getPorcentajeAccion()
hashCode()
setDescripcion()
setIdPerfil()
setNombrePerfil()
toString()
«Call, Import»
TipoAccion
acciones
canonicalNameClass: String
desc: String
nombre: String
puntos: int
TipoAccion()
equals()
getDesc()
getNombre()
getNombreClass()
getPuntos()
hashCode()
setDesc()
setNombre()
setPuntos()
toString()

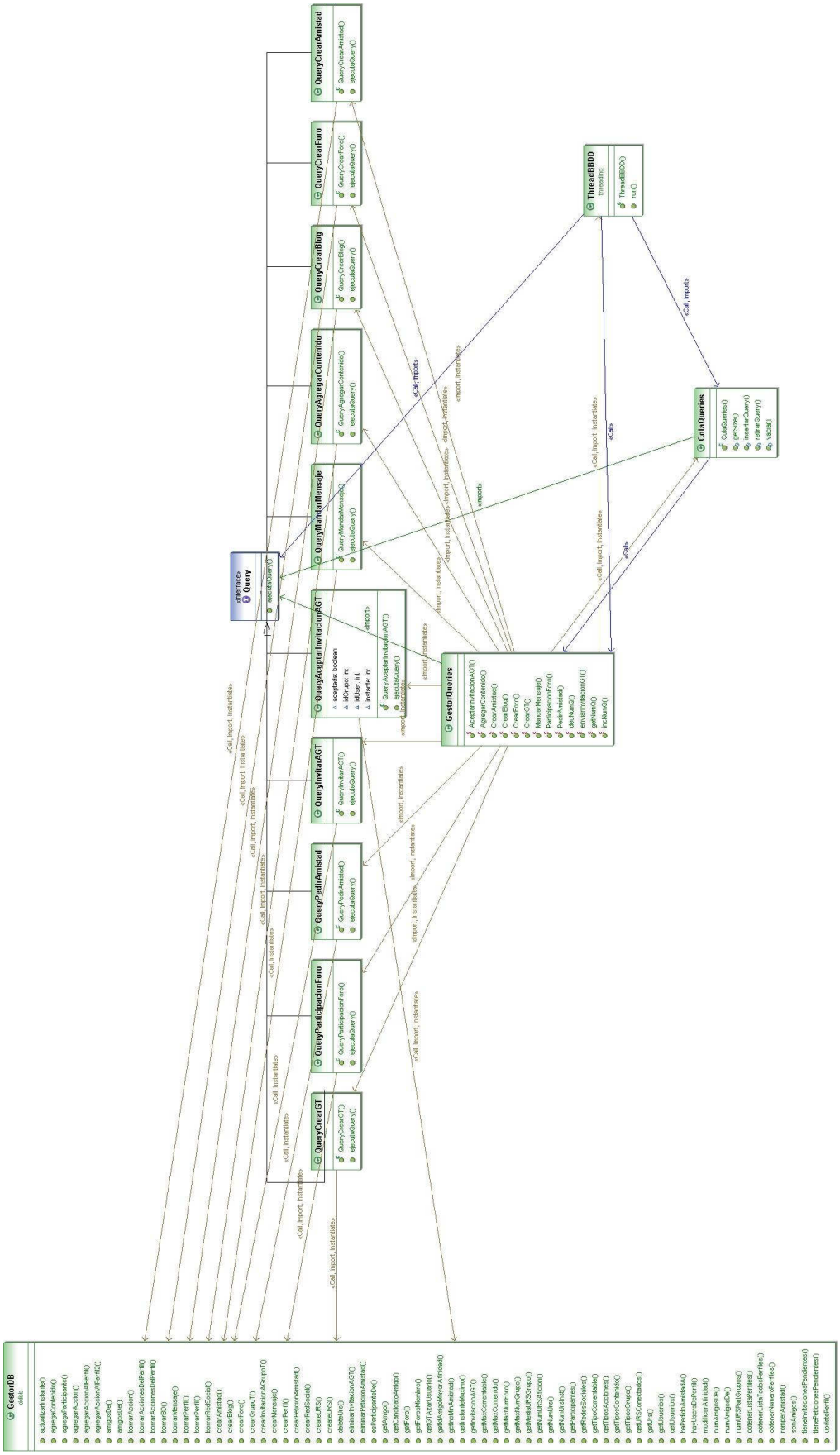
CONTROLADOR DE LA APLICACIÓN



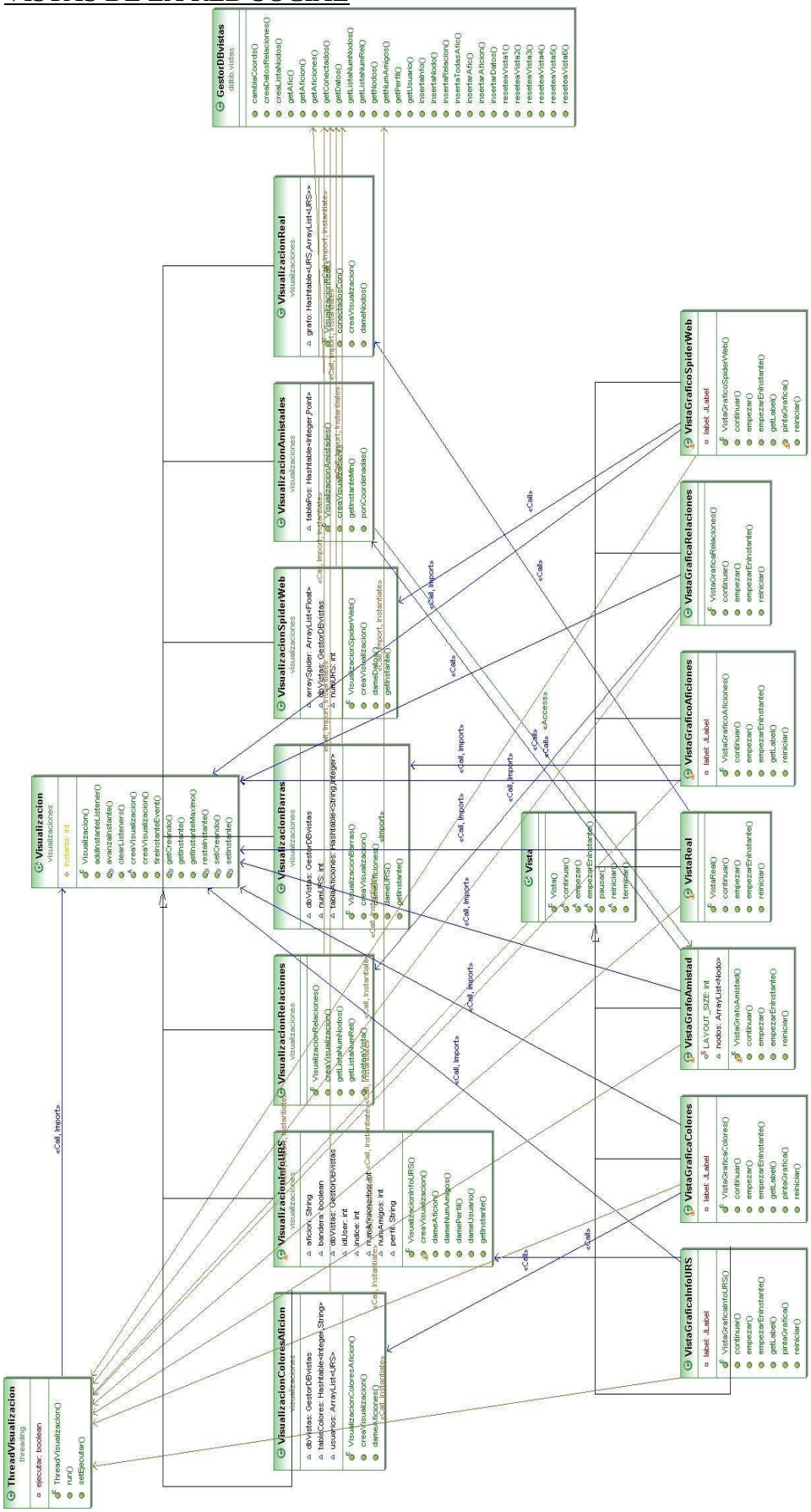
USUARIOS RED SOCIAL



PETICIONES E/S



VISTAS DE LA RED SOCIAL



5 LIBRERÍAS UTILIZADAS

En el sistema Krowdix 2.0 hemos utilizado una serie de librerías que implementan distintas funcionalidades de modo que nos hemos podido ahorrar ese trabajo. Ha habido veces en los que pensábamos que la librería nos iba a solucionar muchos problemas y nos podría ayudar en el diseño del sistema pero esto no ha sido así siempre y hemos tenido que desecharla por completo e implementarnos una solución alternativa. El aprendizaje en el uso de la librería ha sido un factor que ha determinado su uso, ya que existen distintas librerías para una misma funcionalidad y nos ha parecido más apropiado el uso de una u otra por experiencias previas con ella.

5.1 DIET

DIET es una plataforma para desarrollar aplicaciones basadas en agentes. El proyecto DIET (Decentralised Information Ecosystem Technologies) es una plataforma que trata de solucionar los problemas de adaptabilidad y escalabilidad de los sistemas multi-agente existentes. El objetivo principal del proyecto es implementar un conjunto de herramientas de desarrollo de agentes para el apoyo a aplicaciones complejas del mundo real, tales como la gestión de fuentes de información como Internet o intranets a gran escala. Utiliza un diseño “bottom-up” para asegurar que la plataforma es ligera, escalable, robusta, adaptable y extensible. Es especialmente adecuado para desarrollar rápidamente aplicaciones peer-to-peer de prototipos y / o de adaptación, aplicaciones distribuidas que utilizan técnicas inspirados en la naturaleza.

La principal razón por la que se planteó la utilización de la librería DIET fue por su escalabilidad, es capaz de ejecutar un número muy grande de agentes, hasta unos cientos de miles debido a la ligereza de cada agente. Cada agente solamente tiene una mínima capacidad para ejecutarse y comunicarse con otros agentes. Sin embargo, la forma en la que se ejecutan cada uno de estos agentes y hacen uso de un pool de threads generado por DIET no fue nada útil para nuestros propósitos.

Cada agente que creábamos con DIET hacia uso de un thread para ejecutarse, de tal forma que como en nuestra aplicación los agentes crecen exponencialmente se saturaba el pool de threads muy rápidamente. Esto era debido a que nuestros agentes nunca terminan la ejecución hasta que no se detiene la simulación de la red social, de tal forma que DIET no podía arrebatarle el uso del hilo para que pudiera utilizarlo otro agente para realizar otra acción.

Esto lo solucionamos desechando totalmente la idea de utilizar DIET, y nos creamos nuestro propio pool de threads de tal forma que en el momento en que un agente de nuestro sistema termina de ejecutar un instante de tiempo, se libera el hilo para que otro agente pueda ejecutar el instante que le toca. De esta forma hemos llegado poder ejecutar en una sola máquina hasta más de diez mil agentes

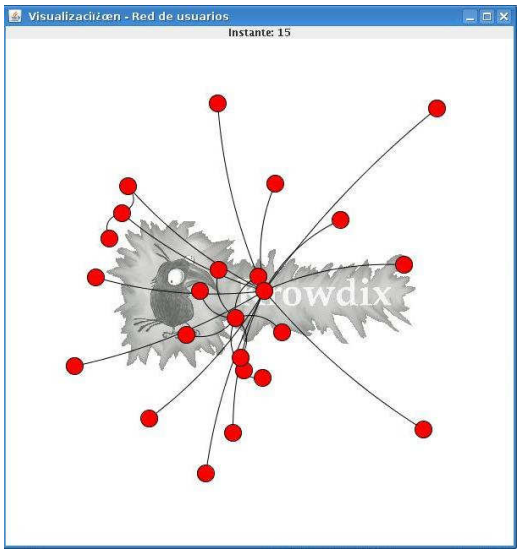
5.2 JUNG 2.0

JUNG 2.0 es una librería para la creación, visualización y análisis de grafos. Está escrita en Java, de tal forma que es perfectamente integrable en nuestro sistema y podemos utilizar completamente su API.

La arquitectura de JUNG está diseñada para soportar una variedad de representaciones de entidades y sus relaciones, como grafos dirigidos y no dirigidos, grafos multi-modal, grafos con bordes paralelos, e hipergrafos. Proporciona un mecanismo para anotar los grafos, las entidades y las relaciones con metadatos. Esto facilita la creación de herramientas de análisis de conjuntos de datos complejos que se pueden examinar las relaciones entre entidades, así como los metadatos adjuntos a cada entidad y relación.

JUNG incluye implementaciones de varios algoritmos de teoría de grafos, minería de datos y análisis de redes sociales, incluida la agrupación, el filtrado, la generación de gráficos al azar, blockmodeling, el cálculo de las distancias de las redes y flujos, y una amplia variedad de métrica (PageRank, HITS, intermediación , cercanía, etc.).

Jung también proporciona un marco de visualización que hace que sea fácil de construir herramientas para la exploración interactiva de datos de red. Los usuarios pueden utilizar uno de los algoritmos de diseño de siempre, o utilizar el marco para crear sus propios diseños personalizados.



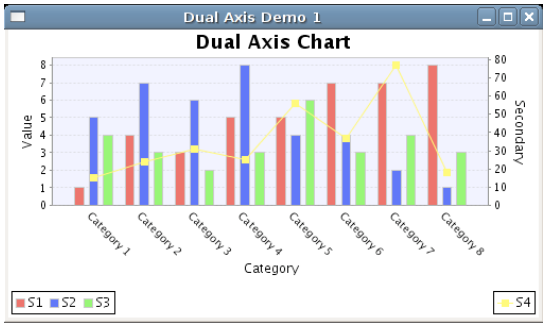
Además existe una amplia documentación on-line en la página web de la librería así como el API de la librería y algunos ejemplos que nos han sido muy útiles para trabajar con ella.

5.3 JFREECHART

JFreeChart es una librería de código libre implementada en Java, que facilita a los desarrolladores incluir gráficas en sus aplicaciones. Tienes varios formatos de salida para los gráficos compatibles con los componentes Swing, lo cual es importante para

nuestro sistema, o archivos de imagen como jpg o png.

Hemos utilizado esta librería para mostrar gráficas sobre distintos aspectos de la red social. Es relativamente fácil utilizarla ya que utiliza unas estructuras para la introducción de los datos de los diferentes ejes partes del gráfico y ella crea el gráfico que lo incluimos como un componente Swing en la aplicación.



Existe el API en la página web de la librería pero el manual para los desarrolladores es de pago, lo que hizo un poco más difícil conocer la librería. Aún así hemos podido utilizar todo el potencial que necesitábamos para nuestra aplicación.

