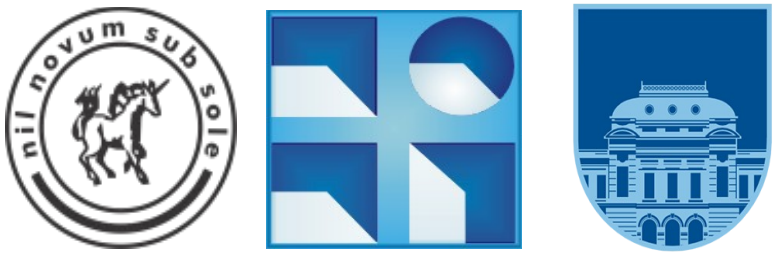
Informe del Proyecto de Grado:

"Recomendador con Friendsourcing Semántico"

Año 2014

Instituto de Computación,
Facultad de Ingeniería,
Universidad de la República,
Montevideo - Uruguay



Tutores:

Dra. Regina Motz
Dra. Libertad Tansini

Integrantes:

Marcos Suiffet
Diego Bastiani

Resumen

El objetivo de este proyecto consiste en el desarrollo de un Sistema Recomendador, que utiliza la información social contenida en Redes Sociales, y cuyo algoritmo de recomendación, emplea la técnica de Friendsourcing Semántico para brindar las recomendaciones a los usuarios. Friendsourcing Semántico extiende la similaridad entre usuarios, a partir de los tipos de relaciones que los usuarios tienen dentro de una Red Social, diferenciando de forma semántica, el tipo de relación por el cuál están conectados los usuarios en la Red Social, y de esta forma, ofrecer recomendaciones mas adecuadas, según sea el vinculo relacional que une a los usuarios.
Para alcanzar dicho objetivo, se realiza un estudio en profundidad de las técnicas de filtrado de información en Sistemas Recomendadores, de los algoritmos de Friendsourcing existentes, de la información social contenida en las Redes Sociales, y de los aportes que nos brinda la utilización de componentes incluidos en la Web Semántica.
El Sistema Recomendador se conforma de una componente lógica, donde se encuentra el algoritmo de Friendsourcing Semántico; de una componente de persistencia, empleando una base de datos relacional; de una componente gráfica para la interacción del usuario con el sistema; y de las fuentes de información, necesarias para un adecuado funcionamiento del Sistema Recomendador.
El testing del Sistema Recomendador, se basa en aplicar técnicas y métricas conocidas, utilizadas para medir la calidad de las recomendaciones brindadas. Se muestra a través de un caso de estudio sobre una Red Social simulada, pero factible de ser probada en la realidad, y aplicando varios casos de prueba sobre la misma, de como se obtienen recomendaciones personalizadas de gran relevancia y de gran calidad, demostrando el correcto funcionamiento del Sistema Recomendador.
Finalmente se extraen las conclusiones, se plantean posibles mejoras a lo realizado y se motiva a realizar futuros trabajos.

Palabras clave: Sistema Recomendador, Friendsourcing, Red Social, Web Semántica.

Índice General

Índice de Figuras

1 – Introducción.

Hoy en día disponemos de tanta cantidad de información dispersa en la web, que la habilidad para identificar cuál sería la mas útil para cada una de nuestras necesidades, se ve disminuida, teniendo en consecuencia, que el usuario no pueda encontrar a veces lo que realmente desea, sumado a esto, el de no disponer de tiempo suficiente para seleccionar lo mas interesante de lo que está en Internet para el mismo.
Es así que los Sistemas de Recomendación o Sistemas Recomendadores, surgen a partir de la necesidad de poder proveer a los usuarios de información relevante y personalizada, facilitándoles la búsqueda y la elección de productos o servicios [1].
Los Sistemas Recomendadores[2] son una clase de sistemas de filtrado de información (clasificados según la técnica de filtrado utilizada[3], basados en: filtrado colaborativo, contenido, la comunidad, conocimiento o híbridos), en los que se recomiendan ítems de información, que pueden interesar a un usuario, en base a sus características o preferencias, como pueden ser sus gustos o situación (datos extraídos por ejemplo de su perfil de usuario), y en base a las opiniones que brinden los demás usuarios (utilizando técnicas de Crowdsourcing o de Friendsourcing por ejemplo).
Crowdsourcing[4] es pedir la opinión de terceros (comunidad) frente a algún tema en particular o el desarrollo de una tarea específica. Utiliza la colaboración de muchas personas para resolver un problema o conocer la opinión frente a algo. En cambio Friendsourcing[4], es una forma de Crowdsourcing que recolecta información adecuada, disponible sólo para un confiable y eventualmente reducido grupo de personas que están conectadas socialmente en una red. De este modo, en el Crowdsourcing los comentarios son impersonales, mientras que en el Friendsourcing son totalmente cercanos y de confianza, y éste es factible de ser aplicado a redes sociales tales como son los casos de Facebook y de Google+, en donde si dos usuarios se encuentran conectados dentro de la red social, mediante algún tipo de relación, es porque existe afinidad y confianza entre ambos.
Por otra parte, en la actualidad las redes sociales forman parte de nuestra vida. En ellas los usuarios pueden comunicarse entré si, y además compartir todo tipo ítems de información o recursos, como ser: imágenes, videos, audios, páginas web, opiniones, eventos, etc., como ocurre en Facebook y en Google+, pero en las mismas, se observa una gran carencia, al momento de tener que calificar los recursos, a través de atributos de calidad de los mismos. En el caso de Facebook solo podemos calificar recursos a través de uno solo de estos: “Me gusta” o “Me encanta” o “Me divierte” o “Me asombra” o “Me entristece” o “Me enoja”, y para el caso de Google+ solo podemos calificar recursos a través de “+1”. Es por ello que es necesario dotar a las redes sociales de mas opciones de atributos de calidad para que los usuarios puedan calificar a los recursos incluidos en las mismas.
Y un volumen tan grande de información existente en las redes sociales (incluidas en Internet), y donde es dificultoso para el usuario identificar aquellos ítems de

información que desea, o en los que pueda estar interesado, es que se hace viable utilizar a los Sistemas Recomendadores en ellas, para facilitar la toma de decisiones de los usuarios, en temas en los que las opciones son numerosas y variadas [5].
En nuestro proyecto, las recomendaciones se realizan utilizando la información existentes dentro de cada red social, y es así que la información es conocida sólo a los usuarios de la misma, por lo tanto, nuestro Sistema Recomendador utiliza un mecanismo de recolección de la información presente en la red social, donde estos ítems de información han sido previamente evaluados por usuarios de la red social, y son visibles a otros usuarios que están directamente conectados socialmente (mediante algún tipo de relación) con el usuario que evaluó el ítem de información correspondiente, para finalmente poder devolver las recomendaciones al usuario que realizó la solicitud. Dicho mecanismo empleado en nuestro proyecto, será un algoritmo de Friendsourcing, dado que las recomendaciones a ofrecer a un usuario dentro de una red social, involucrarán ítems de información evaluados por usuarios próximos y de confianza del mismo.
Ahora bien, estudiando redes sociales como son los casos de Facebook y de Google+ en los **Anexo 1** y **Anexo 2**, se muestra el escaso aporte en lo que refiere a la semántica de las relaciones entre los usuarios de las redes mencionadas, al extraer información sobre los perfiles de usuario y sus relaciones vinculares con otros usuarios de la red, para ser empleadas en recomendaciones o en otra funcionalidad. Además, la semántica sobre las relaciones en estas redes sociales, aún no ha sido tomada en cuenta o usada, y sería muy bueno explotar esta área y dotarla de significado. En este sentido, el aporte que nos provee la Web Semántica puede ayudarnos a solucionar este inconveniente, dado que se incorpora contenido semántico a la web, permitiendo organizar por conceptos la información que ofrecemos, garantizando búsquedas sobre ésta por significado y no por contenido textual[6]. Y esto lo aplicamos a nuestro trabajo, con el fin de obtener similitud entre usuarios a partir de los tipos de relaciones que los usuarios tienen, cuando queremos utilizar las mismas por ejemplo al aplicar Friendsourcing.
Contemplando todo lo expuesto, es que nuestro proyecto consiste en la realización de un Sistema Recomendador, que provee atributos de calidad mas adecuados para que los usuarios puedan calificar en los recursos, y busque las recomendaciones mas apropiadas para los mismos dentro de una red social, utilizando un algoritmo de Friendsourcing Semántico, es decir, que extienda y que se aproveche la similaridad entre usuarios emparejada con los tipos de relaciones que los usuarios tienen, esto es, utilizar las relaciones de forma semántica, con el fin de poder brindar recomendaciones de mayor calidad, apropiadas para usuarios que se encuentren vinculados en la red social mediante algún tipo de relación.

1.1 Descripción del tema.

Para la realización del Sistema Recomendador con Friendsourcing Semántico, es necesario dar a conocer y describir, ciertos aspectos y conceptos, útiles para una comprensión mas adecuada de lo que se ha realizado en el presente proyecto, así como también, una revisión de los antecedentes existentes, consultados previamente para la realización del proyecto.

Sistemas Recomendadores:

Los Sistemas Recomendadores o Sistemas de Recomendación forman parte de un tipo específico de técnica de filtrado de información, los cuales presentan distintos tipos de temas o ítems de información (películas, música, libros, noticias, imágenes, páginas web, etc.) que son de interés para un usuario en particular.
Los sistemas recomendadores son herramientas de software focalizadas en ayudar a usuarios en la toma de decisiones cuando estos interactúan con un gran volumen de datos. Están dirigidos, en principio, a personas que carecen de la experiencia, la capacidad o el tiempo necesarios para evaluar la inmensa cantidad de tópicos/items que generalmente están a su disposición en un sitio web y se basan en la premisa de que, habitualmente, la gente tiene en cuenta las recomendaciones provistas por otros para tomar decisiones rutinarias [7].
Los sistemas de recomendaciones surgieron en la década de los noventa, dentro de los servicios de grupos de noticias (newsgroup), el cual brindaban servicios de filtrado de noticias permitiendo a su comunidad de usuarios acceder a aquellas noticias que podrían llegar a ser de su interés en función de sus preferencias [8]. Los sistemas de recomendaciones utilizan técnicas de descubrimiento de conocimiento para poder generar recomendaciones personalizadas durante una interacción con el usuario [9]. Una gran cantidad de sitios web dedicados al comercio electrónico utilizan sistemas de recomendación para ayudar a sus clientes a la hora de comprar productos que sean de su interés. Los sistemas de recomendación aprenden los intereses de los clientes y recomiendan productos que son más valiosos para ellos a partir de los productos disponibles en el sitio web. De esta manera estos sistemas ayudan a dichos sitios a incrementar sus ventas [10]. En el caso de Amazon, el 35% de las ventas resultan de recomendaciones brindadas en el sitio web [11].
Los sistemas recomendadores se encuentran actualmente en innumerables aplicaciones que exponen al usuario a enormes colecciones de elementos o artículos entre los cuales puede elegir y su función principal es la de proveer al visitante o usuario de una lista de artículos recomendados que podrían resultar de su interés. Para lograr este objetivo, dada la gran diversidad en cuanto a los gustos de los

usuarios, existen diferentes métodos. En algunos casos, el sistema recomendador cuantifica la preferencia de cada artículo, por ejemplo a partir de calificaciones (rankings) que les han asignado otras personas. Ante una elección del usuario por un determinado artículo, ciertos sistemas recomendadores inclusive presentan una lista de tópicos adicionales que resultaron de interés para otros usuarios que eligieron el mismo artículo.

Técnicas de filtrado de información:

Los Sistemas Recomendadores, pueden utilizar diversas técnicas de filtrado de información, como describimos en forma detallada en el **Anexo 3**. Dadas las características de nuestro proyecto, la mas apropiada, es la técnica de filtrado de información mediante Filtrado Colaborativo, dado que para obtener las recomendaciones, nuestro Sistema Recomendador se basa en las evaluaciones que los usuarios realizaron de forma explícita sobre ítems de información presentes en la red social en la que se encuentran, conjuntamente con la similitud entre usuarios que es provista por las relaciones vinculares que los usuarios tienen con otros usuarios dentro de la red social.
Filtrado Colaborativo se basa, en que si una persona A tiene la misma opinión que una persona B sobre un tema, A es más probable que tenga la misma opinión que B en otro tema diferente que la opinión que tendría una persona elegida al azar. Por ejemplo, un sistema de recomendación basado en el filtrado colaborativo para televisión podría hacer predicciones acerca de los programas que le gustarían a un usuario a partir de una lista parcial de los gustos de ese usuario (gustos o disgustos). Estas predicciones son específicas para el usuario, pero utilizan la información obtenida de muchos usuarios.