5.4 JDBC

JDBC es un conector o driver para usar la base de datos MySQL, ***Java Database Connectivity***, más conocida por sus siglas **JDBC**, es una API que permite la ejecución de operaciones sobre bases de datos desde Java, independientemente del sistema operativo 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 biblioteca de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar con cualquier tipo de tareas con la base de datos a las que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

5.5 JSON

Las configuraciones del sistema Krowdix 2.0 se guardan en un archivo json, el cual se puede modificar para cambiar los parámetros de la aplicación. Para leer estas configuraciones hemos creado una librería que realiza el parseo del archivo json. Este librería comprueba la sintaxis del archivo y asignamos cada valor configurable de la aplicación mediante el archivo parseado.

5.6 VIDEO

Se ha creado una librería de video para hacer las capturas de pantalla de las distintas visualizaciones que muestra la aplicación en cada instante. Una vez hechas las capturas de pantalla, al cerrar la visualización, todas estas imágenes se unen para formar un vídeo que muestra la evolución de la red en el tiempo.

Las capturas se realizan estableciendo una zona del escritorio que va a ser capturada cada vez que cambia el instante de la visualización. De forma que si se mueven las ventanas de la visualización o se superponen unas sobre otras, la captura no se hará correctamente. Por esta razón hace falta que el usuario conozca esta limitación para que no mueva las ventanas de la visualización por el escritorio, o que no superponga otro elemento en la vista si quiere que el video se realice correctamente. También esta limitación ha hecho que no podamos mostrar dos visualizaciones a la vez.

6 DESARROLLOS FUTUROS

Hablaremos ahora de las mejoras que se pueden o deben hacer sobre la versión presentada de Krowdix, bien por cuestiones de rendimiento, o bien por mejorar y ampliar la funcionalidad y la interacción con el usuario.

6.1 Pool de consultas a base de datos

Actualmente hay un sistema de colas de consultas necesario para desacoplar las escrituras de base de datos y que las lleven a cabo threads independientes. Sin embargo, este sistema no está bien integrado en la capa de base de datos, siendo totalmente necesario conocer su estructura para poder hacer uso de él. Además, usa clases distintas para cada tipo de consulta, lo cual se convierte en algo difícilmente escalable y configurable. Además, se han apreciado problemas espúreos de sincronización debido a que el preprocesado de las consultas se realiza en el momento en que esta se saca de la cola en lugar de realizarse en el momento de su generación.

Por todo esto se ha estudiado y se plantea implantar un módulo consistente en un pool de consultas más global, que autogestione los recursos que necesita para su ejecución de una forma mucho más transparente, dotado además de señales para indicar el nivel de saturación del mismo. Este módulo pasaría a sustituir el sistema de colas comentado, y se situaría al mismo nivel que los DAOs, haciendo innecesario conocer su existencia desde otras capas. De esta manera, además, su configuración y su uso pasan a no influir en el código de otras partes del sistema, volviendo a tener una capa de gestión de base de datos totalmente completa y con diseño de caja negra, cuyo único punto de acceso vuelve a ser el gestor de base de datos, llamado GestorDB.

6.2 Gestión de múltiples redes

Es importante mejorar esta parte del diseño. Debido a la implementación actual con estructuras únicas, unido a las técnicas de buffering usadas para el pool de Conexiones y las colas de consultas explicadas anteriormente, bajo ciertas condiciones de saturación se producen incoherencias en los datos al transitar de una red social a otra creada. Estos problemas son fácilmente evitables limitando las acciones en interfaz del usuario. Sin embargo, el diseño actual presenta problemas de escalabilidad.
Por ello, sería interesante replantear esta parte del diseño para evitar limitar las acciones del usuario innecesariamente, además de poder solucionar los problemas de incoherencia. Para ello, se ha pensado en un diseño de compartimentos estancos, en el que cada red tuviese su propio espacio de memoria, repartido en los recursos que necesita para la ejecución (threads, buffers...), su tabla de usuarios y algunos otros datos. Estos cambios harían mucho más simple la gestión del cambio de red social activa, evitando parches innecesarios y mejorando la apariencia y la interacción con el usuario. Además, se espera que este cambio, junto con el comentado en la sección

anterior del pool de consultas, mejore apreciablemente el rendimiento global de la aplicación.

6.3 Referencia a datos globales

En la versión actual de Krowdix hay datos que son comunes a todas las redes sociales: acciones disponibles, tipos de grupos, tipos de contenido... Para ahorrar memoria, sólo se mantiene una copia de estos datos. Sin embargo, para referenciar algunos de estos datos, tanto en el código como en base de datos, se usan cadenas de texto en lugar de índices o identificadores numéricos. Esto hace muy complicado cambiar los nombres de este tipo de datos, teniendo que buscar y modificar todo lo referido a ellos a lo largo del código que lo gestiona.
Para evitar todos estos problemas, se hace necesaria una revisión de la parte del modelo de datos que controla este tipo de información, y usar índices y referencias que aísle el dato concreto de su representación en texto o pantalla. Esto ayudaría a poder personalizar y cambiar la configuración de esta información sin que ello conlleve la revisión de código. Además, daría pie a una aplicación mucho más personalizable, al poder, por ejemplo, diseñar un gestor gráfico de acciones que cambiase el nombre, la descripción y los puntos asignados de una forma limpia.

A parte de todas estas ventajas, si se aplicara esta mejora junto con la anteriormente explicada de la gestión de múltiples redes, se podría plantear también el hecho de que cada red tuviese una copia de estos datos globales a los que acceder. Con ello cada red se podría configurar con conjuntos de acciones, perfiles, grupos o contenidos distintos a otras redes, lo que permitiría adquirir una nueva perspectiva en comparaciones y estadísticas.

6.4 Realización de vídeos de las visualizaciones

Las visualizaciones, al abrirse, comienzan a capturar imágenes que finalmente forman un vídeo para tener constancia de la evolución de la visualización. Pero debido a limitaciones de última hora, no es posible grabar el vídeo sin tener la visualización abierta, así como tampoco se puede solapar con otras visualizaciones.

Sería interesante trabajar en esta parte para poder independizar el proceso de creación del vídeo a las condiciones del entorno de la visualización, así como su estado.

6.5 Mejora en el modelo de persistencia

Krowdix es una aplicación bastante intensiva en generación de datos, cuando está en modo simulación. Esto implica que consume un ancho de banda importante de E/S por las operaciones de base de datos, así como una cantidad creciente de memoria por la creación de nuevos usuarios. Con respecto a la base de datos, no existen canales separados para las escrituras y las lecturas, sino que comparten el mismo ancho de banda (y el mismo número de conexiones máximas). Si bien es cierto que es muy posible que el número de lecturas por unidad de tiempo sea menor

al de escrituras, es relativamente fácil que las lecturas se vean penalizadas en tiempo por la falta de conexiones disponibles (por estar éstas utilizándose para las escrituras). Esto tiene un doble efecto. Por un lado, el rendimiento de la aplicación se ve afectado por esas pequeñas latencias de tiempo. Por otro lado, el uso de conexiones para las lecturas hace que no estén disponibles para las escrituras, ralentizando éstas. Todo esto teniendo en cuenta que tan sólo un 5% aproximadamente de la persistencia tiene reflejo en memoria.

Si se incrementase este porcentaje, se podrían ahorrar muchas peticiones de datos, lo que aligeraría el canal de E/S e incrementaría el rendimiento de la aplicación. A cambio, habría que pagar un pequeño incremento en el consumo de memoria de la aplicación.
Es importante antes de acometer esta medida hacer un estudio de qué información reflejar en memoria, y de cuánto va a ser ese coste en memoria. Para aprovechar al máximo esta mejora se debe elegir la información que más veces se pida por unidad de tiempo a base de datos. Como la mayoría de los casos esto depende de los porcentajes de los perfiles, la mejor solución es tomar un caso medio, que proporcione el máximo beneficio posible en la mayoría de los casos.

Mirando muy por encima, un ejemplo de esta medida podría ser incorporar al usuario listas con sus grupos, o amigos.

6.6 API para el desarrollo de acciones

Desde el comienzo del diseño se ha buscado hacer de Krowdix una herramienta totalmente configurable, para poder reflejar la mayor cantidad de casos distintos. Esto se ha conseguido incorporando diversos tipos de grupos, contenidos, la flexibilidad en la configuración de perfiles y acciones... Sin embargo, la parte referente a acciones ha sido siempre algo bastante estático debido a que nuevas acciones implica código nuevo, algo que no es compatible con el carácter monolítico del núcleo de Krowdix.

Con un poco de trabajo sobre la parte del modelo de datos y la arquitectura de las acciones sería posible conseguir desarrollar una API básica para la creación de nuevas acciones. Esta API, unida a la API del núcleo de la aplicación serviría para que otros usuarios pudiesen desarrollar nuevas acciones, exportables en ficheros JAR. Estos ficheros se cargarían dinámicamente en el núcleo y serían añadidos al sistema limpiamente. Esta forma de desarrollo se podría aplicar también al diseño de los tipos de grupo y contenidos existentes.

Otra vía de mejora podría ser el mantenimiento de los grupos y contenidos en el núcleo del sistema, unido a un comentario hecho anteriormente (cada red tendría su propia lista de acciones, contenidos y grupos disponibles para usar), de forma que se pudiese elegir en varios pasos del asistente de creación de redes qué acciones, tipos de grupo y tipos de contenido usar para la nueva red.

7 MANUAL DE USUARIO

7.1 REQUISITOS DEL SISTEMA

- Sistema operativo capaz de ejecutar la máquina virtual de Java JVM versión 1.6 o posterior de SUN.
- Servidor de base de datos MySQL versión 5.0.51 o posterior, y cliente MySQL o similar.
- Espacio en disco 20 Mb + espacio extra para los videos de visualizaciones.
- 200 Mb de memoria RAM libres.
- Descompresor de archivos .ZIP.

7.2 INSTALACIÓN

Para llevar a la instalación hay que descargar el programa desde https://sourceforge.net/projects/krowdix/. La descarga tendrá forma de fichero .zip, que contendrá tanto los ficheros binarios, como los de configuración, necesarios para la instalación y ejecución de Krowdix.

Se debe elegir una ubicación para el programa y extraer allí el fichero comprimido. Se obtendrá así un directorio llamado *krowdix* con todo lo necesario dentro.

A continuación, se debe ejecutar un script sql para construir la base de datos y meter en ella los datos iniciales necesarios para la ejecución del programa. Por tanto, en este paso, es imprescindible que la base de datos esté en marcha, y escuchando conexiones por el puerto 3306. El script está en el directorio *krowdix/tablas/* con el nombre *instalacion.sql*.

Llegados a este punto, la aplicación ya estará lista para su uso. Para ello, se deberá ejecutar el script *krowdix.bat* en los sistemas Windows, o *krowdix.sh* en sistemas UNIX o Mac. En caso de sistemas UNIX, pudiera ser necesario asignar permisos de ejecución al fichero *krowdix.sh*. Para ello, ejecutar el comando *chmod +x krowdix.sh* desde el directorio *krowdix*.

7.3 VENTANA PRINCIPAL

Nada más arrancar Krowdix, nos encontramos con el gestor principal, con diversas zonas. Desde esta pantalla podemos acceder al asistente para crear redes y al gestor de perfiles. Además, podemos cargar redes, borrarlas, arrancar nuevas simulaciones o detenerlas, y abrir las visualizaciones.

1) Zona de gestión de redes sociales. En esta zona tenemos una lista desplegable con las redes creadas disponibles para ser cargadas. Es imprescindible cargar una red para poder reanudar o comenzar su simulación. Para cargarla, la seleccionaremos desde la lista desplegable, y pulsaremos el botón “Cargar red social”. Una vez hecho esto, sus datos aparecerán en la zona 2), de la que hablaremos más tarde. Si lo que queremos es borrar una red social, el proceso es parecido. Se selecciona en la lista desplegable y se pulsa el botón “Borrar red social”. No es posible borrar la red social cargada, por lo que tendremos que cargar otra red antes. Por último, si deseamos crear una nueva red social, escribiremos su nombre en el campo de texto, habilitando así el botón “Crear red social”, que pulsaremos a continuación. De esta manera se abrirá el asistente de creación de una red social nueva. Explicaremos este asistente más adelante.

2) Zona de información y simulación. En esta zona tenemos un botón “Perfiles” para abrir el gestor de perfiles, desde el cual podremos crear, modificar y borrar los perfiles del sistema. Debajo tenemos información relativa a la red social cargada, como su nombre, instante de ejecución y número de usuarios. En caso de haber alguna red social cargada, también aparecerán disponibles un par de botones “Arrancar” y “Detener” para manejar la simulación de la red. Para comenzar la simulación o reanudarla, pulsaremos “Arrancar”, quedando este botón deshabilitado hasta que se detenga mediante el botón “Detener”.

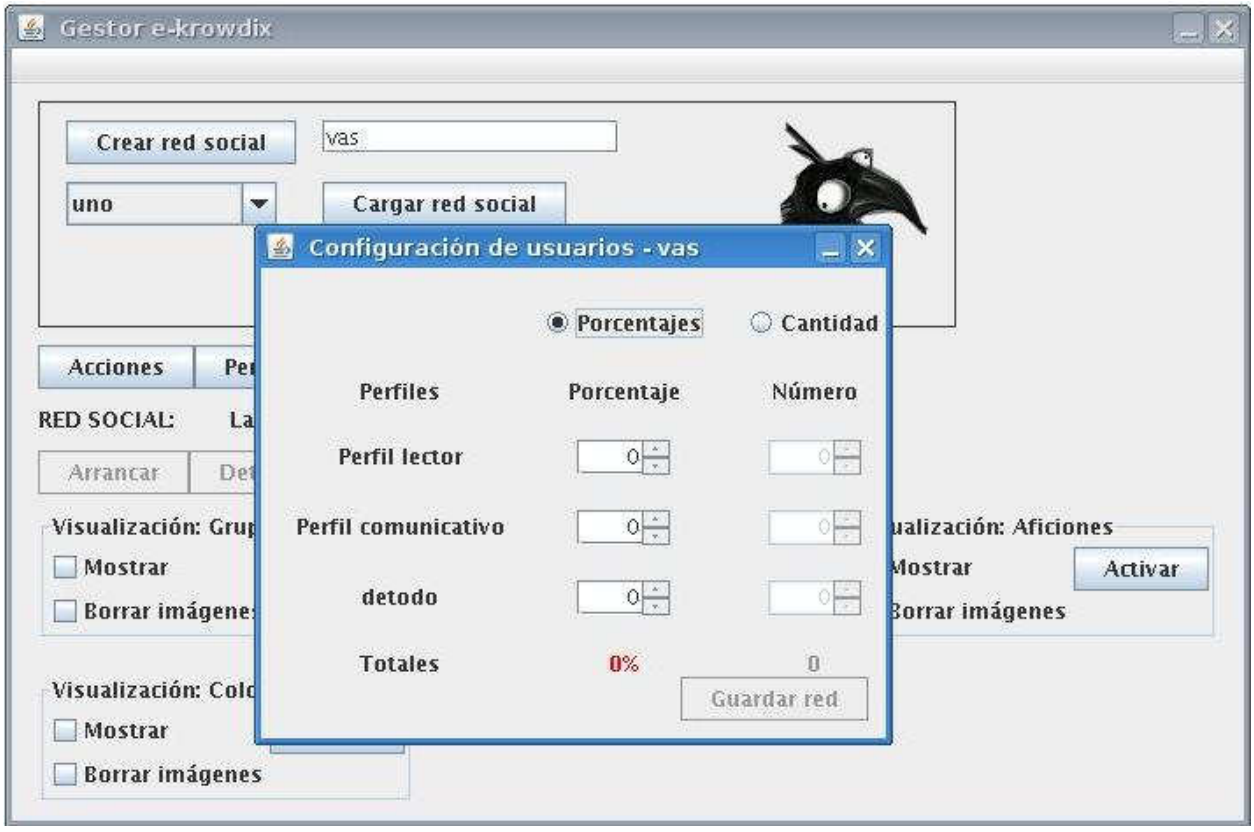
3) Esta zona es la de las visualizaciones. Aquí tendremos un grupo de controles por cada visualización. Por el momento sólo podremos tener abierta una visualización, aunque esto se mejorará en próximas versiones. Las visualizaciones se pueden abrir tanto si se está simulando como si no. En cualquier caso, debería haberse simulado al menos un instante para que los datos mostrados tengan sentido.

Para abrir una visualización, hay que marcar la casilla “Mostrar” y pulsar sobre el botón “Activar”. La visualización se abrirá en una ventana aparte, y comenzará a grabar vídeo con lo que ocurra en la visualización, pero es necesario tener la ventana en primer plano. Para cerrar la visualización, bastará con cerrar la ventana, o bien pulsar de nuevo sobre el botón “Activar”.

7.4 CREACIÓN DE RED SOCIAL

Mediante este asistente podremos configurar una nueva red social, indicando el número inicial de usuarios y sus perfiles. Está dividido en dos pasos. En el primero de ellos seleccionaremos el número de usuarios y los perfiles que usarán. En el segundo paso distribuiremos los perfiles seleccionados en el paso anterior sobre los usuarios.
Para abrir este asistente, en el gestor principal escribiremos un nombre de red en el campo correspondiente y pulsaremos el botón “Crear red”.

En el primer paso, como podemos ver, escribiremos en el campo de texto el número de usuarios inicial de la red social. Debajo tenemos dos áreas, la de perfiles disponibles y la de perfiles seleccionados para nuestra red. Al principio, la última está vacía. En este paso seleccionaremos en el área de perfiles disponibles los perfiles que deseamos usar, y los arrastraremos hasta el área de perfiles seleccionados. Una vez tengamos todos los perfiles que nos interesan, pulsaremos el botón “Siguiente” para acceder al segundo y último paso del asistente.

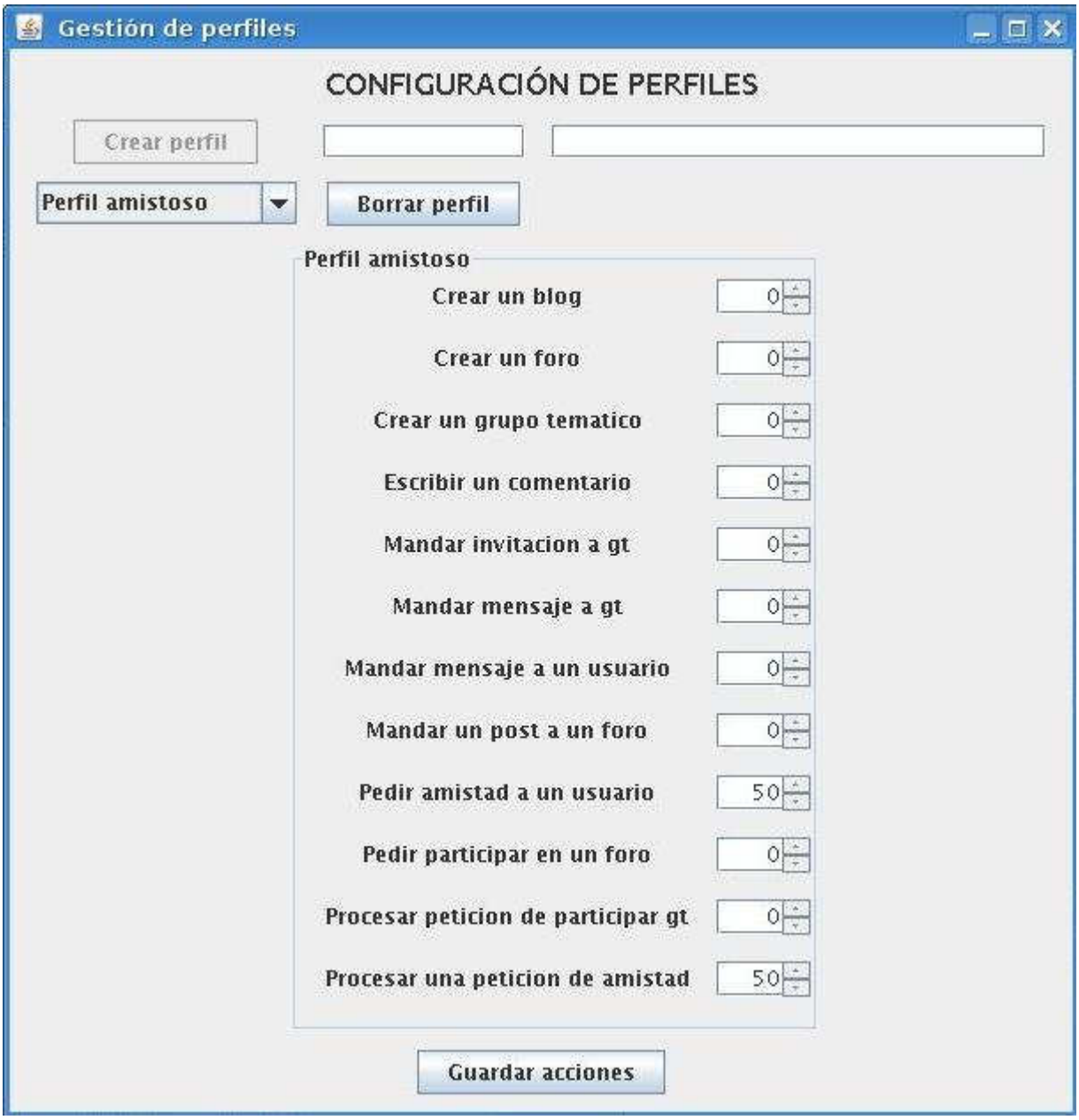


En este segundo paso, distribuiremos los perfiles entre todos los usuarios. Podemos hacerlo de dos formas: por porcentajes o por cantidades absolutas. Si seleccionamos "Porcentajes", para cada perfil tendremos que indicar el porcentaje de usuarios sobre el total que tendrán este perfil. Para dar esta configuración como válida, la suma de los porcentajes, que aparece en la parte inferior, debe sumar 100%. En ese caso, el total aparecerá verde y se habilitará el botón "Guardar red". Por el contrario, si escogemos el modo "Cantidad", para cada perfil tendremos que indicar el número absoluto de usuarios iniciales que tendrán este tipo de perfil. Evidentemente, este número debe ser inferior o igual al total de usuarios configurado en el paso anterior. Además, para validar estos datos, la suma de usuarios debe ser igual al total escogido en el paso 1 del asistente. En este caso, el total será de color verde y se habilitará el botón "Guardar red".

Una vez tengamos todos los datos validados, pulsaremos el botón "Guardar red" y el sistema se actualizará con la nueva red, cerrándose el asistente y regresando al gestor principal de la aplicación. A partir de entonces, dispondremos de nuestra nueva red en la lista desplegable de redes, al final.

7.5 GESTIÓN DE PERFILES

Krowdix lleva incorporado un gestor de perfiles. Para abrirlo, hay que pulsar el botón "Perfiles" del gestor principal. Este gestor tiene dos partes: la parte de gestión de perfiles en sí, y la parte de modificación de porcentajes.



En la parte de gestión de perfiles, podemos crear y borrar perfiles. Tenemos un campo corto de texto para el nombre de un perfil nuevo, y otro campo más largo para una descripción. Una vez que tengamos el texto escrito, debemos pulsar sobre el botón “Crear perfil”, que se encontrará habilitado. Tras esto, el nuevo perfil se incorporará a la lista desplegable junto con el resto de perfiles existentes. Ahora queda asociarle acciones. Para ello, lo seleccionamos en la lista desplegable. Debajo de esto, tenemos todas las acciones existentes, junto a un porcentaje. Este es el porcentaje de la acción dentro del perfil. Mediante los controles del porcentaje podemos ajustar y ponderar la importancia de cada acción en el perfil. Debemos asegurarnos que la suma de los porcentajes es 100. Una vez tengamos el perfil correctamente configurado, pulsamos sobre el botón “Guardar acciones”.
También podemos cambiar la configuración de un perfil anterior, seleccionándolo en la lista desplegable y siguiendo el mismo método explicado anteriormente.

7.6 VISUALIZACIONES

Las visualizaciones nos permiten tener una visión gráfica de determinados comportamientos de la red. Esto ayuda a comprender la evolución de la red a lo largo del tiempo, dándonos una visión global sencilla de entender.
En Krowdix, cada visualización utiliza una ventana externa propia, de manera que no interfiere con el gestor principal. Además, cada vez que abrimos una visualización, se comienza a grabar un vídeo que termina al cerrar la visualización. Esto nos ayudará a hacer análisis posteriores de la ejecución de la simulación, permitiendo también comparar vídeos de distintas visualizaciones.
Veamos una descripción de cada visualización:

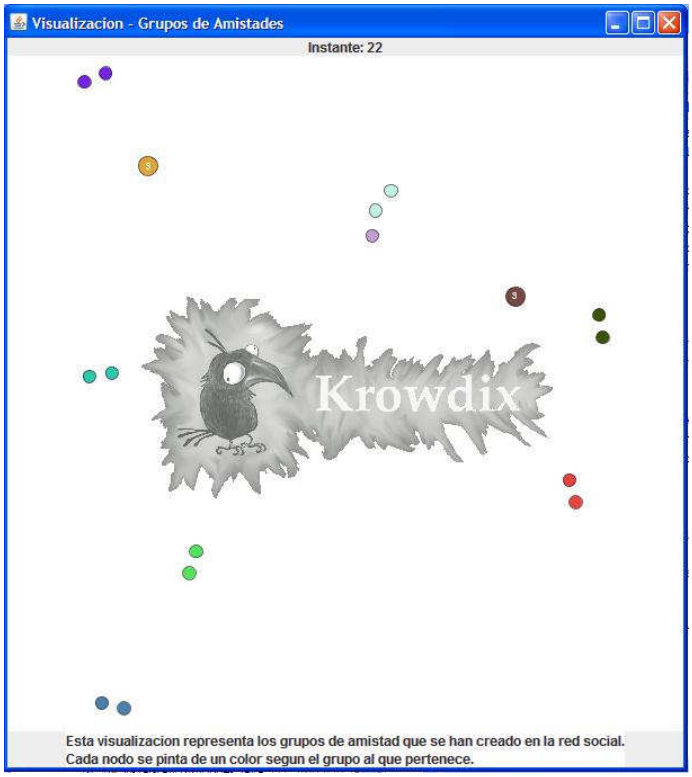
7.6.1 Grupos de amistad

En esta visualización se distinguen los grupos de amistades que se han creado en la red social. Cada grupo de amistad esta pintado con un color distinto. De forma que se pueden apreciar fácilmente con un vistazo. Cada punto pequeño representa un usuario de la red social, cuando se forma un grupo del mismo color (grupo afín), de más de tres usuarios, estos puntos se fusionan en uno con el número de integrantes escrito en el centro. Conforme más usuarios tiene un grupo más grande será el punto que lo representa.

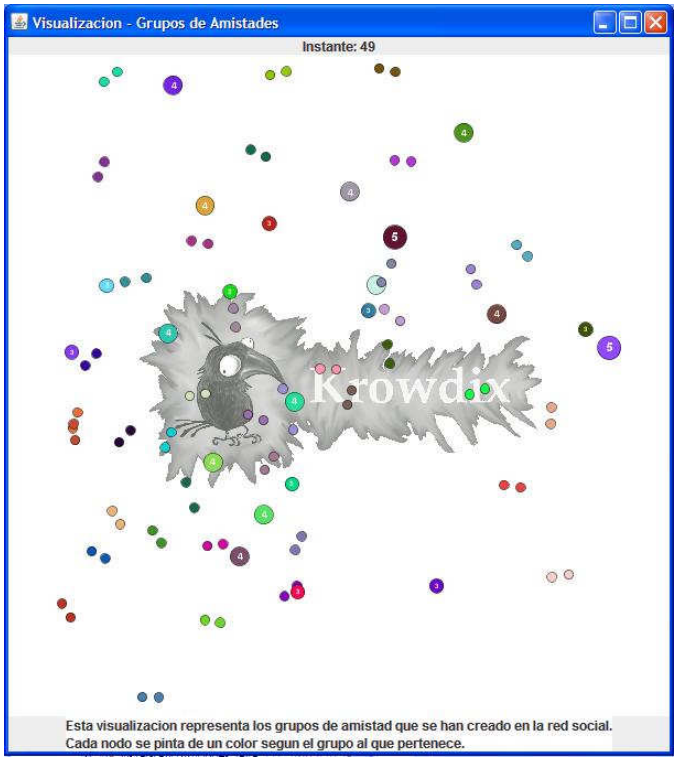
Cada grupo que se crea en la visualización son grupos afines, es decir, existen una serie de relaciones de amistad que entrelazan a los usuarios englobados en el grupo, formando un grupo de amistad diferenciable en la red social.

Los puntos se disponen en la ventana de forma que dos puntos que aparezcan juntos en el espacio, son susceptibles de crear un grupo más grande fusionándose entre sí. Los grupos que aparecen en la periferia de la ventana son grupos más aislados de la red social.