Friendsourcing:

Para hablar de Friendsourcing, primero debemos definir que es Crowdsourcing.
Crowdsourcing es un término utilizado para referirse al reclutamiento en línea de personas, para que éstas completen tareas que son extensas o complejas para ser llevadas a cabo por una sola persona. En las aplicaciones que utilizan Crowdsourcing, una gran cantidad de personas realizan individualmente pequeñas contribuciones agregando medidas o soluciones al valor global de los contenidos[4]. Algunos ejemplos conocidos que utilizan esta técnica: Wikipedia, YouTube, Flickr, Starbucks,

Netflix.
Friendsourcing es una forma de Crowdsourcing que recolecta información adecuada, disponible sólo para un confiable y eventualmente reducido grupo de personas que están conectadas socialmente en una red. Recupera información social en un contexto social para producir resultados de interés [4].
De este modo, Friendsourcing es la recolección automática de información de utilidad para un usuario a partir de su red de amigos o relaciones de confianza en una red social. La información puede consistir en recomendaciones, contenidos, referencias, experiencias, ofertas, etc.
En este proyecto empleamos Friendsourcing sobre una red social simulada donde existen relaciones de confianza entre los usuarios (mediante el tipo de vínculo relacional que los une), y utilizando la estrategia de filtrado colaborativo, que es un método para hacer predicciones automáticas (filtrado) sobre los intereses de un usuario mediante la recopilación de las preferencias o gustos de información de muchos usuarios (colaborador).

Web Semántica:

La web semántica es un área pujante en la confluencia de la Inteligencia Artificial y las tecnologías web que propone introducir descripciones explícitas sobre el significado de los recursos, para permitir que las propias máquinas tengan un nivel de comprensión de la web suficiente como para hacerse cargo de una parte, la más costosa, rutinaria, o físicamente inabarcable, del trabajo que actualmente realizan manualmente los usuarios que navegan e interactúan con la web.
Es así, que en el **Anexo 4**, se estudia lo que es la Web Semántica, cómo es que surge, se analizan los problemas de la web actual y sus limitaciones, se muestra para que sirve, como es que funciona y cuales son sus pilares básicos.
Y en el **Anexo 5**, se explica detalladamente como es que está compuesta su Arquitectura.
De esta forma, analizando en detalle a la Web Semántica, sería muy beneficioso para el funcionamiento de los Sistemas Recomendadores, que se utilice como plataforma para los mismos a la Web Semántica [6].
Es así que en nuestro proyecto incorporamos muchos de los componentes incluidos en la Web Semántica, por ejemplo:

- URIs para referenciar recursos.
- Lenguaje XML para el formato de los archivo de: la red social, los recursos, los atributos de calidad, etc.
- Esquemas XML (XML Schema) para la validación de los archivos XML.

- Representación de la red social mediante un grafo RDF.
- Representación de las relaciones entre usuarios y de las áreas temáticas de los recursos, mediante Ontologías de vocabulario.

También se utilizan herramientas recomendadas para la construcción de aplicaciones basadas en la Web Semántica[12] como ser:

- Construcción de Ontologias de vocabulario con Protégé, por ser un editor simple e intuitivo, y dado que existe en la web una gran cantidad de información para su uso.
- Librerías Apache Jena, para poder utilizar las Ontologías de vocabulario en nuestra implementación del Sistema Recomendador, ya que es realizado en lenguaje de programación Java.
- SPARQL, que es un lenguaje estandarizado de consultas sobre grafos RDF.

Redes Sociales:

Según Watts[13], una red social es la representación de una estructura de relaciones sociales entre individuos. La red social indica de qué forman están conectados los nodos (individuos) por medio de diversos indicadores de familiaridad, que van desde "conocidos" hasta "miembros de una misma familia". El análisis de redes sociales es hoy en día una técnica imprescindible de estudio social.
Según Scott[14], una red social se configura como un conjunto de nodos unidos entre sí por vértices que representan las relaciones entre ellos. Sobre esa red se pueden estudiar múltiples parámetros que definen sus características. Entre los más importantes se encuentran *el grado de un nodo* (el número de enlaces que tiene con los demás nodos), *la centralidad* (que indica la importancia de un nodo dentro de la red), *el alcance* (el grado en que cualquier miembro de la red puede llegar a otros miembros), o *la cercanía* (el grado en el que un individuo se encuentra cerca de todos los otros miembros de la red).
En general, el análisis de redes sociales se realiza sobre comunidades preexistentes. Pero el mismo análisis puede realizarse para crear comunidades y relaciones sociales, empleando los resultados del análisis de redes sociales como guía para su evolución.

Lenguaje de modelados de Redes Sociales (FOAF).

FOAF[15] (Friend of a Friend) es el lenguaje más empleado para el modelado de redes sociales. FOAF está basado en RDF y se define empleando OWL. Fue diseñado para ser extensible y facilitar que sistemas informáticos diversos pudiesen compartir datos. FOAF permite definir a una persona e indicar a qué otras personas conoce.

Recomendador con Friendsourcing Semántico:

Definimos Recomendador con Friendsourcing Semántico como un Recomendador que utiliza una algoritmo para obtener las recomendaciones, basado en el uso de la combinación de Friendsourcing y de la aplicación de semántica sobre las relaciones entre usuarios en una red social (ej.: un “familiar” puede ser un “padre”, una “madre”, un “hermano”, una “hermana”, etc.). En nuestro proyecto, como se mencionó anteriormente, las relaciones entre usuarios son representadas a través de Ontologías de vocabulario.
Es importante diferenciarlo de un Recomendador Semántico con Friendsourcing [16], cuya diferencia radica en que un Recomendador Semántico se basa en recomendar recursos semánticamente similares al realizar la búsqueda de información para el usuario (ej: si pido recomendaciones al sistema recomendador sobre “alimento para perros”, será similar a pedir recomendaciones para “comida balanceada para mascotas” o “ración para canes”), y en que al aplicar el algoritmo de Friendsourcing al mismo, no diferencia semánticamente las relaciones entre usuarios en la red social.
Un Recomendador Semántico con Friendsourcing Semántico es la combinación de las dos anteriores.
Es así que este proyecto se basa fuertemente en los conceptos de Sistemas de Recomendación, en el de Friendsourcing, en el de Web Semántica y en el de las Redes Sociales. También son de suma importancia las herramientas provistas por cada uno de estos conceptos para lograr una implementación adecuada del proyecto.

Antecedentes:

El presente proyecto toma como base otro proyecto existente que realiza recomendaciones empleando el algoritmo de Friendsourcing[17].
Dicho antecedente, está enmarcado en el proyecto de investigación *LACCIR Quality Health Information Retrieval* (QHIR): *Improving Semantic Recommender Systems with Friendsourcing*[18], en el cual se desarrolla una red social en un entorno médico (Red UniSalud[19]) y un sistema de calificación y recomendación de recursos utilizando el contexto social obtenido de la red social.
En el mencionado proyecto, se provee un algoritmo de recomendación basado en Friendsourcing, que brinda recomendaciones personalizadas a los usuarios de la red social y que éstas son confiables para los mismos.
El algoritmo realizado se integró a un sistema real de recomendaciones y también se llevaron a cabo pruebas reales. En las pruebas realizadas para testear el algoritmo se

obtuvieron resultados eficientes (precisión = 75 %, recall = 78 %), mostrando que la información contenida en las redes sociales es sumamente importante para un sistema de recomendación.
Pero en el mismo, no se hacen distinciones entre los diferentes tipos de relacionamientos que se puedan dar en la red social en el que se aplica el mismo, es decir, se realizan las recomendaciones sin tomar en cuenta los diferentes tipos de relacionamientos entre usuarios en la red social, no existiendo semántica alguna que diferencie dichas relaciones.
Es por ello que en pos de aprovechar los distintos tipos de relaciones vinculares entre usuarios que se puedan dar dentro de una red social, y esto pueda llegar a condicionar de manera positiva los ítems obtenidos en una recomendación (satisfagan y resulten útiles para el usuario), y dado que el punto de aplicación se focaliza específicamente en grupos de relaciones vinculares semánticamente definidos, es que se plantea la realización de un Sistema Recomendador con Friendsourcing Semántico que tome en cuenta lo descrito anteriormente y mejore lo realizado en el proyecto existente como antecedente.

1.2 Objetivos del trabajo presentado.

El objetivo de este proyecto es la creación de un Sistema Recomendador que utiliza Friendsourcing Semántico, es decir, que se basa fuertemente en las relaciones vinculares entre usuarios semánticamente similares dentro de una Red Social, para brindar las recomendaciones.
Para ello se provee la especificación, el diseño y la implementación de un algoritmo de recomendación basado en Friendsourcing Semántico, el cuál se encuentra incluido en el Sistema Recomendador.
Se incluye una interfaz web que muestra el funcionamiento del Sistema Recomendador.
Se ejecutan casos de pruebas sobre el Sistema Recomendador y se analizan los resultados obtenidos.
Finalmente se presentan las conclusiones y se expone sobre posibles mejoras para agregarle al trabajo, así como también la incentivación para la realización de trabajos a futuro, tomando como referencia el presente proyecto.

1.3 Estructura del presente documento.

El presente documentó será organizado de la siguiente manera:

- Capítulo 1: Planteamiento y definición del tema, donde se describe y se motiva la realización del proyecto, se definen conceptos que aportan al presente trabajo y son indispensables para entender el mismo, entre ellos se destacan principalmente: los Sistemas de Recomendación, el Friendsourcing, la Web Semántica y la Red Social. Se hace una revisión de los antecedentes existentes, consultados previamente para la realización del proyecto. Se muestra como está organizado el documento entregado.

- Capítulo 2: Se realiza la explicitación del problema, esto es, se especifica el problema a resolver, los requerimientos necesarios para llevarlo a cabo, el modelado y diseño de diversos componentes requeridos para ser usados por el algoritmo de Friendsourcing Semántico, la realización del algoritmo de recomendación con Friendsourcing Semántico y la secuencia de pasos para obtener las recomendaciones.

- Capítulo 3: Se muestra la implementación realizada del algoritmo de Friendsourcing Semántico y como se incluye el mismo en un Sistema Recomendador. Esto incluye: las fuentes de información, la componente lógica, la componente de persistencia y la interfaz gráfica.

- Capítulo 4: Se presentan las técnicas y las métricas utilizadas para evaluar el sistema, se muestran las pruebas realizadas sobre el mismo y se analizan los resultados obtenidos.

- Capítulo 5: Se presentarán las conclusiones del trabajo realizado, las posibles mejoras a aplicarle al mismo y se exponen ideas para el desarrollo de trabajos a futuro.

- Bibliografía: Referencias del material citado en el proyecto.

- Apéndice: Contenidos de información extra para el proyecto.

2 – Algoritmo de Recomendación usando Friendsourcing Semántico.

En este capítulo se define el problema a resolver, así como los requerimientos (funcionales y no funcionales) necesarios para llevarlo a cabo.
Se explica el diseño del algoritmo de recomendación a usar, así como el diseño de los diversos componentes que lo rodean, los cuales son necesarios para su correcto funcionamiento(red social, ontologías de vocabulario, recursos, atributos de calidad de los recursos, recomendaciones, etc.).

2.1 Especificación del Algoritmo.

Crear un algoritmo de Friendsourcing Semántico basándose en el significado que tienen las relaciones vinculares entre usuarios dentro de una red social para ser utilizado en el Sistema Recomendador. El mismo se aplica a una red social donde los usuarios desean obtener recomendaciones de recursos sobre diversas temáticas, los cuales deberán ser evaluados previamente por los integrantes de la red para ser tenidos en cuanta en las futuras recomendaciones.

Los recursos son clasificados dentro de áreas temáticas (películas, música, libros, etc), por lo cuál se desarrolla una ontología de vocabulario para dicho propósito (ontología de áreas temáticas, que pueda distinguir semánticamente las diversas áreas temáticas a las que pertenece un recurso).

Las interacciones entre usuarios dentro de una red social que se puedan dar, dependen fuertemente de las relaciones vinculares que existan entre ellos (amistad, familiar, laboral, académico, etc.), por lo cuál se desarrolla una ontología de vocabulario que contemple este aspecto (ontología de relaciones, que pueda distinguir semánticamente las diferentes relaciones presentes dentro de una red social).

Se especifican y se modelan diversas estructuras necesarias para llevar a término el proyecto descripto:

- La red social, incluyendo el nombre que identifica a la misma, así como los usuarios que la componen y las relaciones vinculares entre éstos (perfiles de usuario).
- Los recursos.
- Los recursos evaluados por usuarios.
- Los atributos de calidad de los recursos de una red social en particular.
- Las consultas realizadas por los usuarios para obtener las recomendaciones.

- El algoritmo de Friendsourcing Semántico, donde se incluye claramente como se realiza el cálculo de las puntuaciones de los recursos evaluados por usuarios para obtener las recomendaciones.

Se provee un usuario con privilegios sobre el sistema recomendador (administrador) para realizar tareas administrativas y/o de mantenimiento sobre el mismo.

No está dentro del alcance del proyecto realizar un extractor de recursos para alimentar el sistema (en caso de no realizarlo se mostrará como obtener los mismos o dar pautas de como se podrían obtener).

Se desarrolla una interfaz gráfica para el Sistema Recomendador.

2.2 Requerimientos

Para realizar lo anterior se especifican a continuación los requerimientos funcionales y no funcionales que debe tener el Sistema Recomendador, entendiéndose por:

- requerimientos funcionales: que debe hacer el Sistema Recomendador.
- requerimientos no funcionales: como debe ser el Sistema Recomendador.