En la siguiente imagen podemos apreciar los primeros pasos de la red social, con pocos usuarios, que han formado grupos la mayoría de dos integrantes y ya se han empezado a formar grupos de tres integrantes. Podemos apreciar un grupo de dos de color azul celeste, que tiene pegado un usuario de color morado. La relación de amistad entre los usuarios de color azul es fuerte pero el usuario de color morado tiene una alta probabilidad de unirse al grupo ya que aparece pegado a él.



En la siguiente imagen podemos apreciar cómo ha crecido en número de usuarios la red social y cómo se han creados subgrupos de amistad fuertes de cuatro o cinco integrantes. Así como sigue habiendo bastantes grupos de dos usuarios que se han incorporado recientemente a la red social y aún no se han incorporado a ningún grupo de amistad.



En la última imagen de esta visualización vemos como se ha creado un grupo

diferenciado en la red social, después de éste se han creado otros grupos también numerosos. Estos grupos son los “grupos fuertes” de la red social, esto es, grupos de usuarios muy interconectados entre sí. Como podemos apreciar también hay usuarios individuales, estos es porque aún no tienen ninguna relación con ninguno de los grupos, o si las tienen no son suficientes para incluirlos dentro de esos grupos.

En conclusión, en esta visualización diferenciamos los grupos de amistad de la red social y vamos agrupando a sus usuarios para poder ver de forma fácil los distintos grupos creados. Nos quedamos con que los grupos formados tienen suficientes relaciones de amistad entre sí para crear un grupo de amistad fuerte, y los usuarios que aparecen cerca tienen vínculos con el grupo, pero todavía no son suficientes para incluirlos dentro de él.

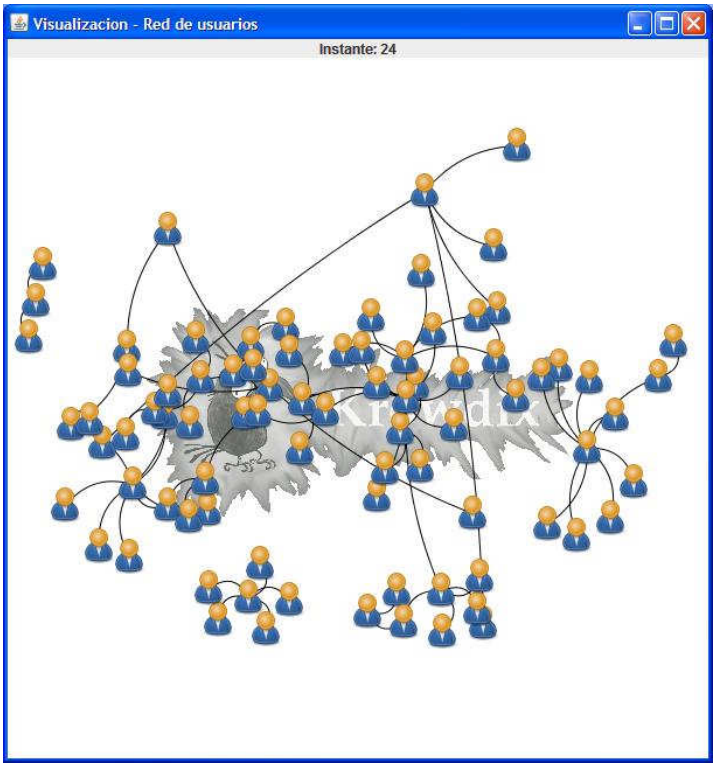
7.6.2 Red de usuarios

Mediante esta visualización se intenta representar las relaciones de amistad de los usuarios. Para ello se ha escogido una representación en forma de grafo no dirigido. Los nodos, de color rojo, representan los usuarios con alguna amistad, mientras que las aristas, de color negro, unen usuarios que comparten una relación de amistad entre sí.

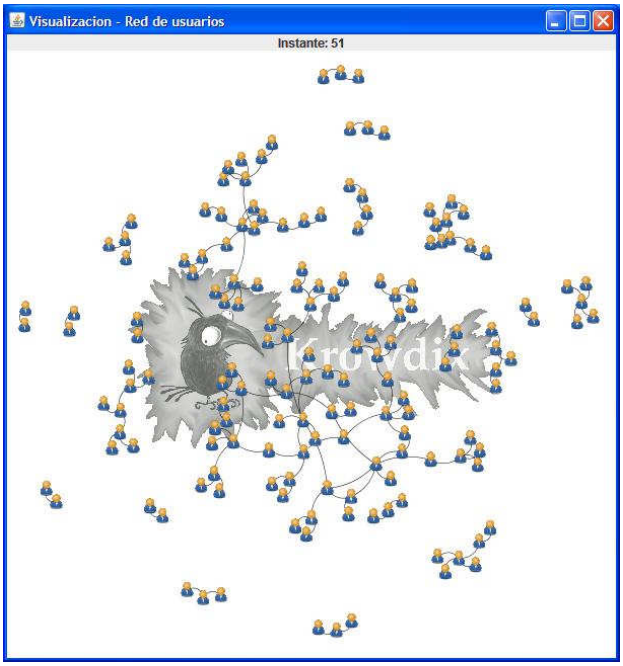
Esta visualización es muy útil para ver cómo van creciendo las relaciones de amistad, y cómo hay algunos nodos que se quedan en la periferia por no estar tan conectados como los nodos centrales. Esto significa que estos usuarios apenas tienen relaciones de amistad. También se puede observar cómo se forman pequeñas componentes

conexas en la periferia, desconectadas del resto del grafo. Estas componentes se corresponden con usuarios que han formado amistades entre sí pero que no tienen relaciones con el resto de usuarios.

Para facilitar la comprensión de esta visualización, se omiten los usuarios sin relaciones de amistad. Esto da lugar a que en los primeros instantes, la imagen pueda aparecer vacía, dando la sensación de que no existen usuarios. En la siguiente imagen podemos ver la red en un instante temprano. Por la imagen podríamos pensar que estamos en una red de apenas 23 usuarios aproximadamente. Sin embargo, esto es por lo anteriormente explicado. La red, en realidad, tiene cerca de 100 usuarios, pero aún hay pocas amistades.

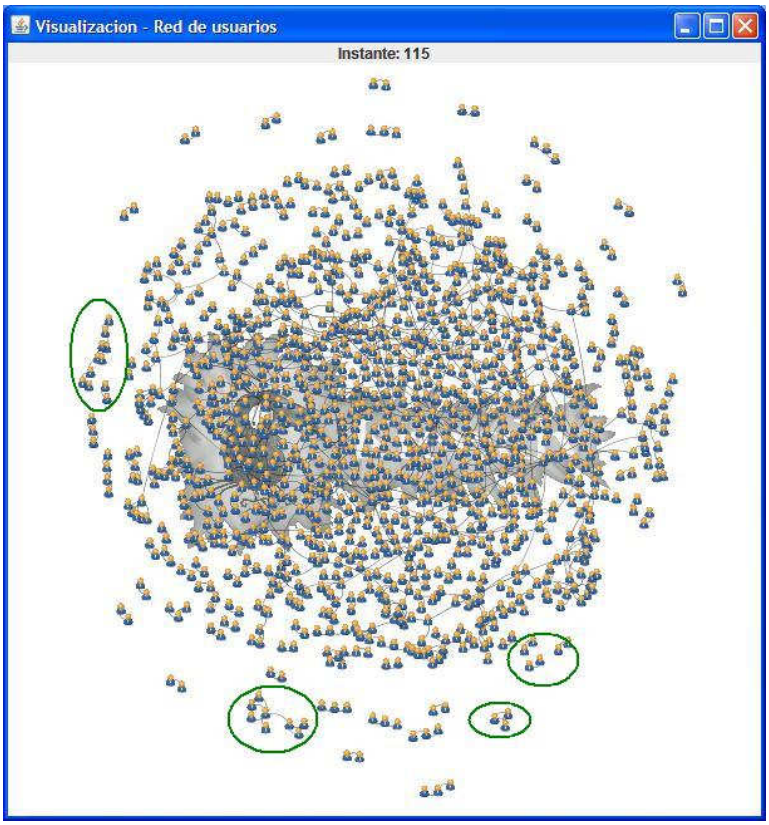


En la siguiente imagen, sin embargo, ya ha pasado algo de tiempo y podemos apreciar que hay muchas más relaciones. Además, podemos ver como los nodos más conectados se han ido situando en la parte central del grafo, mientras que los nodos con menos amistades están en la periferia.



Podríamos incluso sacar un patrón en anillos concéntricos, donde los anillos centrales corresponderían a los nodos más conectados, mientras que los anillos periféricos serían los menos conectados.

Veamos ahora otra imagen con la disposición final de la red:

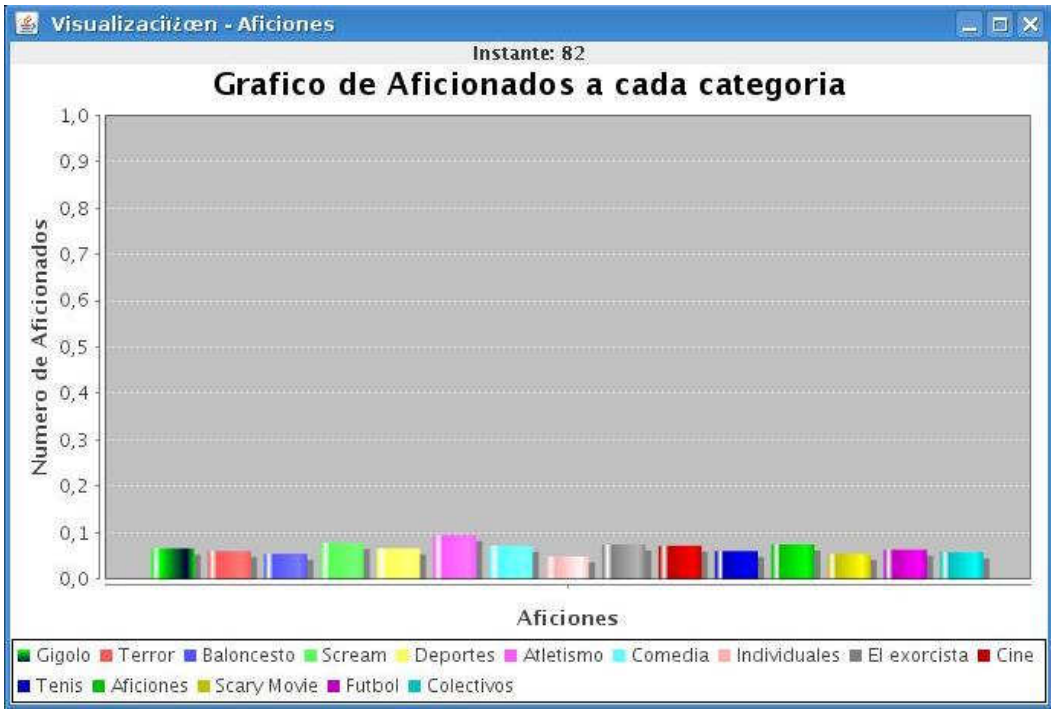


Aquí podemos seguir viendo cómo los nodos más conectados ocupan las posiciones centrales del grafo. Pero además, en esta imagen podemos apreciar que aparecen las

componentes conexas de las que hablamos anteriormente, correspondientes a usuarios relacionados entre sí pero aislados del resto de la red, formando subredes. Algunos de ellos están rodeados de un círculo verde para su mejor localización.

7.6.3 Aficiones

Esta visualización pretende mostrar la proporción de usuarios que hay de cada afición, con respecto del total de usuarios. Es una medida normalizada, por tanto, y que se muestra en forma de diagrama de barras donde cada barra representa una afición, y cuya altura determina el porcentaje de usuarios con esa afición, con respecto al total de usuarios.



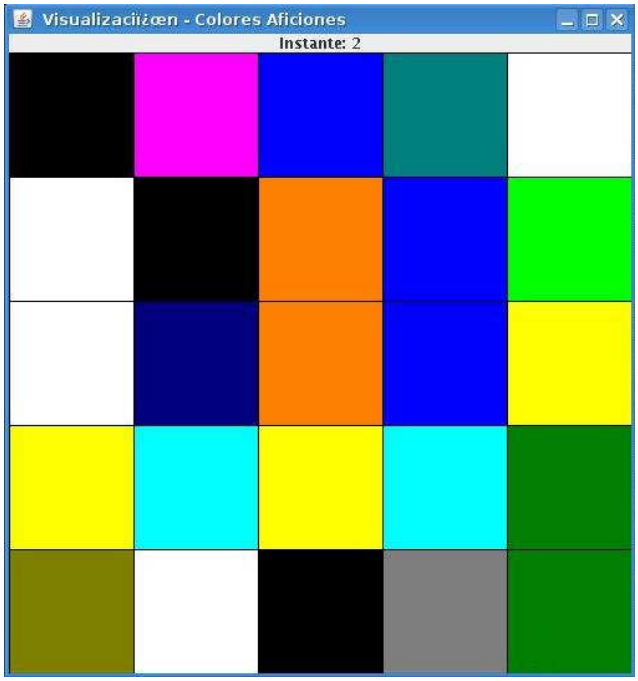
Podemos ver en la imagen que la distribución de aficiones es bastante equilibrada, teniendo más usuarios el atletismo. En futuras versiones, en las que se usarán mucho más las aficiones, veremos mayores diferencias en esta visualización.

7.6.4 Colores aficiones

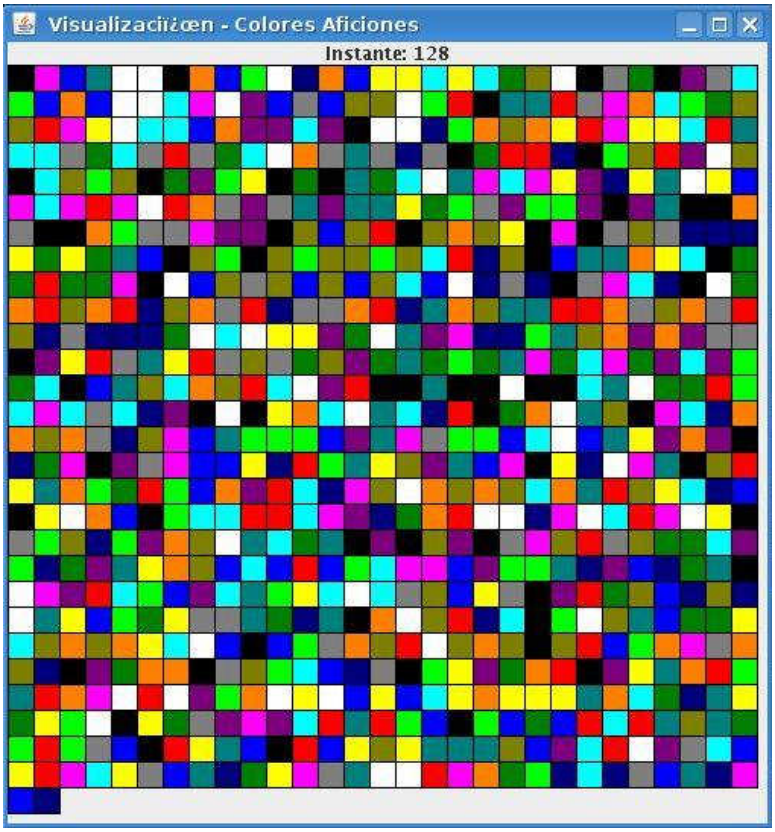
Esta visualización es similar a la anterior. Se trata de mostrar un cuadrado por cada usuario, coloreado de un color según el perfil del usuario al que representa, formando un mapa de usuarios. Conforme va creciendo el número de usuarios, el mapa de usuarios se va redimensionando para dar cabida a todos los usuarios de la red.