2.2.1 Requerimientos funcionales

- ***Fuentes de información:*** El algoritmo de recomendación debe tener en cuenta las siguientes fuentes de información para realizar las recomendaciones personalizadas: la red social a la que pertenece el usuario al que se le quiere brindar las recomendaciones y las ontologías de vocabulario para la relaciones que se establezcan dentro de la red social y para las áreas temáticas a las que pueda pertenecer un recurso. También serán requeridos los recursos, los recursos evaluados por los usuarios(opcional), los atributos de calidad para los recursos, y validadores de las fuentes de información que chequeen el correcto formato de las mismas. En cuanto a las evaluaciones de recursos, se sabe que los usuarios pueden calificar hasta cuatro atributos de calidad que son: trustwhorthy (confiable), objective (objetivo), complete (completo) y well-written (bien escrito), para una red social por defecto (definamos red social por defecto, a una red social distinta a Facebook y a Google+, pudiendo existir o ser creada con el fin de ser utilizada para evaluar el proyecto). Si hablamos de una red social como Facebook se podrá evaluar un solo atributo de calidad de los siguientes: “meGusta”, “meEncanta”, “meDivierte”, “meAsombra”, “meEntristece” y “meEnoja”, y si hablamos de una red social como Google+, el atributo de calidad a evaluar será “+1”.

- ***Persistencia de datos:*** El Sistema Recomendador persistirá los recursos, los recursos evaluados por los usuarios, los atributos de calidad para los recursos y las consultas de usuario que se realicen por parte de los mismos.
- ***Operaciones permitidas en el sistema:***
  - El usuario podrá realizar las siguientes operaciones en el sistema: solicitar recomendaciones, evaluar recursos, ver el historial de las evaluaciones de recursos que realizó, ver el historial de las solicitudes de recomendaciones que realizó.
  - El administrador podrá realizar las siguientes operaciones en el sistema: cargar recursos para ser evaluados por los usuarios así como también podrá setear los pesos de los atributos de calidad de los recursos, que utiliza el sistema para obtener las recomendaciones solicitadas por los usuarios en una red social dada.
- ***Obtención de las recomendaciones:*** Para obtener las recomendaciones el usuario deberá ingresar: la relación en la red social donde se buscaran las recomendaciones, la distancia dentro de la red social a la que estén los usuarios del usuario que hace la solicitud (distancia “1” implica a los usuarios directamente relacionados, distancia “2” implica a los usuarios directa e indirectamente relacionados), el área temática a la que deben pertenecer los recursos y la relación para la cuál tienen como objetivo los recursos evaluados por parte de los otros usuarios de la red social (los usuarios al evaluar un recurso, además de calificar sus atributos, indican a criterio de ellos, cuál debería ser la relación mas apropiada que debe tener el recurso como destinatario dentro de su red social).
- ***Salida del algoritmo de recomendación:*** El algoritmo de recomendación debe devolver una lista de recursos recomendados para un usuario en particular, y el resultado debe estar ordenado en función de la relevancia para el usuario (puntuación).
  A su vez se podrán mostrar al usuario recursos que aún no han sido evaluados y que pueden ser de interés para el mismo.

2.2.2 Requerimientos no funcionales

- ***Tecnología:*** Se requiere que se utilice un software no propietario para el desarrollo del algoritmo de recomendación. Esto es debido a que en caso de que en el futuro se quiera extender el desarrollo del algoritmo, no haya impedimentos en cuanto a licencias. Por lo tanto en las reuniones se sugirió la

opción de desarrollar el algoritmo de recomendación en Java dado que existen gran cantidad de librerías para el manejo de ontologías, grafos rdf, etc..

- ***Adaptabilidad:*** Se requiere que el algoritmo de recomendación a construir esté preparado a readaptarse a diferentes redes sociales.
- ***Escalabilidad:*** El sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental, de manera tal que nuevas funcionalidades y requerimientos relacionados puedan ser incorporados afectando el código existente de la menor manera posible; para ello deben incorporarse aspectos de reutilización de componentes. El sistema debe estar en capacidad de permitir en el futuro el desarrollo de nuevas funcionalidades, modificar o eliminar funcionalidades después de su construcción y puesta en marcha inicial.

2.3 Diseño del algoritmo

A continuación se explican los diversos diseños realizados sobre los componentes primordiales requeridos para la construcción del algoritmo de Friendsourcing Semántico.

2.3.1 Modelado de la Red Social

En la actualidad existen diversas redes sociales con variados objetivos, características y/o propósitos (ej.: Facebook, Google+, Twitter, Instagram, etc.). En general, los servicios de redes sociales permiten a los usuarios crear una cuenta de usuario con un perfil que los identifica, el cual deberá poseer cierta información personal del usuario. Una vez que accedan a su cuenta, podrán establecer contacto con otros individuos, y luego, será posible comunicarse entre ellos.

Para el caso de nuestro algoritmo de recomendación, la red social deberá aportar:

- el nombre de la misma,
- los perfiles de los usuarios (identificador, nombre, etc.),
- las relaciones vinculares entre los usuarios.

Para el caso de redes sociales conocidas se emplearan Facebook o Google+, en caso de otra red social se usará una red social “por defecto”. Esto influirá en los atributos de calidad de los recursos que evaluarán los usuarios de la red social en la que se encuentren.

En el caso de los perfiles de usuario, es necesario que se cuente con un identificador de usuario en cada red social para que las recomendaciones brindadas por el algoritmo puedan ser personalizadas. El perfil de usuario consta de un identificador en la red social e información del usuario.

Se soportarán diversos tipos de vínculos entre los usuarios, como ser de amistad, de familia, de labores y/o académico, abarcando una amplia gama de relaciones posibles entre usuarios dentro de una red social y que luego a la hora de solicitar las recomendaciones los usuarios, se tomarán muy en cuenta el tipo de vínculo con el cuál se relacionan los mismos.

La manera de modelar la red social será a través de un grafo dirigido donde los nodos serán los usuarios (incluyendo la información de sus perfiles) y los arcos serán las relaciones. De esta manera es posible modelar cualquier red social a través de un grafo.

Pero en particular para nuestro proyecto, el Sistema Recomendador utiliza para modelar la red social un grafo RDF (sujeto → predicado → objeto), donde el sujeto es la uri de un usuario, y el objeto y el predicado pueden ser:

- la uri de otro usuario y la uri de la relación que los vincula a ambos.
- una cadena de texto y la uri que establece una propiedad del sujeto.
- una uri que define que algo es una persona y la uri que establece la propiedad de que el sujeto es de ese tipo.

La uri creada para el usuario es: "http://rwsf/nombreRedSocial/nombreUsuario", donde nombreRedSocial identifica a la red social (www.google+.com, www.facebook.com, www.example.com (definido para una red simulada)) y nombreUsuario es el identificador del usuario dentro de la red social, y donde rwsf es la abreviación de “recommender with sematic friendsourcing”.

La uri para las relaciones se describe en la sección 2.3.5.1, además se utilizó la uri estándar conocida "http://xmlns.com/foaf/0.1/knows" para establecer la propiedad de que dos usuarios están conectados entre si en la red social.

La uris estándares conocidas "http://xmlns.com/foaf/0.1/name" y "http://xmlns.com/foaf/0.1/accountName" se utilizaron para establecer propiedades de los usuarios en la red social, como ser el nombre del usuario y el identificador de la cuenta del mismo.

La uri estándar conocida "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" se utilizó como predicado para indicar que un sujeto(uri de un usuario en la red social) es una persona (a través del objeto definido en la uri estándar conocida "http://xmlns.com/foaf/0.1/Person").

En la siguiente imagen se muestra como se modela la red social, tomando para ello, dos usuarios y la relación que existe entre ellos como ejemplo, a través de un grafo RDF:

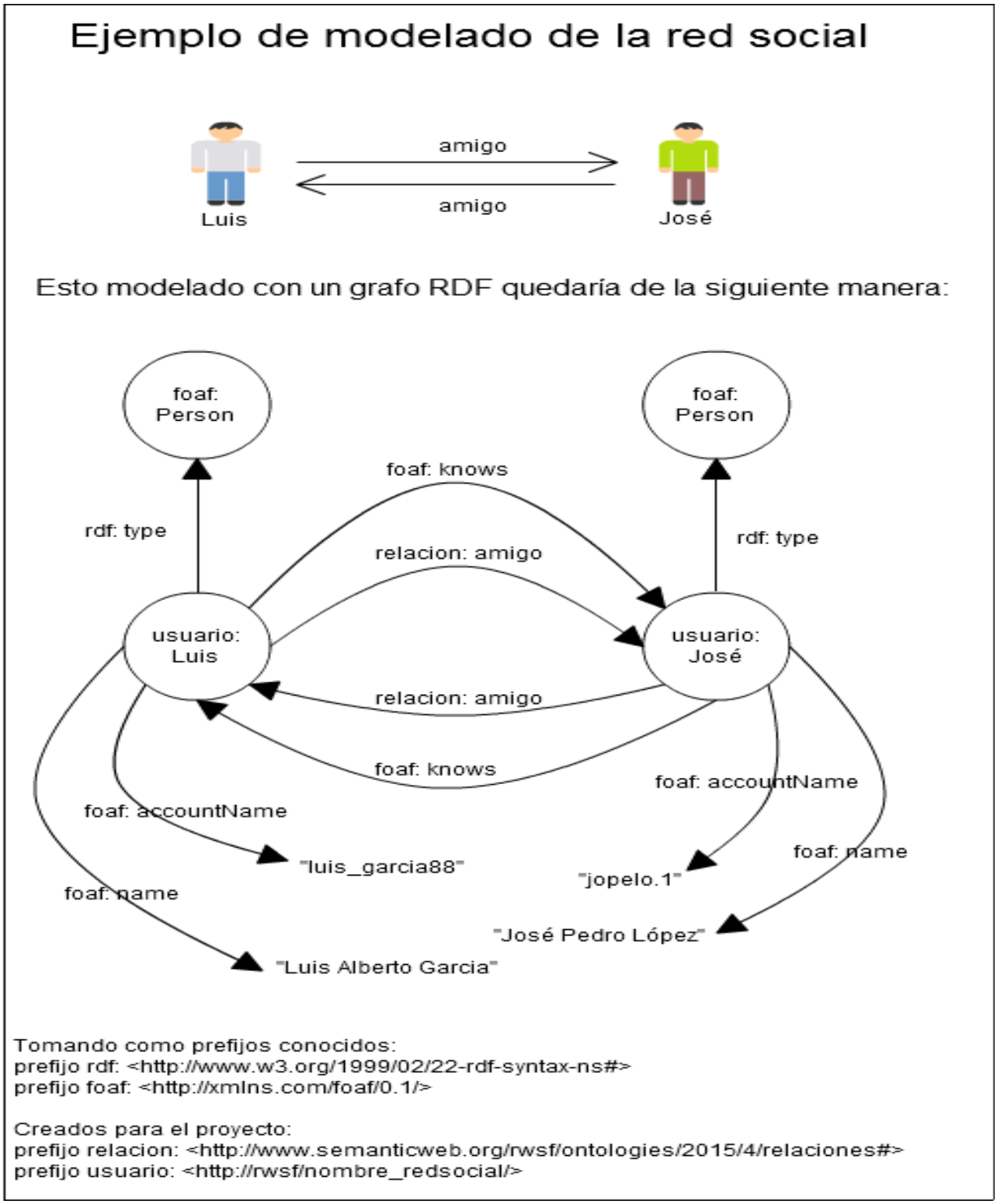


Figura 1 - Ejemplo de Modelado de la Red Social

2.3.2 Modelado de los Atributos de calidad de los Recursos

En las redes sociales como ser Facebook o Google+ existe la forma de calificar recursos (imágenes, videos, noticias, escritos, etc.) a través de un solo atributo de calidad. En el caso de Facebook solo podemos calificar recursos a través de uno solo de estos: “Me gusta” o “Me encanta” o “Me divierte” o “Me asombra” o “Me entristece” o “Me enoja”, y para el caso de Google+ solo podemos calificar recursos a través de “+1”, lo que hace muy limitado el margen de calificación. Sin embargo hay otras redes sociales que permiten realizar calificaciones de recursos con un abanico mas amplio de posibilidades (puntuar de 0 a 10 un recurso por ejemplo).

En el caso de redes sociales como Facebook o Google+, podríamos decir que un recurso adquiere mayor relevancia según la cantidad de “Me gusta” o “Me encanta” (por ejemplo) o “+1” que tenga, y esta valuación se adquiere en forma grupal de los usuarios y no en forma individual, por lo que propondremos que un usuario pueda calificar de forma individual un recurso y darle una valoración extra a los atributos de Facebook, en el rango de 0 a 10, usando por ejemplo a las siguientes valorizaciones (pueden darse otras): “Me encanta” valorado como 10, “Me gusta” valorado como 8, “Me divierte” valorado como 6, “Me asombra” valorado como 4, “Me entristece” valorado como 2 y me “Me enoja” valorado como 0. Y de Google+, elegir un valor adecuado para el atributo +1 (también en un rango de 0 a 10). Como vemos anteriormente, en Facebook podemos darle otro sentido a los atributos que nos presenta, de forma de utilizarlos para diferenciar la calidad de los recursos que se comparten allí y poder utilizar esta categorización como ejemplo para obtener recomendaciones mas adecuadas. No es así en el caso de Google+, donde hay un solo atributo sobre los recursos.