En la siguiente imagen podemos ver la visualización en un instante temprano de simulación de la red, con 20 usuarios. Como podemos ver, los cuadros son del tamaño adecuado para que la visualización ocupe toda la ventana.
Actualmente, el orden de los cuadros no tiene importancia, pero se están estudiando

mejoras que fijarían a cada usuario en un punto del espacio, de forma que el patrón de colores no cambiase demasiado.



En la siguiente imagen podemos ver la misma red en un estado mucho más avanzado. Comparando ambas imágenes podemos ver que ahora, los cuadros son mucho más pequeños para poder situar a todos los usuarios en el mismo espacio.



7.7 CONFIGURACIÓN

En Krowdix hay algunos aspectos que son configurables. Para ello, se usa un fichero JSON situado en el directorio principal *krowdix* de nuestra aplicación, llamado krowdix.json. Este fichero está compuesto por pares “clave”: valor. Para más detalles sobre la sintaxis, consultar www.json.org.
Pasemos a explicar las parejas que intervienen en la configuración del programa:

"tamano array acciones": El número de acciones que contendrá el array de acciones de cada usuario. Cuánto más pequeño sea este número, menos cantidad de memoria necesitará, pero más veces habrá que construir el array, lo que conllevará más lentitud al ejecutar los instantes de tiempo. Un buen valor es el de 100.

"puntos por instante": Cada instante los usuarios reciben una cantidad de puntos a gastar ejecutando acciones. Cada acción lleva un coste de puntos determinada. Este parámetro es esa cantidad de puntos que reciben los usuarios. Cuanto más grande sea, más acciones ejecutarán los usuarios en un solo instante, y más largos serán estos. Un buen valor puede ser el de 15.

"tamano pool threads": Krowdix cuenta con un pool de threads creados. Esos threads se usan para ejecutar los diversos instantes de cada usuario. Cuanto más threads tenga este pool, menor latencia tendrán los instantes, a costa de un mayor consumo de memoria. Un buen valor para este parámetro puede ser el de 100. Se recomienda encarecidamente no modificar este parámetro si no se sabe bien lo que se está haciendo.

"tamano cola tareas": El pool de threads destinados a ejecutar instantes tiene también una lista de tareas pendientes por si no quedan threads libres. Este parámetro indica el tamaño de esta lista. Este parámetro depende mucho del anterior, e igualmente es mejor saber bien lo que se hace antes de modificarlo, porque puede afectar al rendimiento del programa. Un valor aceptable para este parámetro puede ser 500.

"niveles vuelta atras": Este parámetro define el número de niveles que se pueden retroceder en el árbol de aficiones para definir la afinidad entre aficiones. Debe ser un valor positivo o 0. Un valor aceptable para este parámetro es 2, aunque depende mucho de cómo definamos el árbol de aficiones, del que hablaremos más tarde.

"borrar imagenes": Para generar los vídeos de las visualizaciones, primero se captura una imagen por cada instante, y al cerrar la visualización se forma el vídeo con ellas. Este parámetro define si se quieren borrar o no las imágenes. Un 1 significará que se van a borrar, mientras que un 0 significará que se mantendrán en el directorio.

"numero colas": Para las escrituras en base de datos se usa un sistema de colas de espera. Este parámetro define cuántas colas se crearán para este propósito. Es un parámetro interno y no se debería tocar a menos que se sepa lo que se está haciendo. Un buen valor para este parámetro es el de 25.

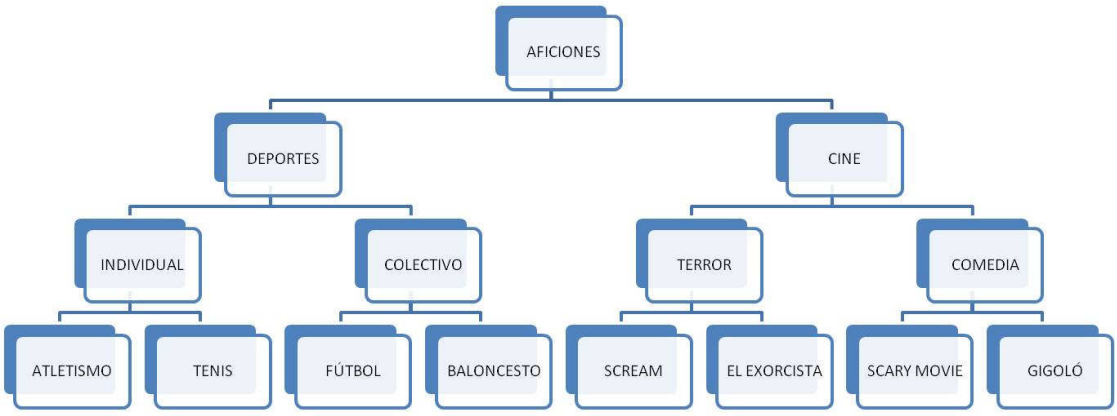
Se está estudiando cambiar el formato de este fichero a un xml para facilitar nuevos parámetros, así como parámetros anidados, y su modificación a través de la interfaz

gráfica.

7.8 AFICIONES

Cada usuario tiene una afición que se define en el momento de su creación de forma aleatoria. Estas aficiones son configurables mediante otro fichero json llamado ontologia.json, situado en el mismo directorio que el fichero de configuración.
Para modificar las aficiones es imprescindible conocer bien la estructura que debe tener este fichero, y que detallaremos a continuación:

Se trata de un fichero jerárquico que representa un árbol binario. Cada nodo representa una afición, y está compuesto por la clave "nodo" donde se indica el nombre de la afición, y las claves "HijoIzq" y "HijoDcho", que contienen los hijos izquierdo y derecho respectivamente, que serán otros nodos representando aficiones. Cada nodo debe estar encerrado entre llaves {}. Se puede ver el archivo que se proporciona a modo de ejemplo, y ampliarlo o reducirlo a discreción, pero es imprescindible que el nodo raíz tenga como nombre de nodo "Aficiones".



8 CONTINUACIÓN DEL DESARROLLO

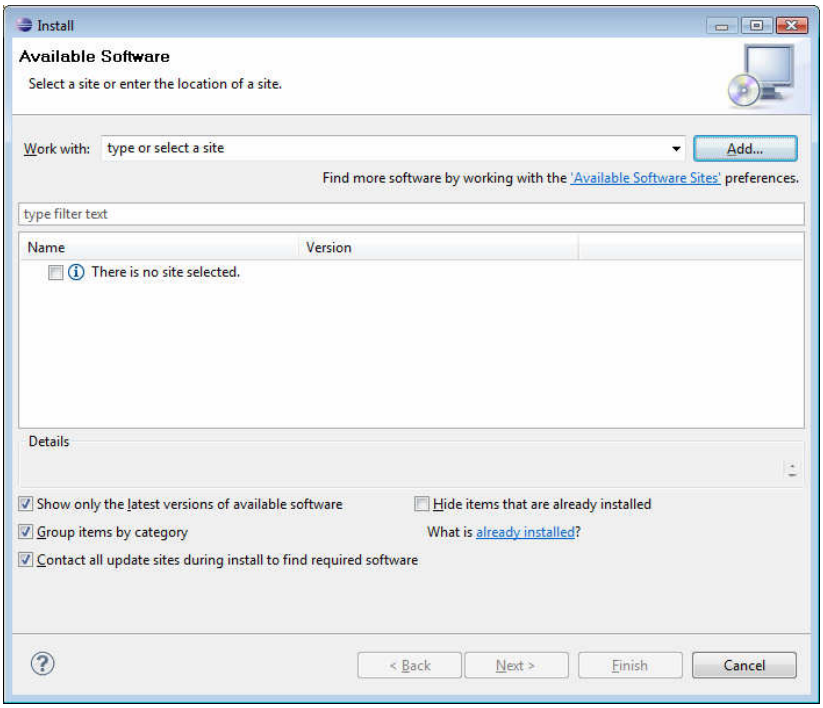
Para seguir con el desarrollo de Krowdix, primero debemos preparar el entorno de programación, para poder descargar el código con ayuda de la herramienta SVN que nos proporciona SourceForge. En los siguientes apartados procederemos a la explicación del proceso de montaje y preparación del entorno, así como la descarga del código, librerías necesarias y el proceso de compilación.

8.1 Eclipse como entorno elegido

Nosotros hemos elegido Eclipse por ser multiplataforma, gratuito y sencillo de manejar. Para instalar eclipse, nos descargaremos el IDE desde su página oficial de descargas http://www.eclipse.org/downloads/. Allí hay varias versiones de eclipse, según nuestras necesidades. Krowdix es un programa escrito íntegramente en Java, por lo que nos bastará con la versión Eclipse IDE for Java developers.
La descarga consiste en un fichero comprimido con el programa, que deberemos descomprimir allá donde queramos tener el entorno. Por tanto, no tiene instaladores.

8.1.1 Plugin para SVN

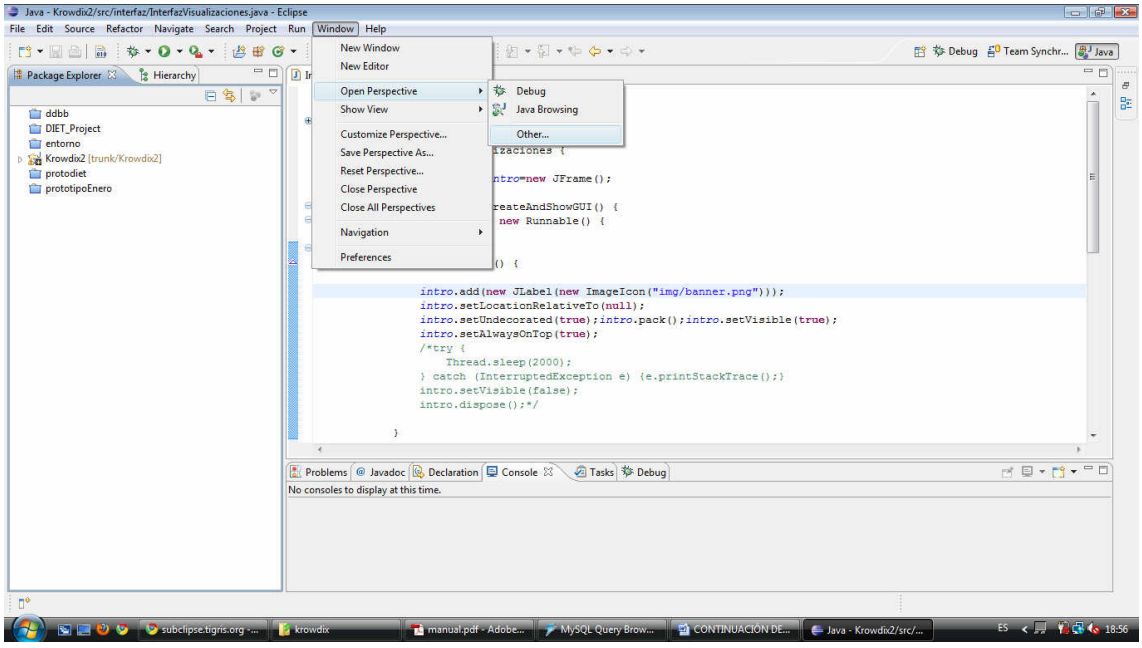
Para dotar a Eclipse de la capacidad para poder descargar y gestionar proyectos desde un repositorio SVN, necesitamos instalar algún plugin que nos proporcione cliente SVN con el que llevar a cabo todo esto. Nosotros explicaremos cómo hacerlo con el plugin Subclipse, cuya dirección es http://subclipse.tigris.org/. Lo primero que debemos hacer es abrir nuestro Eclipse recién instalado. A continuación, pulsaremos sobre el menú *"help"* y elegiremos la opción "*install new software…*". Obtendremos una ventana como esta:



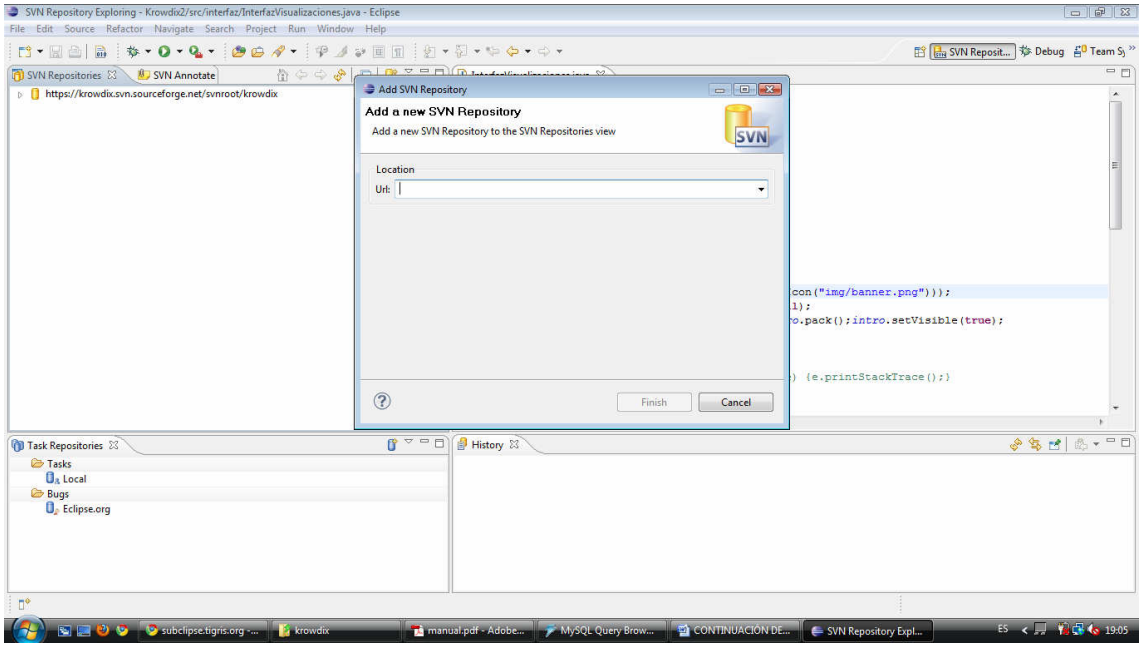
A continuación, pulsaremos el botón *"Add"* para añadir la dirección de descarga del plugin. En el cuadro de diálogo, pondremos en *"location"* la dirección http://subclipse.tigris.org/update_1.6.x y en name lo que queramos nosotros para identificar el plugin. Al pulsar *"OK"* el sistema de instalación cargará en la ventana central los componentes del plugin. Seleccionaremos todo para instalar, tal y como aparece en la siguiente imagen, y pulsamos *"Next"*. Nos saldrá el cuadro de diálogo "*install details*". Pulsamos nuevamente *"Next"*, aceptamos la licencia y pulsamos *"Finish"*. En este momento comenzará la instalación del plugin. A lo largo de la instalación es posible que nos pida confirmación sobre la instalación de software sin firmar. Esto es completamente normal y tendremos que aceptar. Una vez terminada la instalación, el programa nos preguntará sobre el reinicio de eclipse. Pulsaremos sobre *"yes"*. Una vez reiniciemos eclipse, el plugin debería estar listo para usarse.

8.1.2 Preparando el repositorio

Ahora tenemos que añadir la dirección del repositorio SVN de Krowdix al sistema. Para ello, pulsaremos sobre el menú "*Window->open perspective->other…*":



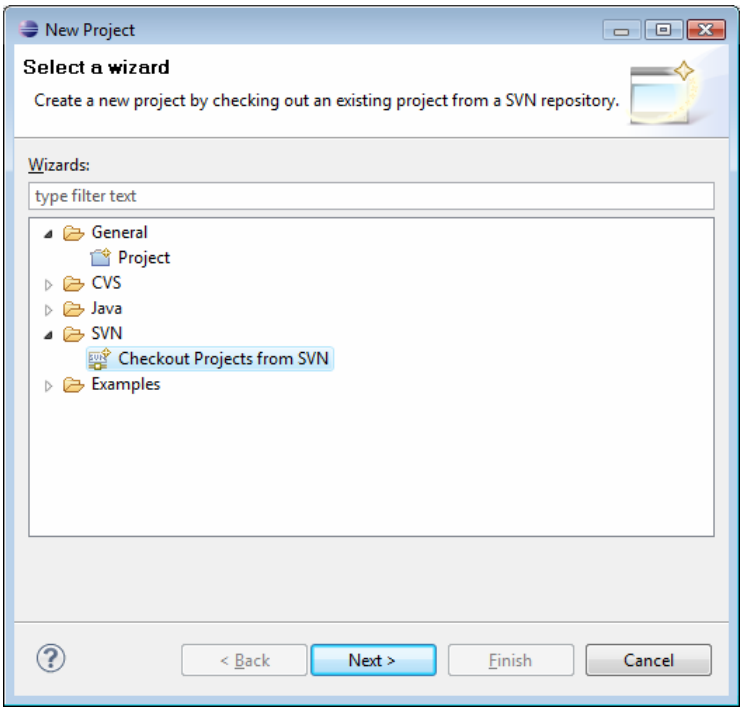
Una vez allí, elegiremos la perspectiva "*SVN Repository Exploring*". En la izquierda, dentro de la pestaña SVN Repositories, pulsaremos con el botón secundario del ratón, y elegiremos *"New ->Repository location…"*:



En el campo URL introduciremos la dirección del repositorio de Krowdix: https://krowdix.svn.sourceforge.net/svnroot/krowdix/trunk y pulsamos sobre "*finish*". Una vez tengamos el repositorio agregado, podemos crear el proyecto y bajarnos el código necesario.

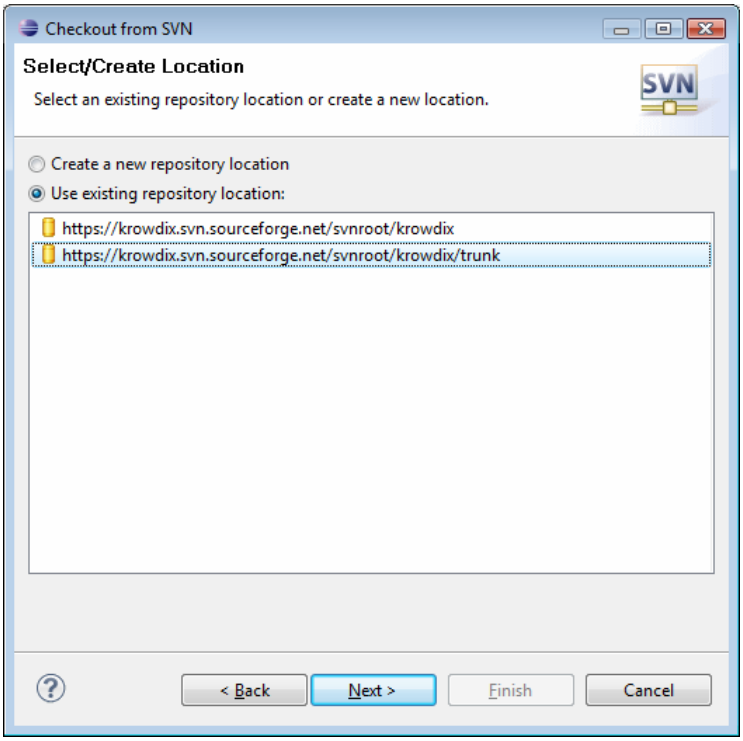
8.2 Crear el proyecto

Para crear el proyecto, tenemos que abrir la perspectiva de Java. En la parte de package explorer, pulsamos con el botón derecho del ratón, y elegimos "*new->Project…*".

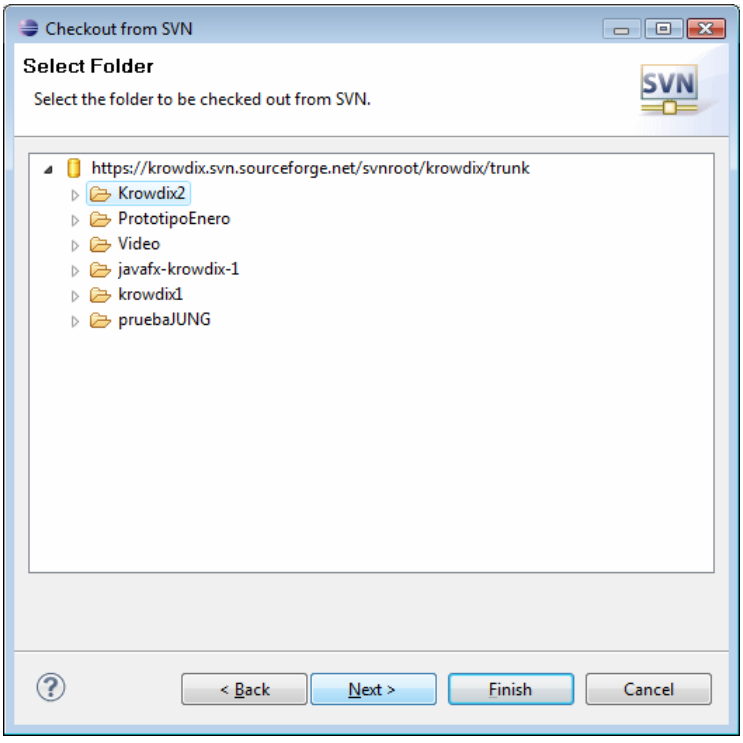


Elegimos el tipo *SVN->Checkout Projects from SVN* y pulsamos "*next*". Después

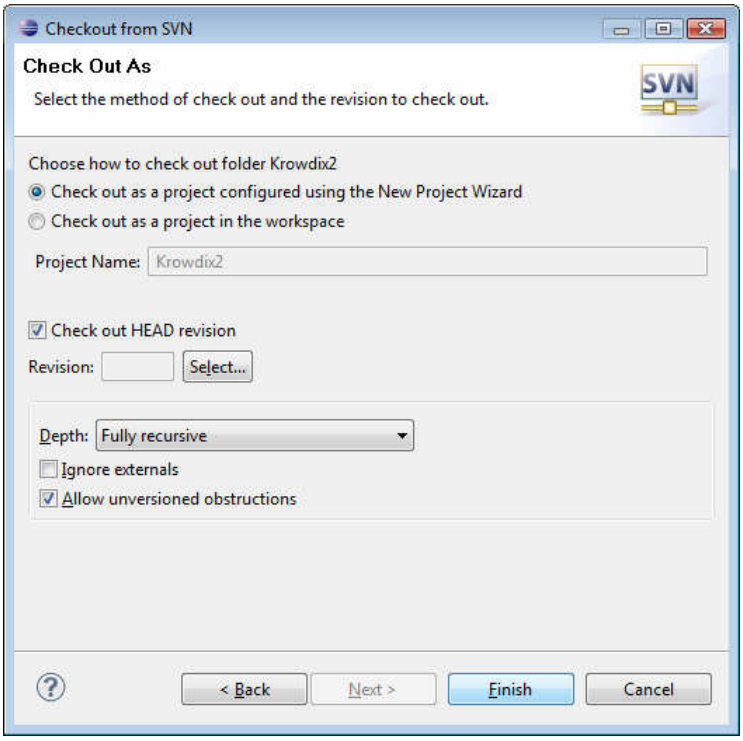
elegimos el repositorio que hemos agregado en el paso anterior.



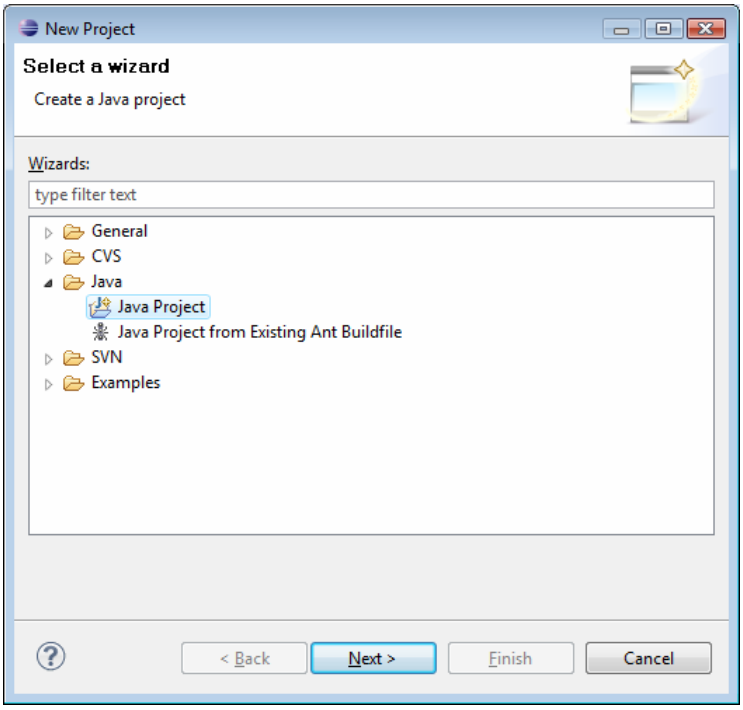
Después nos saldrá una lista de proyectos. El único que nos interesa es el que se llama Krowdix2. Lo seleccionamos y pulsamos "*next*".



Cuando estemos en la ventana de "*Checkout as*", deberemos seleccionar "*checkout as a Project configured using the new Project Wizard*" y pulsar "*Finish*".

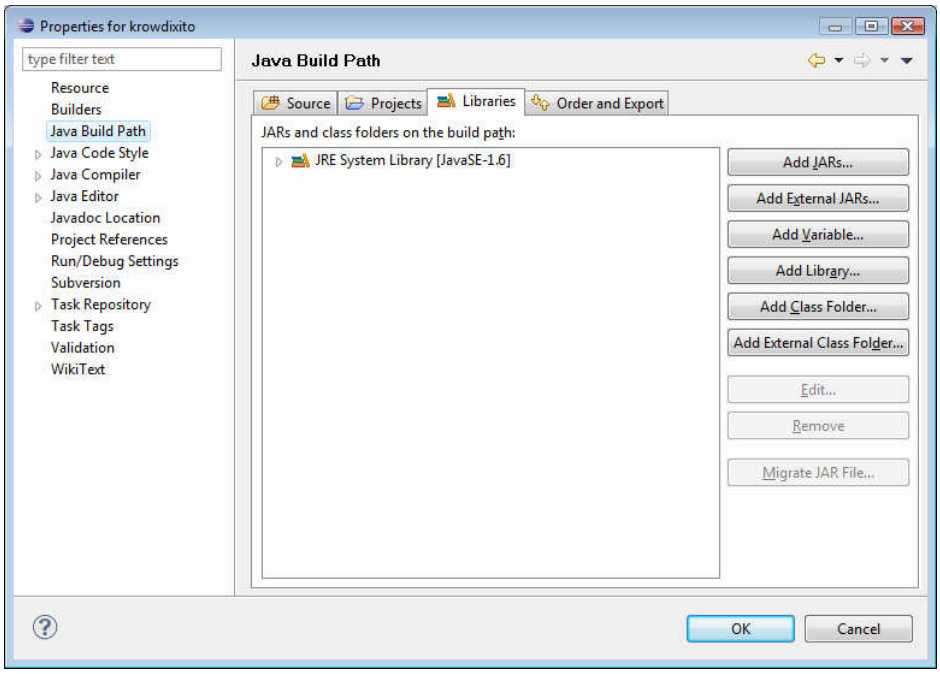


Entonces comenzará la descarga de la información del proyecto y mostrará de nuevo el cuadro de diálogo del asistente de creación de proyectos. Esta vez deberemos escoger proyecto java, tal y como se muestra en la imagen, y pulsar "*next*".