Para el caso en que no se esté en una red social como ser Facebook o Google+ (la definimos como red social por defecto), se posibilita evaluar un recursos a través de cuatro atributos de calidad, los cuales son:

- Trustwhorthy (confiable)
- Objective (objetivo)
- Complete (completo)
- Well-written (bien escrito)

Los mismos son los que se utilizaron para el proyecto tomado como antecedente[17] y fueron mantenidos para tal fin.

Los atributos de calidad para un recurso pueden ser calificados en un rango entre 0 y 10 inclusive para una red por defecto.

Es decir:

- Trustwhorthy -> se califica de 0 a 10
- Objective -> se califica de 0 a 10
- Complete -> se califica de 0 a 10
- Well-written -> se califica de 0 a 10

Ahora, se desea modelar que las valuaciones de un recurso por parte de los usuarios se encuentre siempre entre los valores 0 y 10.

Para ello se emplean ponderadores (reales entre 0 y 1) que indican el peso del atributo de calidad, que deben cumplir lo siguiente:

- $\sum \text{peso}_{\text{atributo-i}} = 1$
- $0 \leq \text{peso}_{\text{atributo-i}} \leq 1$

Entonces las valuaciones de los recursos por parte de los usuarios, según las redes sociales será:

- Facebook: $0 \leq \text{calificación}_{\text{'atributo-i'}} * \text{peso}_{\text{'atributo-i'}} \leq 10$ (donde atributo-i = “Me gusta” o “Me encanta” o “Me divierte” o “Me asombra” o “Me entristece” o “Me enoja”, y $\text{peso}_{\text{'Atributo-i'}} = 1$)
- Google+: $0 \leq \text{calificación}_{\text{''+1'}} * \text{peso}_{\text{'+1'}} \leq 10$ (donde $\text{peso}_{\text{'+1'}} = 1$)
- Red por defecto:
  - $0 \leq \text{calificación}_{\text{'atributo-i'}} \leq 10$
  - $0 \leq (\text{calificación}_{\text{'Trustwhorthy'}} * \text{peso}_{\text{'Trustwhorthy'}}) + (\text{calificación}_{\text{'Objective'}} * \text{peso}_{\text{'Objective'}}) + (\text{calificación}_{\text{'Complete'}} * \text{peso}_{\text{'Complete'}}) + (\text{calificación}_{\text{'Well-written'}} * \text{peso}_{\text{'Well-written'}}) \leq 10$

Con lo anterior cada vez que se evalúa un recurso se pueden obtener valuaciones de entre 0 y 10 puntos.

El peso de los atributos de calidad deberán ser cargados en el sistema previamente a solicitar recomendaciones.

Además cuando uno califica los atributos de un recurso, solo se le indica al sistema, el valor del atributo de calidad (el sistema conoce los atributos y los muestra para ser evaluados según la red en la que se encuentre). En el caso de Facebook solo puede elegir uno de los atributos de calidad y en el caso de Google+, se utiliza el atributo +1. Pero estos ya tienen una calificación impuesta.

Por ejemplo si estoy en Facebook y un usuario evalúa un recurso con el atributo “Me gusta”, la valuación obtenida de ese recurso será de 8 puntos.

2.3.3 Modelado de la evaluación de los Recursos

Los usuarios en el sistema pueden evaluar un recurso, y de esta forma el sistema recomendador se nutre de nueva información para ser consultada y posiblemente incluir en las recomendaciones que brinda.

La siguiente imagen muestra en que consiste evaluar un recurso:

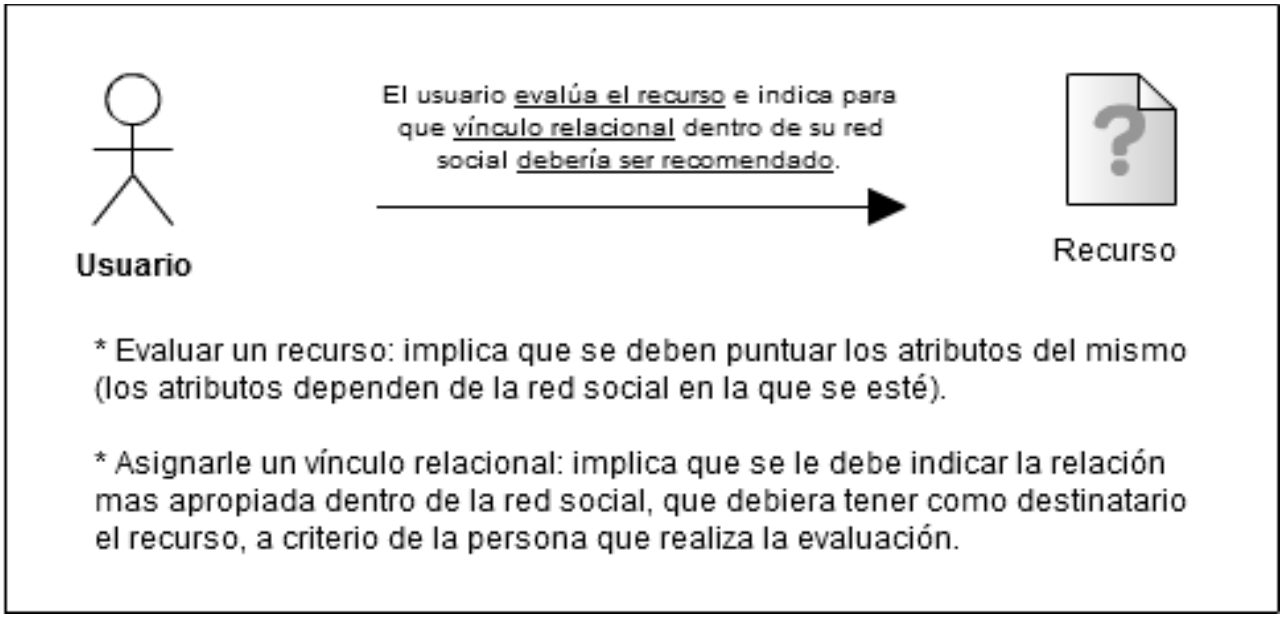


Figura 2 - Modelado de evaluación de los Recursos

Un ejemplo de usuarios que evalúan recursos dentro de una red social por defecto se muestra en el siguiente ejemplo:

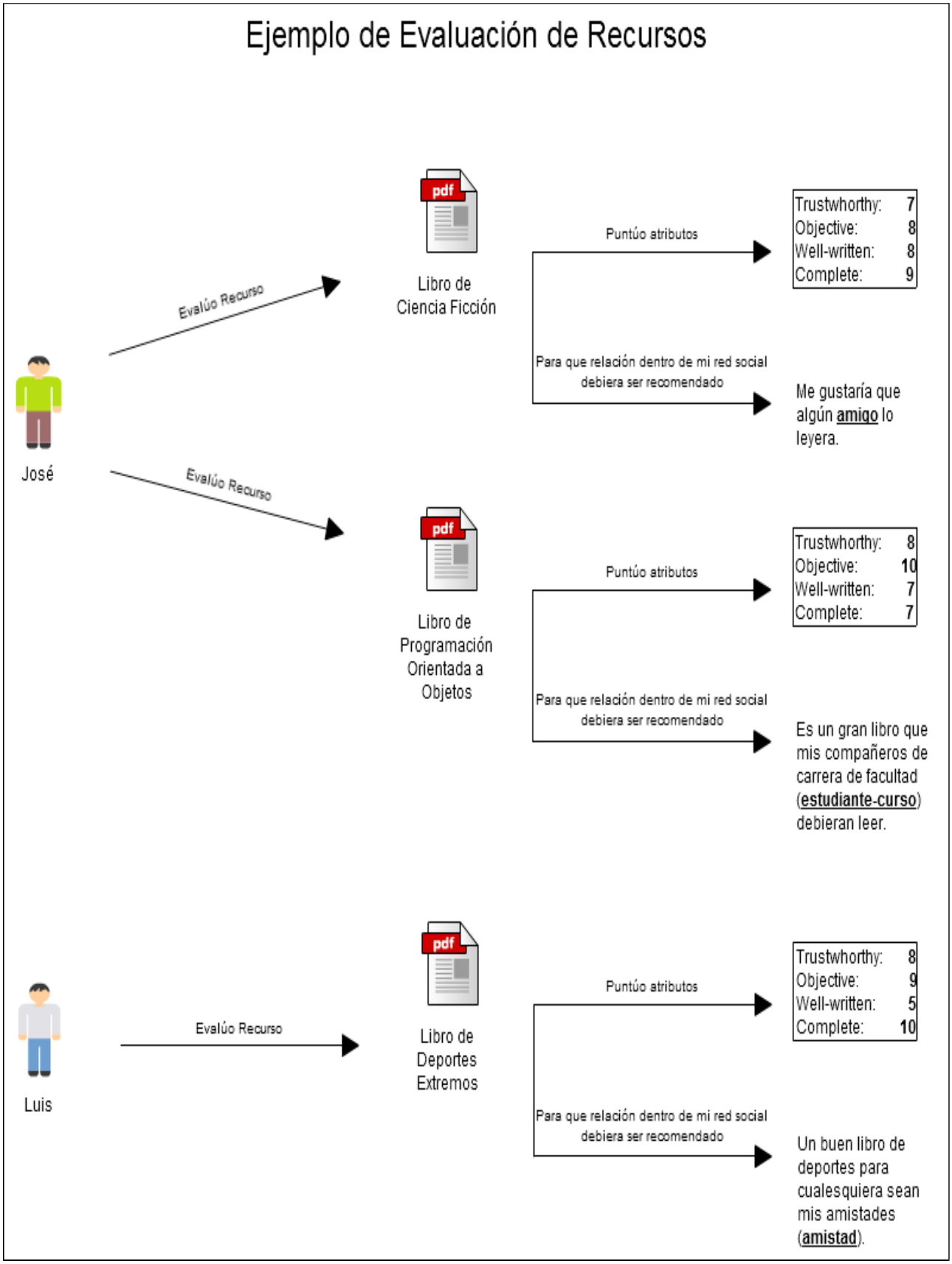


Figura 3 - Ejemplo de Evaluación de Recursos

2.3.4 Modelado de la obtención de la Puntuación sobre los Recursos

Cuando solicitamos al Sistema Recomendador, que nos brinde recomendaciones de recursos pertenecientes a alguna área temática, el mismo busca los recursos evaluados por parte de otros usuarios, donde dichos usuarios están relacionados por algún tipo de vinculo indicado sobre la red social y a cierta distancia (1 o 2), contemplando también para que relación dentro de la red social fueron evaluados esos recursos preferentemente, y procede a realizar el cálculo de las puntuaciones de los recursos evaluados, empleando el algoritmo de Friendsourcing Semántico, para luego devolvernos las recomendaciones en forma ordenada de mayor a menor, según la puntuación obtenida.

Para la obtención de recursos a recomendar, el Sistema Recomendador se basará solamente en la relación que indica el usuario dentro de la red social, el área temática y la distancia (1 o 2).

Para la obtención de las puntuaciones de los recursos a recomendar por parte del Sistema Recomendador, serán usadas la distancia entre usuarios y la relación como preferencia que tiene un recurso evaluados, además de tener en cuenta la valuación que tienen el recurso evaluados (el valor de los atributos de calidad que contiene el recurso) y los pesos seteados en el sistema para los atributos de calidad.

Se modela la obtención de las puntuaciones de los recursos evaluados para las recomendaciones de la siguiente manera:

- La puntuación de un recurso evaluado será: la valuación del recurso evaluado (calculado a partir de las calificaciones de los atributos de calidad y los pesos correspondientes) mas la aplicación de ponderadores, los cuáles utilizan: la incidencia que implica la distancia entre usuarios (si es 1 o 2) y la incidencia que tiene la relación como preferencia en un recurso evaluado.
- Se define a la incidencia que implica la distancia entre usuarios (si es 1 o 2) como $valor_1$.
- Se define a la incidencia que tiene la relación como preferencia en un recurso evaluado como $valor_2 + valor_3$.
- Se define a la puntuación que obtendrá un recurso evaluado como: $puntuación\_recurso = valuación\_recurso + valor_1 + valor_2 + valor_3$.

El se $valor_1$ se obtiene de la siguiente manera:

- Si el recurso fue evaluado por un usuario de de mi red social que se encuentra a distancia “1” de mi (relacionado directamente conmigo) -> $valor_1 = 0$. Si se

encuentra a una distancia “2” de mi (relacionado indirectamente conmigo) -> $valor_1 = -2$.

El $valor_2$ se obtiene de la siguiente manera:

- Paso1: Obtengo la lista de relaciones y subrelaciones, a partir de la relación objetivo que debiera tener del recurso, indicado cuando solicito la recomendación al sistema (la lista la obtengo utilizando una ontología de vocabulario para las relaciones).
- Paso 2: Si el sistema seleccionó un recurso evaluado en particular para recomendarse, y la relación como preferencia incluida en ese recurso evaluado está contenida en la lista del Paso1 -> $valor_2 = 1$, si no -> $valor_2 = 0$.

El $valor_3$ se obtiene de la siguiente manera:

- Paso 1: Obtengo la lista de relaciones y subrelaciones, a partir de la relación objetivo que debiera tener del recurso, indicado cuando solicito la recomendación al sistema (la lista la obtengo utilizando una ontología de vocabulario para las relaciones).
- Paso 2: Si el sistema seleccionó un recurso evaluado en particular para recomendarse, y la relación como preferencia incluida en ese recurso evaluado, coincide en su primer elemento de la lista del Paso 1 (es la cabeza de la lista) -> $valor_3 = 1$, si no -> $valor_3 = 0$.