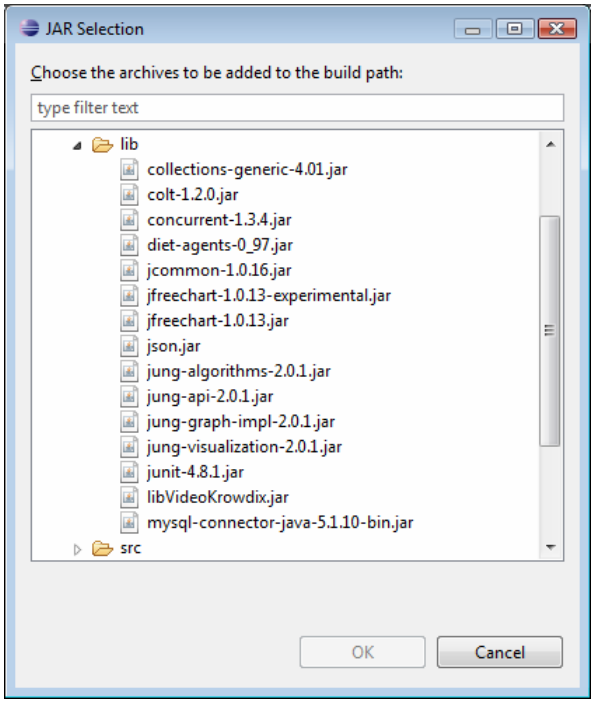


En el cuadro de diálogo de "*new java Project*", lo único a destacar es que podemos llamar al proyecto como prefiramos, ya que este nombre será solamente local. Una vez tengamos esto solucionado, pulsamos sobre "*finish*".
Es posible que en este momento, eclipse pida confirmación para terminar de bajar algunos recursos del repositorio, teniendo que aceptar para poder completar el

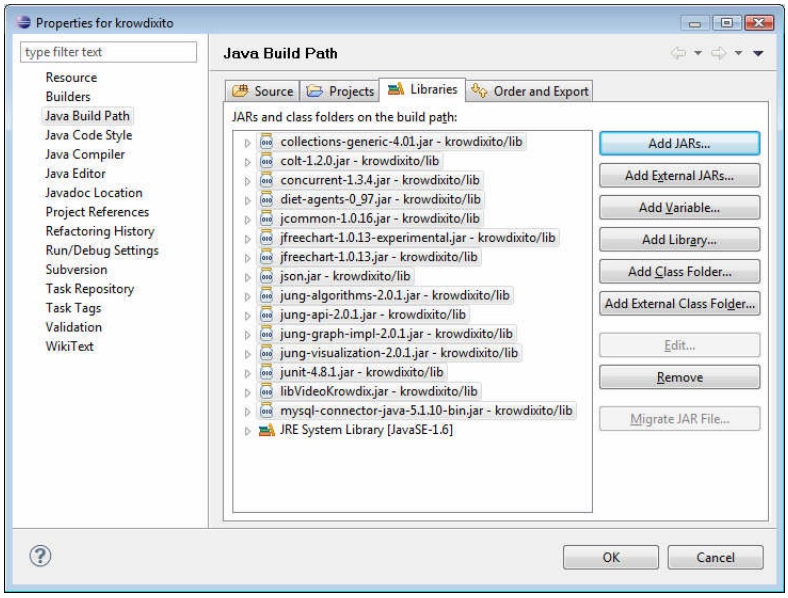
proceso.
Una vez hecho esto, sólo nos quedaría añadir las librerías necesarias. Para ello, seleccionamos el proyecto en la pestaña "*package explorer*" y pulsamos sobre él con el botón secundario, y elegimos la opción "*properties*". Seleccionamos "*Java build path*", y en la parte central, pestaña "*libraries*".



Pulsamos el botón "*Add JARs*" y desplegamos el directorio del proyecto y el directorio "*lib*" que hay dentro, quedando como se muestra:

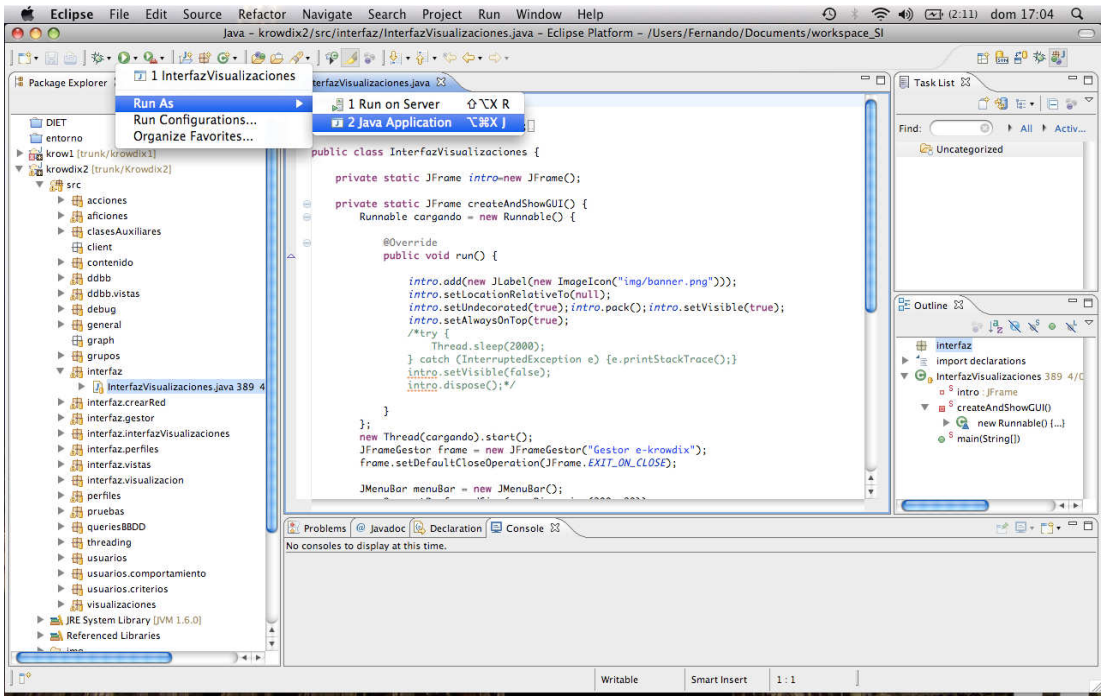


Seleccionaremos todas las librerías .jar que aparecen y pulsaremos "*OK*".



Una vez hecho esto, pulsaremos el botón *"OK"*.

Y ya tenemos el proyecto listo para usar. Simplemente abrimos la clase "*InterfazVisualizaciones*" que se encuentra en el paquete "*interfaz*" y pulsamos en "*run as->java application*". Hay que acordarse de que la aplicación accede a la base de datos MySQL así que hay asegurarse de que está arrancada y con la tablas necesarias creadas, véase el manual.



9 AMPLIACIÓN DE FUNCIONALIDAD

En los siguientes párrafos explicaremos cómo agregar nueva funcionalidad al programa, los pasos a seguir y cómo modificar el modelo de datos para ponerlo en funcionamiento. Para ello, usaremos ejemplos sencillos que iremos desarrollando paso a paso. Para llevar a cabo estos ejemplos, es imprescindible que se tenga el proyecto preparado para desarrollar, tal y como se explica en la sección anterior CONTINUACIÓN DEL DESARROLLO.

9.1 Agregar una nueva acción

Vamos a intentar agregar al programa una acción sencilla como puede ser la de subir una fotografía. Además, para que quede constancia de las fotos subidas por cada usuario, lo reflejaremos en la base de datos también.

Iremos de abajo a arriba. Primero pensaremos en cómo guardaremos la nueva información en base de datos. Como las fotografías serán comentables, añadimos una entrada nueva a la tabla *tiposComentables*. Una línea como esta puede bastar:

insert into tiposComentables (nombreComentable,descComentable) values ('Foto','Se puede comentar una fotografia.');

Vamos a crear también una tabla de fotografías, que relacione al usuario con la foto concreta que subió. Una tabla como esta puede bastarnos:

create table fotos (

idUser integer not null,

idFoto integer not null,

idComentable integer not null,

instante integer not null,

idRedSocial integer,

primary key (idFoto,idRedSocial),

foreign key (idUser,idRedSocial) references urs(idUser,idRedSocial),

foreign key (idComentable,idRedSocial) references comentables(idComentable,idRedSocial),

foreign key (idRedSocial) references redes(idRed)

)ENGINE = InnoDB;

Para facilitar las tareas de instalación y desinstalación, agregamos una línea al comienzo de los drops del script como esta:

drop table if exists fotos;

Ahora vamos a agregar la información necesaria para la acción en sí. Para ello, tenemos que añadir una nueva entrada en la tabla *acciones*, como esta:

insert into acciones(nombreAccion,descAccion,puntos,clase) values ('Subir foto','otraaccion',20,'AccionSubirFoto');

Sobre los parámetros, los dos primeros son un nombre y descripción de la acción, el parámetro numérico son los puntos asociados a la acción, y el último parámetro es el nombre que tendrá la clase responsable de la acción.

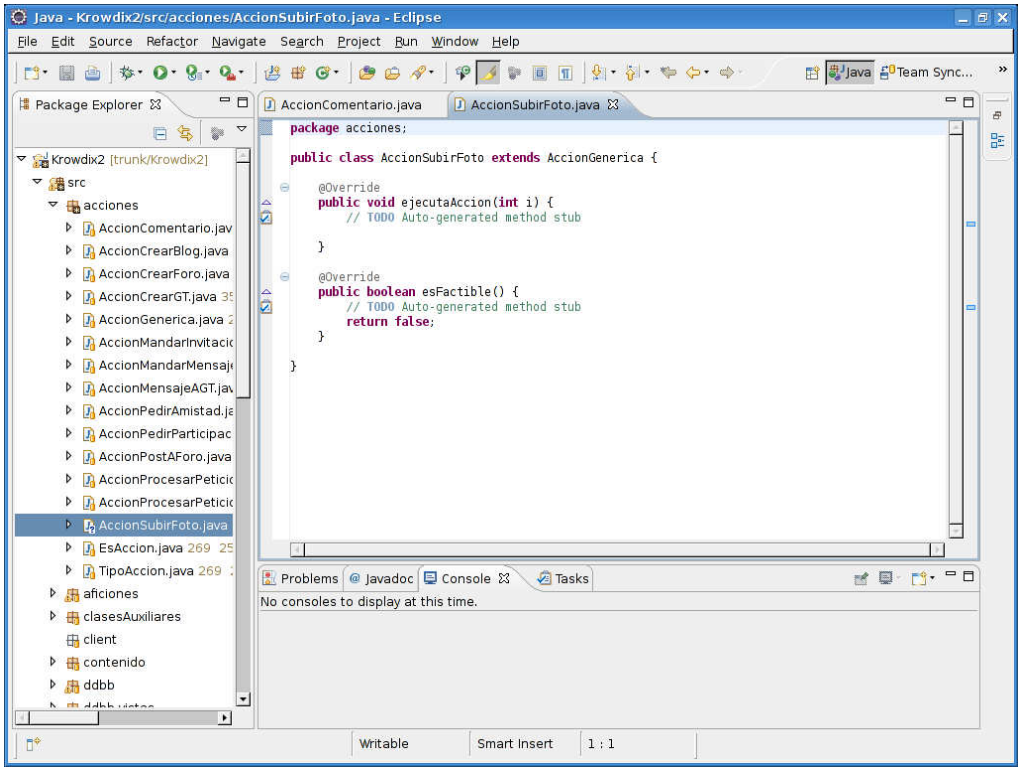
Con esto, deberíamos haber terminado el trabajo en base de datos; ya tenemos lo necesario para la nueva acción en el modelo de datos.

Ahora crearemos el DAO para gestionar esta tabla desde el código. Para ello, crearemos en el paquete *ddbb* una clase pública que se llame igual que la tabla, en nuestro caso, *fotos*. Esta clase tendrá las siguientes funciones:

protected int getMaxFoto(): Devolverá el id máximo de foto para la red actual.

Cada una de estas funciones debe tener su correspondiente función pública en GestorDB:

A continuación, crearemos la acción. Creamos una clase llamada *AccionSubirFoto* en el paquete *acciones*, que heredará de *AccionGenerica* situada en el mismo paquete. Debido a que la clase *AccionGenerica* implementa la interfaz *EsAccion*, deberemos añadir los métodos *ejecutaAccion* y *esFactible*, tal y como aparece en la imagen:



Agregaremos a nuestra acción una constructora pública vacía y un par de atributos estáticos: el primero, de tipo *TipoAccion* llamado *accion*, y el segundo, del tipo de la acción que estamos haciendo (en este caso, *AccionSubirFoto*), llamado *proto*. Este último lo iniciaremos con la constructora vacía.

Pero esto sólo hace parte de la inicialización. Para terminar esta, nos tenemos que ir a la clase *general.ExclusionMutua*. Lo primero que haremos aquí es agregar el contador que se encargará de gestionar los identificadores de fotos. Agregamos un nuevo atributo estático *AtomicInteger* a la clase, de la forma que muestra la imagen:

```
public class ExclusionMutua {
    private static AtomicInteger IDGRUPO;
    private static AtomicInteger IDCONTENIDO;
    private static AtomicInteger IDCOMENTABLE;
    private static AtomicInteger IDUSER;
    private static AtomicInteger IDFOTO;
    private static AtomicInteger INSTANTE;
    private static boolean EJECUTAR=false;
    private static boolean SATURACION=false;
```

Tenemos que inicializarlo en la función *inicializarIdentificadores*:

```
public static void inicializarIdentificadores() {
    IDGRUPO=new AtomicInteger(1);
    IDCONTENIDO=new AtomicInteger(1);
    IDCOMENTABLE=new AtomicInteger(1);
    IDUSER=new AtomicInteger(1);
    IDFOTO=new AtomicInteger(1);
    INSTANTE=new AtomicInteger(0);
}
```

Lo inicializamos a uno porque hemos definido este campo como not null en la tabla. Creamos las funciones necesarias para gestionar el contador:

```
public static int consumeIdFoto() {
    return IDFOTO.getAndIncrement();
}
public static void seteaIdFoto(int i) {
    IDFOTO.set(i);
}
public static int getIdFoto() {
    return IDFOTO.intValue();
}
```

En la función *cargarRedSocial*, que es la encargada de cargar una red social ya creada desde base de datos, pedimos mediante las funciones agregadas antes a *GestorDB* el identificador de la última foto agregada, para inicializar el contador desde la siguiente:

```
public static void cargarRedSocial(int i) {
    for (RedSocial rs:redes)
        if (rs.getIdRed()==i) redActiva=rs;

    INSTANTE.set(redActiva.getInstante());

    GestorDB db=new GestorDB();
    ArrayList<URS> usuarios=db.getUsuarios();
    if (usuarios.size()>0)
        seteaIdUsuario((usuarios.get((usuarios.size()-1)).getID()+1));

    else seteaIdUsuario(1);
    int maxC=db.getMaxContenido();
    if (maxC>0)
        seteaIdContenidos(maxC+1);
    else seteaIdContenidos(1);
    int maxG=db.getMaxNumGrupo();
    if (maxG>0)
        seteaIdGrupos(maxG+1);
    else seteaIdGrupos(1);
    int maxCom=db.getMaxComentable();
    if (maxCom>0)
        seteaIdComentable(maxCom+1);
    else seteaIdComentable(1);
    int maxIdFoto=db.getMaxFoto();

    if (maxIdFoto>0) seteaIdFoto(maxIdFoto+1);
    else seteaIdFoto(1);

    tabla.clear();
    for (URS u:usuarios) {
        u.setInstante(INSTANTE.intValue());
        tabla.put(new Integer(u.getID()),u);
    }
}
```

Por último, en el bloque de inicialización estática, debemos hacer un par de cosas:

La primera es inicializar los datos de la acción. Para ello, añadimos una nueva línea en el bucle que recorre los tipos de acción, que sea simétrica a todas las anteriores, pero cambiando debidamente el nombre de la acción. Quedaría así:

```
else if (n.equals(AccionSubirFoto.class.getSimpleName()))
{AccionSubirFoto.accion=t;AccionSubirFoto.proto.setCostePuntos(t.getPuntos());AccionSubirFoto.prot
o.setNombre(t.getNombre());}
```

La segunda es agregar el prototipo de acción al sistema. Para ello, debemos agregar una nueva entrada en el array *prototiposAcciones* de la siguiente manera:

```
prototiposAcciones[0]=AccionComentario.proto;
prototiposAcciones[1]=AccionCrearBlog.proto;
prototiposAcciones[2]=AccionCrearForo.proto;
prototiposAcciones[3]=AccionCrearGT.proto;
prototiposAcciones[4]=AccionMandarInvitacionGrupoTematico.proto;
prototiposAcciones[5]=AccionMandarMensajeUsuario.proto;
prototiposAcciones[6]=AccionMensajeAGT.proto;
prototiposAcciones[7]=AccionPedirAmistad.proto;
prototiposAcciones[8]=AccionPedirParticipacionForo.proto;
prototiposAcciones[9]=AccionPostAForo.proto;
prototiposAcciones[10]=AccionProcesarPeticionAmistad.proto;
prototiposAcciones[11]=AccionProcesarPeticionParticiparGT.proto;
prototiposAcciones[12]=AccionSubirFoto.proto;
```

Después de crear y modificar adecuadamente el modelo de datos, de crear las clases adecuadas para gestionar ese modelo de datos, y de inicializar la información de la acción, sólo queda definir el comportamiento de dicha acción. Para ello, nos volvemos a la clase *AccionSubirFoto* que creamos anteriormente, y definimos su método *ejecutarAccion*. El comportamiento será el de que el usuario que ejecuta la acción subirá una foto. Para ello, pediremos un identificador de foto y otro de comentable (porque quedamos en que las fotos eran comentables) para la foto, y mandaremos crear la foto al gestor de base de datos mediante la función creada para tal fin.

En cuanto al método *esFactible*, se trata de un método usado en acciones que usan criterios de elección, que nosotros no necesitaremos esta vez.

Agregamos también los métodos *setAccion* y *getAccion*, y el *toString*, que nos ayudará en depuración.

```
public class AccionSubirFoto extends AccionGenerica {
    public static TipoAccion accion;
    public static AccionSubirFoto proto=new AccionSubirFoto();
    public AccionSubirFoto(){}

    @Override
    public void ejecutaAccion(int i) {
        instante=i;
        int idCom=ExclusionMutua.consumeIdComentable();
        int idFoto=ExclusionMutua.consumeIdFoto();
        int idUser=usuario.getID();
        GestorDB db=new GestorDB();
        db.subirFoto(idUser,idFoto,idCom,i);

    }

    @Override
    public boolean esFactible() {
        return true;
    }
    @Override
    public String toString() {
        return "Acciï¿½n subir foto en el instante "+instante;
    }
    public  void setAccion(TipoAccion t) {accion=t;}
    public TipoAccion getAccion() {return accion;}

}
```

9.2 Añadir nuevos grupos de red social

Vamos a ver como podemos añadir al sistema un nuevo grupo en el que van a poder participar distintos usuarios de las red social. En la versión actual tenemos hechos los grupos *Foro* y *Grupo temático.* Vamos a ver cómo crear uno nuevo y le vamos a llamar *Club de Fans.*

Lo primero es añadir un identificador del tipo de grupo nuevo, el nombre y una descripción al modelo en la base de datos. Para ello modificamos el script de creación de la base de datos llamado *tablas.sql* y agregamos la siguiente línea.

```
insert into tiposGrupo (nombreTipoGrupo,descTipoGrupo) values ('Blog','En el blog solo puede escribir entradas el creador. Solo es accesible por los amigo
insert into tiposGrupo (nombreTipoGrupo,descTipoGrupo) values ('Club de Fans', 'Grupo que tendra a Fans de algo o alguien. Se participa por invitacion');
```

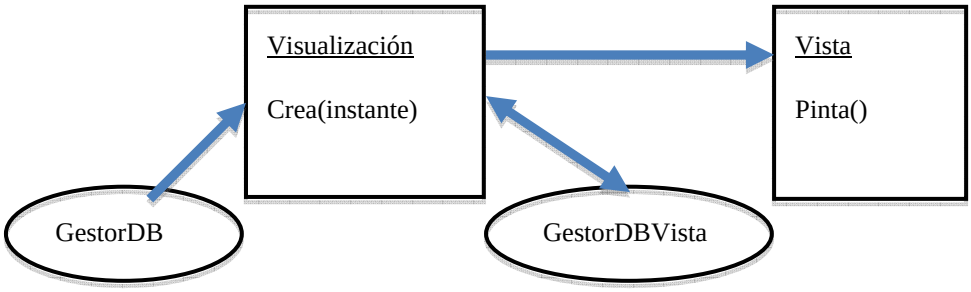
Y ya sólo nos queda crear una nueva clase en el paquete grupos que vamos a llamar *ClubDeFans*, esta nueva clase debe heredar de la clase *Grupo*.

```
public class ClubDeFans extends Grupo {

    public ClubDeFans(int grupo, int comentable, URS cr, int inst) {
        super(grupo, comentable, cr, inst, ExclusionMutua.getIdTipoGrupo("Club de Fans"));
        participantes=new ArrayList<URS>();
        participantes.add(cr);
    }

    public ClubDeFans(URS cr, int inst){
        this(ExclusionMutua.consumeIdGrupo(), ExclusionMutua.consumeIdComentable(), cr, inst);
    }

}
```

Una vez tenemos esto, para que tuviera sentido el nuevo grupo creado, deberíamos añadir acciones asociadas a él como crear un club de fans, invitar a club de fans y aceptar invitación a club de fans. De tal forma que al ejecutar los URS estas acciones se utiliza el nuevo grupo que hemos creado.

9.3 Agregar nuevas visualizaciones

Lo primero que debemos hacer para crear una nueva visualización de la red social, es crear una clase que herede de la clase abstracta *Visualizacion* en el paquete *visualizaciones.* Esta clase debe implementar el método abstracto *creaVisualizacion(),* ver diagrama de clases de las visualizaciones. El nuevo método que creemos debe encargarse de recopilar los datos que queremos mostrar de la red social, guardados en las distintas tablas de la base de datos, y crear una nueva estructura a la que la parte de interfaz se encargará de acceder para pintarla.



Como podemos ver en el esquema, la clase *GestorDB* tiene unos métodos a los que podemos llamar para obtener la información de la base de datos, cómo por ejemplo, todos los usuarios de la red social. Una vez obtenidos, los datos se procesan y se guardan mediante la clase *GestorDBVista* en otra tabla de la base de datos que nos

hemos tenido que crear para guardar los datos de la visualización. Seguramente tengamos que añadir a la clase *GestorDBVista*, los métodos necesarios para insertar lo datos necesarios en la base de datos. Es recomendable echar un vistazo a cualquiera de los métodos implementados para realizar la misma operación.

Una vez tenemos los datos de la visualización debemos crear una clase en el paquete *interfaz.vistas* que herede de la clase *Vista.* La clase *Vista* a su vez extiende *JPanel* de forma que podemos agregar a nuestra clase elementos gráficos, como botones o, como tenemos en otras vistas, grafos realizados con la librería JUNG o gráficas hechas con JFreeChart.

La clase *Vista* contiene unos atributos que vamos a tener que utilizar. Estos son un *ThreadVisualizacion,* que se trata del hilo que, cuando se muestre la vista que estamos creando, ejecuta el método *creaVisualizacion()* y avanza el instante de la vista cada segundo. Tenemos también un atributo *Visualizacion* que se trata de la clase anteriormente descrita que tiene el método *creaVisualizacion().* Y tenemos también un *Timer* que se encarga de refrescar la vista mostrada en la interfaz de tal manera que cuando el método *creaVisualizacion()* llamado en el *ThreadVisualizacion* modifica los datos se llama a el método *pinta()* de la vista para que se muestren estos nuevos datos por pantalla. Con estos elementos conseguimos que nuestra vista muestre la evolución de la red cada segundo, pintando un instante distinto cada vez.

En la nueva clase que hereda de *Vista*, debemos implementar los métodos abstractos que se encargan arrancar esta visualización de la red social. Podemos tomar como ejemplo los ya realizados para las otras visualización ya que el proceso sería el mismo.

```
@Override
public void empezar() {
        try {
            timer.schedule(new RemindTask(), 500, 500);
            //vistaMostrada.setInstante(0);
            hilo.start();
            hilo.setEjecutar(true);
            ejecutando=true;
        } catch (Exception e) {
            System.err.println("La visualización ya está empezada");
        }
}
```

También debemos crear la clase *RemindTask()* con el método *run()* el cuál se la a encargar a llamar al método *pinta()* de nuestra nueva vista. El método *pinta()* va a llevar a cabo todas las acciones necesarias para mostrar por pantalla la nueva visualización de la red social.

```
private class RemindTask extends TimerTask {

    @Override
    public void run() {

        pinta();
    }
```

Una vez creada la vista de la red social tenemos que añadir un menú a la interfaz principal del sistema para poder lanzarla. Para ello en la clase *JFrameGestor* debemos añadir las siguientes líneas en la constructora:

```
Visualizacion v= new VisualizacionAmistades(0);
Vista gr = new VistaGrafoAmistad(v);
GestorPanelVisualizacion aux=new GestorPanelVisualizacion("Grupos de Amistades",v,gr,this);
aux.addVisualizacionListener(this);
visualizaciones.add(aux);
```

Simplemente tenemos que llamar a la constructora de la nueva clase que hemos hecho que hereda de *Visualizacion* y a la nueva que hereda de *Vista.* De esta forma se crea el menú con el botón para mostrar y parar nuestra nueva vista en la interfaz y podemos verla en una ventana nueva al darle al botón mostrar.

10 GLOSARIO DE TÉRMINOS

- **URS**: Usuario de la red social. Nombramos URS a cada entidad que se encarga de simular a cada uno de los usuarios de la red social. Representa su perfil así como las acciones y relaciones con otros usuarios. Cada URS tiene un perfil asociado haciendo que este se comporte de una forma u otra en nuestra red.

- **Perfiles**: Clases en las que se dividen los URS las cuales determinan el comportamiento de cada uno según los porcentajes de realización de acciones que hay determinados en el perfil. Para cada perfil asignamos el porcentaje de cada tipo de acción que va a realizar el URS en un instante de tiempo.

- **Instante**: Entidad en la que se divide la línea temporal de la red social. Cada URS tiene asociado un instante de tiempo en el que se encuentra y un determinado número de acciones que puede realizar en este instante.

- **CRS:** Contenido de la red social. Un CRS es una entidad relacionada estrechamente con uno o varios grupos. Son las encargadas de unir usuarios y grupos, a través de acciones que tienen como consecuencia la creación, modificación o eliminación de CRSs.

- **Acciones:** La forma que tienen los usuarios de interactuar con la red social es mediante la ejecución de acciones a lo largo del tiempo. Estas acciones producen cambios en el estado de la red social, relacionados con los propios usuarios que las ejecutan, o con otros usuarios ajenos. Estas acciones conllevan el establecimiento de nuevas relaciones o la creación de nuevos grupos o contenidos.

- **Aficiones:** Cada usuario tiene una afición, escogida de entre un conjunto de aficiones preconfiguradas. Esta afición sirve para poder establecer nuevos criterios a la hora de tomar decisiones. Las aficiones forman una ontología, por lo que se pueden definir afinidades entre ellas.

- **Grupos:** Los grupos son entidades destinadas a relacionar usuarios entre sí sin necesidad de existir amistades. Cada tipo de grupo tiene asociadas unas determinadas acciones y contenidos a través de los cuales se puede interactuar con él, para por ejemplo, ingresar en el grupo, subir contenidos al mismo, o crearlo.

- **Comentarios:** Muchos de las entidades en Krowdix son comentables, como los usuarios, o muchos contenidos. Esto da pie a nuevas acciones y un tipo nuevo de contenido: el comentario. Un usuario puede comentar cualquier elemento que sea comentable y al cuál tenga acceso, existiendo para ello una acción llamada “Crear Comentario”. A su vez, los comentarios pueden ser también comentados, creando una cadena de comentarios.

11 BIBLIOGRAFÍA

ALONSO Fernández, Daniel, ARNALDOS Navarro, Mario, MONTENEGRO Portillo, César. Krowdix: simulador de redes sociales. Proyecto sistemas informáticos (2009). Universidad Complutense de Madrid. Facultad de Informática.

Only connect: Felix Grant looks at the application of data analysis software to social networks", *Scientific Computing World* June 2010: pp 9-10

Coleing, A., "The application of social network theory to animal behaviour" in *Bioscience Horizons*, 2009. 2(1): p. 32.

Huisman, M. and Van Duijn, M. A. J. (2005). Software for Social Network Analysis. In P J. Carrington, J. Scott, & S. Wasserman (Editors), *Models and Methods in Social Network Analysis* (pp. 270–316). New York: Cambridge University Press.

Dimitris V. Kalamaras (2010). The SocNetV Manual, http://socnetv.sourceforge.net/docs/manual.html

Vladimir Batagelj: First steps to visualization of networks with Pajek (Lecture at University of Konstanz, May 21, 2002).

Erwin Bonsma (2004), DIET Agents tutorial revision 1.26. http://diet-agents.sourceforge.net/.rsrc/tutorial.html

Cefn Hoile, Fang Wang, Erwin Bonsma and Paul Marrow, *"Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System"*, Proc. 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS2002), pp. 623-630, July 2002, Bologna, Italy

JUNG Manual http://sourceforge.net/apps/trac/jung/wiki/JUNGManual

Colaboradores de Wikipedia (2010). Wikipedia La Enciclopedia Libre, Java DataBase Connectivity (JDBC) http://es.wikipedia.org/wiki/Java_Database_Connectivity