La idea que hay detrás de escoger estos valores para $valor_1$, $valor_2$ y $valor_3$, es que si el sistema encuentra un recurso evaluado que cumple de manera muy satisfactoria con las preferencias indicadas al solicitar la recomendación el “usuario A”, lo que implica que $valor_1 + valor_2 = 2$, pero el “usuario B” que evaluó ese recurso, se vincula con el “usuario A” a través de cierta relación en la red social, pero de forma indirecta (distancia 2), implicando que $valor_3 = -2$, entonces $valor_1 + valor_2 + valor_3 = 0$. De esta manera, la obtención de la puntuación de ese recurso que realiza el sistema, será equivalente a la valuación que se calcule del recurso. Con lo anterior, lo que realizamos es que, haya un equilibrio entre, si un recurso es potencialmente bueno para recomendarse al usuario y que al mismo tiempo, ese recurso haya sido evaluado por un usuario relacionado de manera indirecta con el usuario que solicita la recomendación, favoreciendo así que la puntuación obtenida para ese recurso, no lo relegue mucho en el ranking de recursos recomendados al usuario final.

En síntesis, para obtener la puntuación de un recurso, se cumplen las siguientes

premisas:

- $-2 \leq$ puntuación_recurso ≤ 12

  lo que equivale a:

  $-2 \leq$ valuación_recurso + $valor_1$ + $valor_2$+ $valor_3 \leq 12$
- $0 \leq$ valuación_recurso ≤ 10 (ver parte anterior)
- $-2 \leq valor_1 + valor_2 + valor_3 \leq 2$, donde los valores $valor_1$, $valor_2$, $valor_3$ son los ponderadores.
- $valor_1 = \{-2 \text{ o } 0\}$
- $valor_2 = \{0 \text{ o } 1\}$
- $valor_3 = \{0 \text{ o } 1\}$

Resumiendo lo mas relevante de la obtención de la puntuación de un recurso evaluado, la siguiente imagen nos muestra:

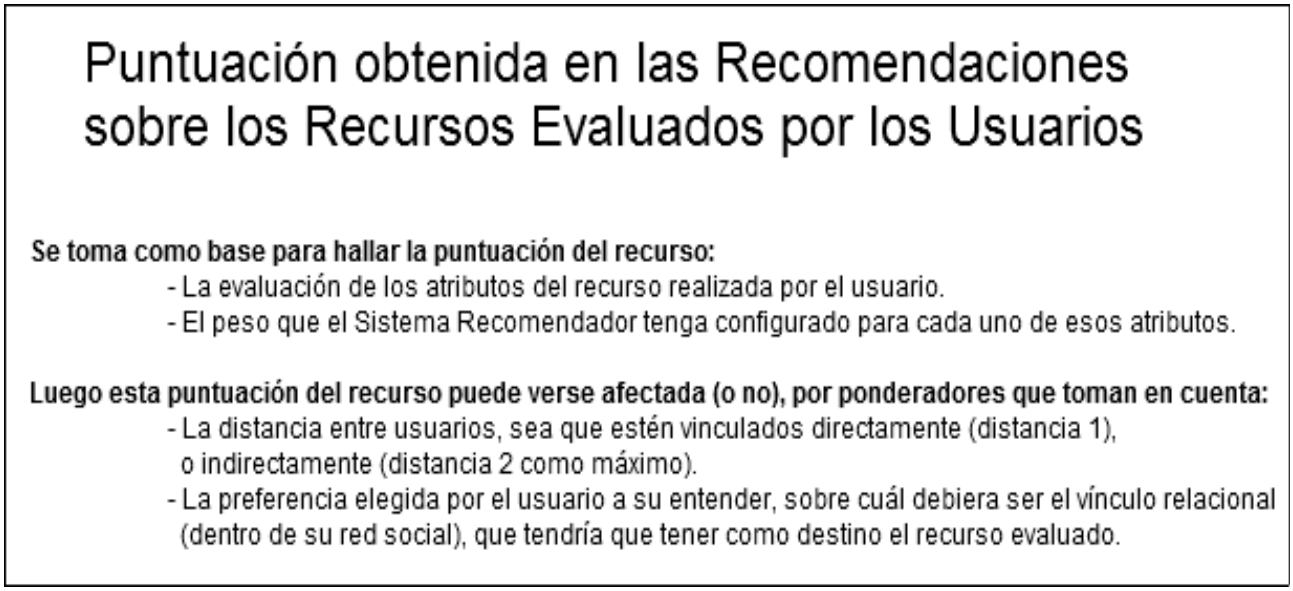
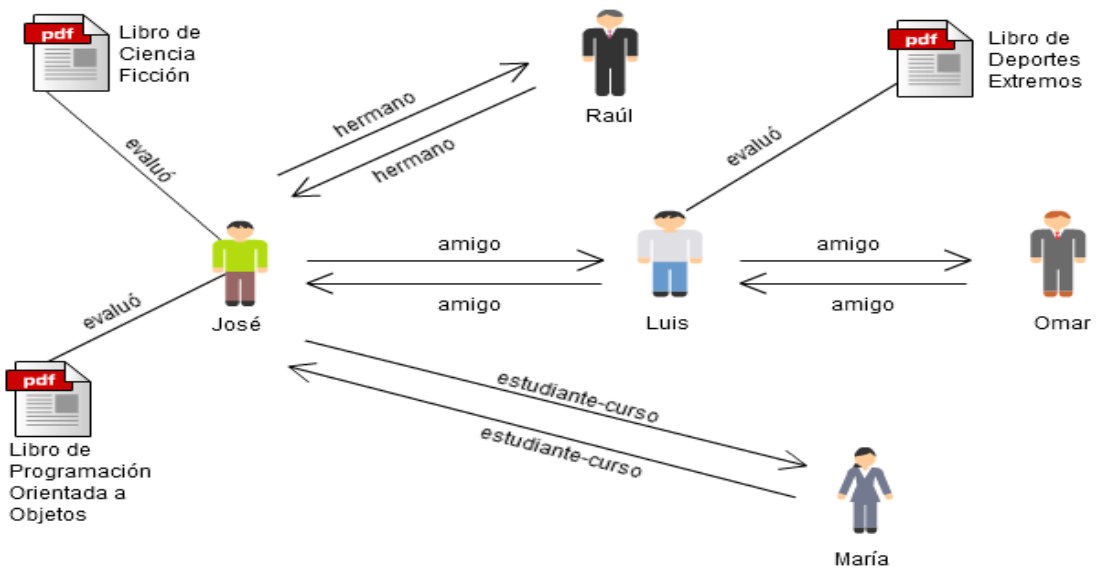


Figura 4 - Puntuación obtenida en las Recomendaciones sobre los Recursos Evaluados por los Usuarios

Un ejemplo de la obtención de recomendaciones para un usuario dentro de una red social por defecto, incluidos los valores obtenidos en las puntuaciones de los recursos (no se muestra cómo se calculan, sino el valor que devuelve el sistema), sería el siguiente (ver ejemplo presentado en Figura 3, sobre evaluación de recursos para una mejor comprensión):

Ejemplo de Obtención de Recomendaciones



---

1)
María solicita al Sistema Recomendador: Libros evaluados por sus compañeros de facultad (relación estudiante-curso), que están directamente relacionados con ella (distancia 1) y que preferentemente esos libros hayan sido evaluados para estudiantes de la facultad (relación estudiante-curso).

Resultados:	Recurso	Puntuación
	Libro de Programación Orientada a Objetos	10
	Libro de Ciencia Ficción	8

---

2)
Luis solicita al Sistema Recomendador: Libros evaluados por sus amigos (relación amigo), que están directamente relacionados con él (distancia 1) y que preferentemente esos libros hayan sido evaluados para los amigos (relación amigo).

Resultados:	Recurso	Puntuación
	Libro de Ciencia Ficción	10
	Libro de Programación Orientada a Objetos	8

---

3)
Raúl solicita al Sistema Recomendador: Libros evaluados por su familia (relación Familiar), que están directamente relacionados con él (distancia 1) y que preferentemente esos libros hayan sido evaluados para el ámbito de trabajo (relación Laboral).

Resultados:	Recurso	Puntuación
	Libro de Programación Orientada a Objetos	8
	Libro de Ciencia Ficción	8

---

4)
Omar solicita al Sistema Recomendador: Libros evaluados por sus amigos y por los amigos de sus amigos (relación amigo), que están directamente o indirectamente relacionados con él (distancia 2), y que preferentemente esos libros hayan sido evaluados para las amistades (relación Amistad).

Resultados:	Recurso	Puntuación
	Libro de Deportes Extremos	10
	Libro de Ciencia Ficción	7
	Libro de Programación Orientada a Objetos	6

Figura 5 - Ejemplo de Obtención de Recomendaciones

2.3.5 Modelado del Vocabulario de las Relaciones y de las Áreas Temáticas

Las relaciones entre usuarios dentro de una red social y las áreas temáticas a las que pueda pertenecer un recurso, deben ser modeladas de alguna forma, utilizando alguna estructura, que permita incluir vocabulario que defina de forma semántica a las relaciones dentro de una red social y a las áreas temáticas a las que pueda pertenecer un recurso, y poder hacer consultas sobre las mismas. Esta estructura debe ser fácilmente escalable, con el fin de poderle realizar modificaciones (enriqueciendo el vocabulario) tanto en lo que se refiere a relaciones como a áreas temáticas. Es por ello, que de entre varias posibles formas de modelado de las mismas (por ejemplo usando estructuras de arreglos, listas, árboles, archivo de texto, etc.), y aprovechando que para el proyecto usamos varias de los componentes que nos provee la arquitectura de la Web Semántica, tanto el modelado del vocabulario de la relaciones como el modelado del vocabulario de las áreas temáticas, se realizan utilizando ontologías de vocabulario, para establecer la jerarquía de términos incluidos tanto sea para relaciones como para áreas temáticas.

Ambas ontologías de vocabulario se desarrollan utilizando la herramienta Protégé.

A continuación se detallan como se realizan las mismas.

2.3.5.1 Vocabulario de las Relaciones

Existen variados recursos en la web (ontología de vocabularios) para describir relaciones entre personas. Para este proyecto se plantea realizar una ontología de vocabulario para las relaciones entre personas tomando como referencia la siguiente ontología de vocabulario[20] (la misma se encuentra en idioma inglés). Pero la misma no contempla en algunos aspectos, vocabulario mas específico, por ejemplo para definir relaciones que se dan dentro de un ámbito académico, o para definir una relación de Tío – Sobrino por ejemplo. En ella se usan relaciones mas generales (en inglés), como ser “Conoce a”, o “Descendiente de”, “Ancestro de”, etc.. Además provee vocabulario extra, que no necesitamos para nuestra modelo de vocabulario, como ser “Enemigo de”, “Antagonista de”, etc..

Entonces a partir de ésta, y contemplando nuestras necesidades de cuál debe ser el vocabulario de las relaciones a usar, se desarrolla una ontología de vocabulario, que sea consistente con la lengua española y que contemple las siguientes grupos de relaciones entre personas:

- **Amistad:** para las relaciones entre personas que son amigos o amigas entre sí, o que tienen relaciones de amistad de mayor confianza entre sí (mejores amigos que incluye a mejor amigo o mejor amiga).

- **Académico:** para las relaciones entre personas que se dan en el ámbito estudiantil y académico.
- **Familiar:** para las relaciones entre personas que poseen un vínculo familiar entre sí, el mismo puede ser directo, político o religioso.
- **Laboral:** para las relaciones que se establecen entre personas en el ámbito laboral donde una persona desarrolle su trabajo.

La uri creada para las relaciones es: "http://www.semanticweb.org/rwsf/ontologies/2015/4/relaciones#nombreRelacion", donde nombreRelacion es el posible nombre de la relación vincular que se tenga entre dos usuarios dentro de la red social, siendo "http://www.semanticweb.org/rwsf/ontologies/2015/4/relaciones#" el espacio de nombres definido para el vocabulario de las relaciones, donde éste es el que se estableció al crear el mismo utilizando el editor de ontologías (Protégé).

2.3.5.2 Vocabulario de las Áreas Temáticas

Existen variadas ontología de vocabularios, de gran calidad y específicas para describir recursos en la web[21], ya sea música, libros, películas, etc. y otras mas utilizadas en comercio electrónico(e-commerce) por ejemplo. Pero en este caso se busca una ontología de vocabulario, que contemple e incluya, a la mayor cantidad de posibles áreas temáticas a las que pertenece un recurso y que utilicen nuestra lengua española.

Es así, que se procedió a desarrollar una ontología de vocabulario que abarca la mayoría de las áreas temáticas a las que pueda pertenecer un recurso.

Para los casos de Películas y de Libros, que son las áreas temáticas que utilizamos para los recursos en los casos de prueba de nuestro Sistema Recomendador, para describir los mismos, se tomaron como referencia algunas de las ontologías de vocabulario que describen a las mismas en Linked Open Vocabularies (LOV) [21] como ser “The DBpedia Ontology Film” para descripciones de películas e “ISBD elements” para descripciones bibliográficas.

La uri creada para las áreas temáticas es: "http://www.semanticweb.org/rwsf/ontologies/2015/4/area_tematica#nombreAreaTematica", donde nombreAreaTematica es el nombre del área temática, en la cual un recurso de la red social pueda pertenecer, siendo

"http://www.semanticweb.org/rwsf/ontologies/2015/4/area_tematica#" el espacio de nombres definido para el vocabulario de las relaciones, donde éste es el que se estableció al crear el mismo utilizando el editor de ontologías (Protégé).

2.3.6 Descripción del Algoritmo de Recomendación (Friendsourcing Semántico)

A continuación se expone el funcionamiento que emplea nuestro algoritmo de Friendsourcing Semántico para obtener las recomendaciones, mostrando los datos de entrada, la descripción del algoritmo y la salida obtenida.

También se muestran ejemplos para facilitar la comprensión del mismo.

Datos de Entrada:

- Identificador de usuario que pide las recomendaciones (idUsuario).
- Relación vincular del usuario en la red social, donde se buscarán los recursos a recomendar (relacionPersona).
- Distancia, del usuario que pide las recomendaciones con respecto a los demás usuarios en la red social (distancia).
- Relación vincular que debería tener un recurso evaluado, indicado por parte de los usuarios cuando realizaron la evaluación de un recurso (relacionEnRecursoEvaluado).
- Área temática a la que deben pertenecer los recursos que el usuario solicita (areaTematica).

Algoritmo:

- Obtener en una lista, el área temática y las subáreas temáticas de la misma que deben tener los recursos a buscar utilizando el dato de entrada areaTematica (listaAreaTematica).
- Obtener lista de usuarios de la red social relacionados con el usuario que pide la recomendación, usando los datos de entrada idUsuario, realacionPersona y distancia (listaRelacionesUsuario).
- Para cada usuario en la lista de usuarios listaRelacionesUsuario:

  - Para cada área temática en la lista de áreas temáticas listaAreaTematica:
    - Obtener la lista de los recursos evaluados por los usuarios incluidos en listaRelacionesUsuario, que cumplan que el área temática del recurso evaluado se incluya en la lista de áreas temáticas listaAreaTematica (listaRecurosEval).

- Obtener los atributos de calidad para la red social (listaAtributos).
- Obtener en una lista, la relación y las subrelaciones de la misma, que el recurso debe tener como destinatario, usando el dato de entrada relacionEnRecursoEvaluado (listaRelacionEsperadaEnRecursoEval).
- Para cada recurso evaluado en la lista de los recursos evaluados listaRecurosEval:
  - Obtener lista de recursos a recomendar. **(*)**

(*) Donde cada recurso a recomendar contiene::
- Datos del recurso.
- Datos del usuario que evaluó el recurso.
- Puntuación obtenida, empleando:
  - La valuación del recurso, usando:
    - Los atributos de calidad en listaAtributos, los pesos de los atributos de calidad (seteados en el sistema), las calificaciones de los atributos de calidad (en el recurso evaluado).
  - Los ponderadores (valor1, valor2 y valor3), usando:
    - Distancia a la que está el usuario que evaluó el recurso (valor1), y la lista listaRelacionEsperadaEnRecursoEval y la relacionEnRecursoEvaluado (valor2 y valor3).

Datos de Salida:

- Lista de recursos recomendados ordenada por puntuación (de mayor a menor), y sin recursos repetidos.

2.3.7 Ejemplos concretos de uso del algoritmo de Friendsourcing Semántico

A continuación se muestran ejemplos de cómo se obtienen las recomendaciones y sus puntuaciones, empleando el algoritmo de Friendsourcing Semántico. Para ello se emplean los ejemplos mostrados en la Figura 3 y en la Figura 5.
Se desarrollan dos de los cuatro ejemplos contenidos en la Figura 5 (1 y 4), los ejemplos 2 y 3 se calculan siguiendo el mismo procedimiento.

1) María solicita al Sistema Recomendador:
Libros evaluados por sus compañeros de facultad (relación estudiante-curso), que están directamente relacionados con ella (distancia 1) y que preferentemente esos libros hayan sido evaluados para estudiantes de la facultad (relación estudiante-curso).

- Lista áreas temáticas de Libros

{Libro, LibroDeAprendizaje, LibroDeCiencias, LibroDeHistoria, ..., LibroDeProgramacion, LibroDeProgramacionC, LibroProgramacionJava, ...}

- Lista usuarios en relación estudiante-curso(o una subrelación de estudiante-curso, en este caso no las hay), a distancia 1:

{José}

- Obtengo los recursos evaluados, por los usuarios de la lista de usuarios, que hayan evaluado algún recursos cuya área temática esté incluida en la lista de áreas temáticas:

{Libro de Ciencia Ficción, Libro de Programación Orientada a Objetos}

- Obtengo lista atributos de la red por defecto:

{Trustwhorthy, Objective, Complete, Well-written}

- Obtengo lista de relaciones y de subrelaciones deseables en los recursos evaluados, a partir de dato de entrada estudiante-curso:

{estudiante-curso}

- Para cada uno de los recursos evaluados en la lista, calculo su puntuación (supongamos los pesos de los atributos en iguales proporciones):

– **Libro de Ciencia Ficción**

Atributo	Peso	Valoración
Trustwhorthy	0.25	7
Objective	0.25	8
Well-written	0.25	8
Complete	0.25	9

- Calculo valuación del recurso: 0.25*7 + 0.25*8 + 0.25*8 + 0.25*9 = **8**
- Calculo ponderadores semánticos:
  - *valor1*:
    Distancia de usuario = 1, entonces valor1 = **0**
  - *valor2*:
    Relación objetivo del recurso que estoy evaluando = amigo (dato obtenido del recurso evaluado)
    ¿Está incluida en lista de relaciones deseables que deberían tener los recursos evaluados ({estudiante-curso})?
    No, entonces valor2 = **0**
  - *valor3*:
    Relación objetivo del recurso que estoy evaluando = amigo (dato obtenido del recurso evaluado)
    ¿Es el primer elemento (cabeza de lista) en lista de relaciones deseables que deberían tener los recursos evaluados ({estudiante-curso})?
    No, entonces valor3 = **0**

- Puntuación del recurso = 8 + 0 + 0 + 0 = **8**

– **Libro de Programación Orientada a Objetos**

Atributo	Peso	Valoración
Trustwhorthy	0.25	8
Objective	0.25	10
Well-written	0.25	7
Complete	0.25	7

- Calculo valuación del recurso: 0.25*8 + 0.25*10 + 0.25*7 + 0.25*7 = **8**
- Calculo ponderadores semánticos:
  - *valor1*:

Distancia de usuario = 1, entonces valor1 = **0**

- *valor2*:
  Relación objetivo del recurso que estoy evaluando = estudiante-curso (dato obtenido del recurso evaluado)
  ¿Está incluida en lista de relaciones deseables que deberían tener los recursos evaluados ({estudiante-curso})?
  Si, entonces valor2 = **1**
- *valor3*:
  Relación objetivo del recurso que estoy evaluando = estudiante-curso (dato obtenido del recurso evaluado)
  ¿Es el primer elemento (cabeza de lista) en lista de relaciones deseables que deberían tener los recursos evaluados ({estudiante-curso})?
  Si, entonces valor3 = **1**

- Puntuación del recurso = 8 + 0 + 1 + 1 = **10**

- Resultado (ordenado de mayor a menor según puntuación obtenida):

Libro de Programación Orientada a Objetos	**10**
Libro de Ciencia Ficción	**8**

4) Omár solicita al Sistema Recomendador:
Libros evaluados por sus amigos y por los amigos de sus amigos (relación amigo), que están directamente o indirectamente relacionados con él (distancia 2), y que preferentemente esos libros hayan sido evaluados para las amistades (relación Amistad).

- Lista áreas temáticas de Libros:

{Libro, LibroDeAprendizaje, LibroDeCiencias, LibroDeHistoria, ..., LibroDeProgramacion, LibroDeProgramacionC, LibroProgramacionJava, ...}

- Lista usuarios en relación amigo (o una subrelación de amigo, en este caso no las hay) a distancia 1 o 2:

{Luis, José}

- Obtengo los recursos evaluados, por los usuarios de la lista de usuarios, que hayan evaluado algún recursos cuya área temática esté incluida en la lista de áreas temáticas:

{Libro de Deportes Extremos, Libro de Ciencia Ficción, Libro de Programación Orientada a Objetos}

- Obtengo lista atributos de la red por defecto:

{Trustwhorthy, Objective, Complete, Well-written}

- Obtengo lista de relaciones y de subrelaciones deseables en los recursos evaluados, a partir de dato de entrada Amistad:

{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}

- Para cada uno de los recursos evaluados en la lista, calculo su puntuación(supongamos los pesos de los atributos en iguales proporciones):

– **Libro de Deportes Extremos**

Atributo	Peso	Valoración
Trustwhorthy	0.25	8
Objective	0.25	9
Well-written	0.25	5
Complete	0.25	10

- Calculo valuación del recurso: 0.25*8 + 0.25*9 + 0.25*5 + 0.25*10 = **8**
- Calculo ponderadores semánticos:
  - *valor1*:
    Distancia de usuario = 1, entonces valor1 = **0**
  - *valor2*:
    Relación objetivo del recurso que estoy evaluando = Amistad (dato obtenido del recurso evaluado)
    ¿Está incluida en lista de relaciones deseables que deberían tener los recursos evaluados ({Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo})?
    Si, entonces valor2 = **1**
  - *valor3*:
    Relación objetivo del recurso que estoy evaluando = Amistad (dato

obtenido del recurso evaluado)
¿Es el primer elemento (cabeza de lista) en lista de relaciones deseables que deberían tener los recursos evaluados ({Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo})?
Si, entonces valor3 = **1**

- Puntuación del recurso = 8 + 0 + 1 + 1 = **10**

– **Libro de Ciencia Ficción**

Atributo	Peso	Valoración
Trustwhorthy	0.25	7
Objective	0.25	8
Well-written	0.25	8
Complete	0.25	9

- Calculo valuación del recurso: 0.25*7 + 0.25*8 + 0.25*8 + 0.25*9 = **8**
- Calculo ponderadores semánticos:
  - *valor1*:
    Distancia de usuario = 2, entonces valor1 = **-2**
  - *valor2*:
    Relación objetivo del recurso que estoy evaluando = amigo (dato obtenido del recurso evaluado)
    ¿Está incluida en lista de relaciones deseables que deberían tener los recursos evaluados ({Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo})?
    Si, entonces valor2 = **1**
  - *valor3*:
    Relación objetivo del recurso que estoy evaluando = amigo (dato obtenido del recurso evaluado)
    ¿Es el primer elemento (cabeza de lista) en lista de relaciones deseables que deberían tener los recursos evaluados ({Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo})?
    No, entonces valor3 = **0**

- Puntuación del recurso = 8 + (-2) + 1 + 0 = **7**

– **Libro de Programación Orientada a Objetos**

Atributo	Peso	Valoración
Trustwhorthy	0.25	8
Objective	0.25	10
Well-written	0.25	7
Complete	0.25	7

- Calculo valuación del recurso: 0.25*8 + 0.25*10 + 0.25*7 + 0.25*7 = **8**
- Calculo ponderadores semánticos:
  - *valor1*:
    Distancia de usuario = 2, entonces valor1 = **-2**
  - *valor2*:
    Relación objetivo del recurso que estoy evaluando = estudiante-curso (dato obtenido del recurso evaluado)
    ¿Está incluida en lista de relaciones deseables que deberían tener los recursos evaluados ({Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo})?
    No, entonces valor2 = **0**
  - *valor3*:
    Relación objetivo del recurso que estoy evaluando = estudiante-curso (dato obtenido del recurso evaluado)
    ¿Es el primer elemento (cabeza de lista) en lista de relaciones deseables que deberían tener los recursos evaluados ({Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo})?
    No, entonces valor3 = **0**

- Puntuación del recurso = 8 + (-2) + 0 + 0 = **6**

- Resultado (ordenado de mayor a menor según puntuación obtenida):

Libro de Deportes Extremos	**10**
Libro de Ciencia Ficción	**7**
Libro de Programación Orientada a Objetos	**6**

3 – Prototipo del Sistema Recomendador con Friendsourcing Semántico.

A continuación se detallan los elementos involucrados en la implementación del recomendador semántico, esto incluye el entorno de trabajo usado, el lenguaje de programación empleado y que herramientas fueron utilizadas para alcanzar tal fin. Se muestran detalladamente las fuentes de información necesarias para el recomendador, así como las capas que lo conforman: componente lógica, interfaz gráfica y persistencia.

3.1 Entorno de trabajo, lenguaje de programación y herramientas utilizadas.

El Sistema Recomendador con Friendsourcing Semántico fue desarrollado en el lenguaje de programación Java, dado que es un lenguaje en el cual ambos miembros del grupo estamos acostumbrados a lo largo de la carrera en utilizar el mismo. Se investigaron herramientas de la web semántica que nos ayudaran con el desarrollo, como ser editores de ontologías, y que además se pudieran utilizar con el lenguaje de programación Java, como ser librerías para poder utilizar las ontologías creadas y lenguajes de consulta sobre las mismas.

Como entorno de trabajo se utilizó Eclipse (en sus versiones Kepler y Galileo), también por estar familiarizados con el mismo a lo largo de la carrera.

Para el desarrollo de las ontologías se utilizó la herramienta Protégé, dado que es un editor simple y muy intuitivo para crear ontologías.

Para poder utilizar las ontologías dentro del proyecto, se utilizó Apache Jena(librerías) de forma de poder (desde la componente lógica del recomendador) interactuar con las mismas, mediante consultas sobre ellas, empleando el lenguaje SPARQL.

Para la persistencia de los datos, se empleó una base de datos MySQL 5 (v. 5.6), así como diversas herramientas y elementos que conforman dicho software (Conector MySQL para Java, MySQL Workbench, etc.).

Para la interfaz gráfica se utilizó JSF, con el servidor de aplicaciones JBoss. El cliente JSF usado es Primefaces. Además para crear el estilo y formato de las páginas web se usó CSS3.

3.2 Fuentes de Información

El Sistema Recomendador y por ende, el algoritmo de recomendación de Friendsourcing Semántico, necesitan ser alimentados (previamente a su ejecución con el fin de obtener las recomendaciones de forma correcta) mediante dos tipos de fuentes de información: por un lado las fuentes de información que están involucradas en la componente lógica, y por otro lado las fuentes de información que aplican a la persistencia.

En cuanto a las fuentes de información a nivel de la componente lógica son necesarias:

- La Red Social del usuario al que se le brindarán las recomendaciones. La misma está diseñado utilizando XML (RedSocial.xml).
  Esta Red Social debe ser validada previamente (se valida el archivo RedSocial.xml), para ello fue diseñado el validador utilizando XML-Schema (ValidadorXMLRedSocial.xsd).
- Una ontología de vocabulario que defina semánticamente las relaciones entre usuarios dentro de una red social. La misma fue diseñada en Protégé. (relaciones.owl).
- Una ontología de vocabulario que defina semánticamente el área temática a la que pueda pertenecer un recurso. La misma fue diseñada en Protégé. (area_tematica.owl).

En cuanto a las fuentes de información a nivel de la persistencia son necesarias:

- Los recursos que podrán evaluar los usuarios. Los mismos está diseñados utilizando XML (Recursos.xml, Peliculas.xml, Libros.xml, etc.).
  Estos recursos deben ser validados previamente (se valida el archivo .xml), para ello fue diseñado el validador utilizando XML-Schema (ValidadorXMLRecursos.xsd).
- La configuración de los pesos de los atributos de calidad de los recursos. La misma está diseñada utilizando XML (Atributos.xml).
  Estos recursos deben ser validados previamente (se valida el archivo Atributos.xml), para ello fue diseñado el validador utilizando XML-Schema (ValidadorXMLAtributos.xsd).

Las dos fuentes de información anteriores son necesarias para que el administrador del sistema recomendador pueda realizar operaciones sobre el mismo.

Adicionalmente para el caso en que se desee testear el sistema recomendador, se podrán ingresar recursos evaluados por los usuarios a través de una archivo xml. El mismos será verificado por un validador xml-schema como se ha venido realizado a lo largo del proyecto con los archivos xml que se ingresan al sistema. Claramente esta acción también será realizada exclusivamente por un administrador.

Otra forma de cargar las fuentes de información a nivel de la persistencia (operando directamente sobre la base de datos):

- Los atributos de calidad de los recursos. Pueden verse como los metadatos de los recursos. Dependiendo del contexto en que se esté(Facebook, Google+, etc.), los atributos de calidad serán distintos. (Pasos: BaseRecomendador.sql -> TablaAtributo.sql -> CargaAtributos.sql).
- Los recursos, para que los usuarios los puedan evaluar y para ser devueltos en las recomendaciones (Pasos: BaseRecomendador.sql -> TablaRecurso.sql -> CargaRecursos.sql).
- Los recursos evaluados. Son los recursos ya evaluados por los usuarios de la red social. Utilizado para mitigar el problema de “arranque en frío” en las recomendaciones. (Pasos: BaseRecomendador.sql -> TablaEvaluacionRecurso.sql -> CargaEvaluaciones.sql).

Lo anterior (archivos.sql) debe ser realizado por un administrador de la base de datos, ya que las operaciones se realizan directamente sobre la misma. Además hay que tener extremo cuidado cuando se opera directamente sobre la base de datos y no a través del Sistema Recomendador, porque puede dejar inconsistencias en el mismo y afectar su buen funcionamiento.

Los archivos de creación de la base de datos y de las tablas correspondientes (BaseRecomendador.sql, TablaAtributo.sql, TablaRecurso.sql, TablaEvaluacionRecurso.sql y TablaConsultaUsuario.sql) se detallan mas adelante en la sección destinada a la Persistencia del sistema recomendador (sección 4.4).

Como se detalló anteriormente, las fuentes de información a nivel de la persistencia pueden ser ingresadas:

- a través del sistema recomendador (por parte del administrador del sistema recomendador),
- o a través de la manipulación directa sobre la base de datos del recomendador (por parte de una administrador de la base de datos).

3.2.1 Fuentes de información a nivel de la componente lógica

A continuación se detallan cada una de las fuentes de información a nivel de la componente lógica.

3.2.1.1 RedSocial.xml

Este archivo contiene información de la Red Social a la cual pertenece el usuario que solicita las recomendaciones. En el mismo se indica el dominio o nombre de la red social, así como los usuarios que la componen. Y para cada usuario en la red social, se indica un identificador de usuario, el nombre del usuario y una lista de usuarios con los que está relacionado en la red. Para cada una de estas relaciones que el usuario contiene en la red se indica el identificador del usuario, el nombre de usuario y el vínculo que lo relaciona al mismo. Cabe indicar que se asume que la red social es consistente en el sentido de que si existe una relación entre dos usuarios, la misma debe ser en ambas direcciones y estar explicitada en el archivo de la red social, es decir a modo de ejemplo que si A(masculino) es primo de B(femenino), debe existir la relación B es prima de A, lo mismo se asume por ejemplo con las relaciones transitivas.

A modo de ejemplo un archivo xml conteniendo la red social sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<RedSocial>
     <Dominio>www.example.com</Dominio>
     <Usuario>
          <Identificador>diego.bastiani</Identificador>
          <Nombre>Diego Bastiani</Nombre>
          <ListaRelacionamiento>
               <Vinculo>
                    <Identificador>marcos.suiffet</Identificador>
                    <Nombre>Marcos Suiffet</Nombre>
                    <TipoVinculo>amigo</TipoVinculo>
               </Vinculo>
               <Vinculo>
                    <Identificador>marcos.suiffet</Identificador>
                    <Nombre>Marcos Suiffet</Nombre>
                    <TipoVinculo>estudiante-curso</TipoVinculo>
               </Vinculo>
               <Vinculo>
                    <Identificador>ernesto.perez</Identificador>
```

```
                    <Nombre>Ernesto Perez</Nombre>
                    <TipoVinculo>jefe</TipoVinculo>
               </Vinculo>
               <Vinculo>
                    <Identificador>pedro.martnez</Identificador>
                    <Nombre>Pedro Martinez</Nombre>
                    <TipoVinculo>mejorAmigo</TipoVinculo>
               </Vinculo>
               <Vinculo>
                    <Identificador>ana.sosa</Identificador>
                    <Nombre>Ana Sosa</Nombre>
                    <TipoVinculo>prima</TipoVinculo>
               </Vinculo>
               .........
               .........
               .........
          </ListaRelacionamiento>
     </Usuario>
     <Usuario>
          <Identificador>marcos.suiffet</Identificador>
          <Nombre>Marcos Suiffet</Nombre>
          <ListaRelacionamiento>
               <Vinculo>
                    <Identificador>diego.bastiani</Identificador>
                    <Nombre>Diego Bastiani</Nombre>
                    <TipoVinculo>amigo</TipoVinculo>
               </Vinculo>
               <Vinculo>
                    <Identificador>diego.bastiani</Identificador>
                    <Nombre>Diego Bastiani</Nombre>
                    <TipoVinculo>estudiante-curso</TipoVinculo>
               </Vinculo>
               <Vinculo>
                    <Identificador>martin.costa</Identificador>
                    <Nombre>Martin Costa</Nombre>
                    <TipoVinculo>empleado</TipoVinculo>
               </Vinculo>
               .........
               .........
               .........
          </ListaRelacionamiento>
     </Usuario>
     .........
     .........
     .........
</RedSocial>
```

3.2.1.1.1 ValidadorXMLRedSocial.xsd

Si bien no es una fuente de información es relevante porque valida una fuente de información muy importante como lo es la Red Social. Este archivo valida que el archivo RedSocial.xml esté bien estructurado. Esto es que contenga las etiquetas adecuadas y que los campos de dichas etiquetas contengan datos correctos (ej: no sean vacíos, contenga caracteres, etc.), aplicándose diversas restricciones sobre los mismos.

Con esto nos aseguramos que la Red Social esté formada adecuadamente y sea validada para ser utilizada en el recomendador semántico.

El archivo empleado para tal fin es el siguiente:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="RedSocial">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Dominio">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:pattern value=".*[^\s].*"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
   <xs:element name="Usuario" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Identificador">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:minLength value="1"/>
         <xs:pattern value=".*[^\s].*"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
      <xs:element name="Nombre">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:minLength value="1"/>
         <xs:pattern value=".*[^\s].*"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
      <xs:element name="ListaRelacionamiento" minOccurs="0">
       <xs:complexType>
```

```
        <xs:sequence>
         <xs:element name="Vinculo" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Identificador">
             <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:minLength value="1"/>
               <xs:pattern value=".*[^\s].*"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:element>
            <xs:element name="Nombre">
             <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:minLength value="1"/>
               <xs:pattern value=".*[^\s].*"/>
              </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="TipoVinculo">
             <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:minLength value="1"/>
               <xs:pattern value=".*[^\s].*"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

</xs:schema>
```

3.2.1.2 Relaciones.owl

Este archivo contiene la ontología de vocabulario para los diferentes tipos de relaciones que se puedan dar dentro de una red social. La misma está realizada de forma jerárquica, yendo de la relación mas general a la mas particular, definida a partir de clases y subclases de éstas. Como se verá a continuación, existen en esta

ontología 4 niveles jerárquicos, donde el nivel 1 es definido para la clase “Relación”.

Fueron definidas las siguientes subclases para la clase “Relación”(nivel 2):

En este sentido vemos que las subclases de Académico, Amistad, Familiar y Laboral son las siguientes (nivel 3):

Por último existen otras subclases (nivel 4) para algunas de las clases del nivel 3 como se muestra en la imagen anterior. Por ejemplo la clase “mejoresAmigos”(nivel 3) tiene como subclases a “mejorAmigo” y a “mejorAmiga” (ambas del nivel 4). Por lo general las clases que se encuentran en el nivel 4, son definiciones masculino y femenino ambas en singular, de la clase que preceden.

Protégé que es el software con el cual se generaron las ontologías de vocabulario, no resuelve bien el carácter “ñ”, por lo cual se sustituyó a cuñados por cuniados, cuñada, por cuniada, cuñado por cuniado, concuñados por concuniados, concuñado por concuniado, concuñada por concuniada, y compañeroDeTrabajo por colegaDeTrabajo, las cuales eran las que se habían propuesto en un principio.

3.2.1.3 AreaTematica.owl

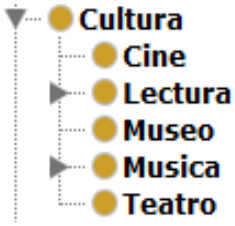
Este archivo contiene la ontología de vocabulario para los diferentes tipos de áreas temáticas a las cuáles puede pertenecer un recurso. La misma está realizada de forma jerárquica, yendo de la relación mas general a la mas particular, definida a partir de clases y subclases de éstas. Como se verá a continuación, existen en esta ontología 7 niveles jerárquicos, donde el nivel 1 es definido para la clase “AreaTematica”.

Fueron definidas las siguientes subclases para la clase “AreaTematica”(nivel 2):

A modo de muestra se exponen como se estructuran las siguientes subclases de la clase AreaTematica:

- Alimentación:

- Cultura:



- Lectura:

- Música

- Diversión y Entretenimiento:

- Educación:

3.2.2 Fuentes de información a nivel de la persistencia

A continuación se detallan cada una de las fuentes de información a nivel del componente de la persistencia.

Para los archivos .sql se establece como precondición que deben estar corridos los scripts que crean la base de datos del recomendador (baseRecomendador.sql), así como los scripts que crean las tablas correspondientes (tablaRecurso.sql, tablaAtributo.sql y tablaEvaluacionRecurso.sql).

Como se verá a continuación, se distinguen dos formas de utilizar las fuentes de información a nivel de la persistencia, que obtienen el mismo efecto (por ejemplo para la carga de recursos), pero son ingresadas de maneras diferentes al sistema, ya sea que las ingresa el administrador del Sistema Recomendador, o que las ingresa el administrador de la base de datos.

3.2.2.1 Recursos.xml

Este archivo contiene los recursos (a ser evaluados por los usuarios de la red social) que se cargarán en la tabla de recursos de la base de datos del recomendador (lo realiza el administrador del Sistema Recomendador).

A modo de ejemplo un archivo xml conteniendo recursos (por ejemplo películas) sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<Recursos>
      <Recurso>
            <Titulo>Forrest Gump</Titulo>
            <Descripcion>Titulo original: Forrest Gump | Origen: USA | Año: 1994
| Duracion: 2h 22min | Clasificacion: PG-13 | Director: Robert Zemeckis |
Reparto: Tom Hanks, Robin Wright Penn, Gary Sinise, Mykelti Williamson, Sally
Field, Rebecca Williams, Michael Conner Humphreys, Harold G. Herthum, Haley Joel
Osment, George Kelly, Bob Penny, John Randall, Sam Anderson, Margo Moorer, Ione
M. Telech, Christine Seabrook | Sinopsis: Forrest Gump (Tom Hanks) es un chico
que sufre un cierto retraso mental. A pesar de todo, gracias a su tenacidad y a
su buen corazón será protagonista de acontecimientos cruciales de su país.
Jenny, su gran amor desde la infancia, será la persona más importante de su
vida.</Descripcion>
            <URL>http://www.imdb.com/title/tt0109830/</URL>
            <AreaTematica>PeliculaDramatica</AreaTematica>
      </Recurso>
      <Recurso>
            <Titulo>2001: Odisea del espacio</Titulo>
            <Descripcion>Titulo original: 2001: A Space Odyssey | Origen: UK |
Año: 1968 | Duracion: 2h 29min | Clasificacion: G | Director: Stanley Kubrick |
Reparto: Keir Dullea, Gary Lockwood, William Sylvester, Daniel Richter, Douglas
```

```
Rain, Laonard Rossiter, Margaret Tyzack, Robert Beatty, Sean Sullivan, Frank
Miller, Penny Brahms, Alan Gilfford, Vivian Kubrick | Sinopsis: Hace millones de
años, antes de la aparición del "homo sapiens", unos primates descubren un
monolito que los conduce a un estadio de inteligencia superior. Millones de años
después, otro monolito, enterrado en una luna, despierta el interés de los
científicos. Por último, durante una misión de la NASA, HAL 9000, una máquina
dotada de inteligencia artificial, se encarga de controlar todos los sistemas de
una nave espacial tripulada.</Descripcion>
            <URL>http://www.imdb.com/title/tt0062622/</URL>
            <AreaTematica>PeliculaDeCienciaFiccion</AreaTematica>
      </Recurso>
      ....
      ....
      ....
      <Recurso>
            <Titulo>Rescatando al soldado Ryan</Titulo>
            <Descripcion>Titulo original: Saving Private Ryan | Origen: USA |
Año: 1998 | Duracion: 2h 49min | Clasificacion: R | Director: Steven Spielberg |
Reparto: Tom Hanks, Tom Sizemore, Edward Burns, Barry Pepper, Adam Goldberg, Vin
Diesel, Giovanni Ribisi, Jeremy Davies, Matt Damon, Ted Danson, Paul Giamatti,
Dennis Farina, Joerg Stadler, Max Martini, Dylan Bruno, Bryan Cranston |
Sinopsis: Durante la invasión de Normandía, en plena Segunda Guerra Mundial, a
un grupo de soldados americanos se le encomienda una peligrosa misión: poner a
salvo al soldado James Ryan. Los hombres de la patrulla del capitán John Miller
deben arriesgar sus vidas para encontrar a este soldado, cuyos tres hermanos han
muerto en la guerra. Lo único que se sabe del soldado Ryan es que se lanzó con
su escuadrón de paracaidistas detrás de las líneas enemigas.</Descripcion>
            <URL>http://www.imdb.com/title/tt0120815/</URL>
            <AreaTematica>PeliculaBelica</AreaTematica>
      </Recurso>
</Recursos>
```

3.2.2.1.1 ValidadorXMLRecursos.xsd

Si bien no es una fuente de información, es necesario para validar una fuente de información muy importante como lo son los recursos del sistema. Este archivo valida que los archivo xml que contienen recursos estén bien estructurados. Esto es que contengan las etiquetas adecuadas y que los campos de dichas etiquetas contengan datos correctos (ej: no sean vacíos, contenga caracteres, etc.), aplicándose diversas restricciones sobre los mismos.

Con esto nos aseguramos que los recursos ingresados al sistema (para ser evaluados por parte de los usuarios del mismo) estén formados adecuadamente y sean validados para ser utilizados en el Sistema Recomendador.

El archivo empleado para tal fin es el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:element name="Recursos">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Recurso" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Titulo">
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:minLength value="1"/>
                           <xs:pattern value=".*[^\s].*"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:element>
                  <xs:element name="Descripcion">
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:minLength value="1"/>
                           <xs:pattern value=".*[^\s].*"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:element>
                  <xs:element name="URL">
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:minLength value="1"/>
                           <xs:pattern value=".*[^\s].*"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:element>
                  <xs:element name="AreaTematica">
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:minLength value="1"/>
                           <xs:pattern value=".*[^\s].*"/>
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

3.2.2.2 Atributos.xml

Este archivo contiene los atributos de calidad (que serán evaluados sobre los recursos según la red social en la que se esté) y que se cargarán en la tabla de atributos de la base de datos del Sistema Recomendador. Notar que no se le indica el dominio (o nombre de la red social), esto lo resuelve el Sistema Recomendador. Si el archivo contiene una configuración para otro dominio (o nombre) que no sea el

correspondiente a la red social en donde se está, no surtirá efecto alguno sobre la tabla asociada para los atributos en la base de datos (no habrá modificaciones).

Un posible ejemplo del archivo para un dominio por defecto sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<Atributos>
	<Atributo>
		<Nombre>Trustworthy</Nombre>
		<Peso>0.25</Peso>
	</Atributo>
	<Atributo>
		<Nombre>Objective</Nombre>
		<Peso>0.25</Peso>
	</Atributo>
	<Atributo>
		<Nombre>Well-written</Nombre>
		<Peso>0.25</Peso>
	</Atributo>
	<Atributo>
		<Nombre>Complete</Nombre>
		<Peso>0.25</Peso>
	</Atributo>
</Atributos>
```

3.2.2.2.1 ValidadorXMLAtributos.xsd

Si bien no es una fuente de información, es necesario para validar la fuente de información que contiene la configuración de los atributos sobre un dominio (red social) en particular. Este archivo valida que el archivo Atributos.xml esté bien estructurado. Esto es que contenga las etiquetas adecuadas y que los campos de dichas etiquetas contengan datos correctos (ej: no sean vacíos, contenga caracteres, etc.), aplicándose diversas restricciones sobre los mismos.

Con esto nos aseguramos que los recursos ingresados al sistema (para ser evaluados por parte de los usuarios del mismo) estén formados adecuadamente y sean validados para ser utilizados en el Sistema Recomendador.

El archivo empleado para tal fin es el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Atributos">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Atributo" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
```

```
                    <xs:element name="Nombre">
                       <xs:simpleType>
                          <xs:restriction base="xs:string">
                             <xs:minLength value="1"/>
                             <xs:pattern value=".*[^\s].*"/>
                          </xs:restriction>
                       </xs:simpleType>
                    </xs:element>
                    <xs:element name="Peso">
                       <xs:simpleType>
                          <xs:restriction base="xs:string">
                             <xs:minLength value="1"/>
                             <xs:pattern value="0\.[0-9]?[0-9]?[0-9]?[0-9]?[0-9]?|
0|1|1\.0[0]*"/>
                          </xs:restriction>
                       </xs:simpleType>
                    </xs:element>
                 </xs:sequence>
              </xs:complexType>
           </xs:element>
        </xs:sequence>
     </xs:complexType>
</xs:element>
</xs:schema>
```

3.2.2.3 CargaRecursos.sql

Este archivo contiene los recursos (a ser evaluados por los usuarios de la red social) que se cargarán en la tabla de recursos de la base de datos del Sistema Recomendador, operando directamente sobre la base de datos, lo cual es soportado en nuestro sistema, pero no es recomendado. Lo correcto es realizar la carga de recursos en el sistema, a través de un archivo xml.

Un posible ejemplo del archivo sería:

```
INSERT INTO recomendador.recurso (titulo, descripcion, url, fecha_rec,
areaTematica, evaluado, contexto)
VALUES ('Articulo1', 'Articulo1', 'www.example.com/Articulo1', NOW(),
'Pelicula', 0, 'www.facebook.com'),
('Articulo2', 'Articulo2', 'www.example.com/Articulo2', NOW(), 'Pelicula', 0,
'defecto'),
('Articulo3', 'Articulo3', 'www.example.com/Articulo3', NOW(), 'Musica', 0,
'www.google.com'),
('Articulo4', 'Articulo4', 'www.example.com/Articulo4', NOW(), 'Pelicula', 0,
'defecto'),
('Articulo5', 'Articulo5', 'www.example.com/Articulo5', NOW(), 'Viaje', 0,
'defecto'),
('Articulo6', 'Articulo6', 'www.example.com/Articulo6', NOW(), 'Pelicula', 0,
'defecto'),
...
...
```

```
...
('Articulo28', 'Articulo28', 'www.example.com/Articulo28', NOW(), 'Musica', 0,
'www.facebook.com'),
('Articulo29', 'Articulo29', 'www.example.com/Articulo29', NOW(), 'Musica', 0,
'defecto'),
('Articulo30', 'Articulo30', 'www.example.com/Articulo30', NOW(), 'Pelicula', 0,
'defecto');
```

3.2.2.4 CargaAtributos.sql

Este archivo contiene los atributos de calidad (que serán evaluados sobre los recursos según el dominio al que pertenezca la red social) que se cargarán en la tabla de atributos de la base de datos del Sistema Recomendador, operando directamente sobre la base de datos, lo cual es soportado en nuestro sistema, pero no es recomendado. Lo correcto es realizar la carga de atributos en el sistema, a través de un archivo xml.

Un posible ejemplo del archivo sería:

```
INSERT INTO recomendador.atributo (nomAtributo, contexto, peso)
VALUES ('Trustwhorthy', 'defecto', 0.25),
('Objective', 'defecto', 0.25),
('Well-written', 'defecto', 0.25),
('Complete', 'defecto', 0.25),
('Me Gusta', 'www.facebook.com', 1),
('+1', 'www.google.com', 1);
```

3.2.2.5 CargaEvaluaciones.sql

Este archivo contiene los recursos evaluados por usuarios que se cargarán en la tabla de recursos evaluados de la base de datos del Sistema Recomendador, operando directamente sobre la base de datos, lo cual es soportado en nuestro sistema, pero no es recomendado. Lo correcto es realizar la carga de recursos evaluados en el sistema, a través de un archivo xml. No es un archivo necesario pero se puede utilizar la carga del mismo para mitigar el problema del arranque en frío del Sistema Recomendador (es decir que no existen evaluaciones realizadas)

Un posible ejemplo del archivo sería:

```
INSERT INTO recomendador.evaluacionesrec (idUsuario, idRecurso, areaTematica,
contexto, nom_atributo1, puntaje1, nom_atributo2, puntaje2, nom_atributo3,
puntaje3, nom_atributo4, puntaje4, tipo_vinculo, fecha_eval)
VALUES ('diego.bastiani', 1, 'Pelicula', 'www.facebook.com', 'Me gusta', 10,
'Otro', 0, 'Otro', 0, 'Otro', 0, 'Amistad', NOW()),
('diego.bastiani', 2, 'Pelicula', 'defecto', 'Trustwhorthy', 8, 'Objective', 6,
```

```
'Well-written', 6, 'Complete', 7, 'Academico', NOW()),
('marcos.suiffet', 3, 'Musica', 'www.google.com', '+`1', 10, 'Otro', 0, 'Otro',
0, 'Otro', 0, 'Academico', NOW()),
('diego.bastiani', 4, 'Pelicula', 'defecto', 'Trustwhorthy', 3, 'Objective', 6,
'Well-written', 6, 'Complete', 6, 'Academico', NOW()),
('marcos.suiffet', 5, 'Viaje', 'defecto', 'Trustwhorthy', 6, 'Objective', 3,
'Well-written', 5, 'Complete', 4, 'Academico', NOW()),
('diego.bastiani', 5, 'Viaje', 'defecto', 'Trustwhorthy', 7, 'Objective', 6,
'Well-written', 8, 'Complete', 6, 'Academico', NOW()),
('marcos.suiffet', 6, 'Pelicula', 'defecto', 'Trustwhorthy', 2, 'Objective', 8,
'Well-written', 3, 'Complete', 6, 'Academico', NOW()),
('diego.bastiani', 7, 'Libro', 'defecto', 'Trustwhorthy', 3, 'Objective', 8,
'Well-written', 5, 'Complete', 8, 'Academico', NOW()),
('marcos.suiffet', 8, 'Musica', 'defecto', 'Trustwhorthy', 6, 'Objective', 4,
'Well-written', 7, 'Complete', 3, 'Academico', NOW()),
('diego.bastiani', 9, 'Libro', 'defecto', 'Trustwhorthy', 5, 'Objective', 7,
'Well-written', 8, 'Complete', 8, 'Academico', NOW()),
('marcos.suiffet', 10, 'Libro', 'defecto', 'Trustwhorthy', 7, 'Objective', 3,
'Well-written', 9, 'Complete', 5, 'Academico', NOW()),
('diego.bastiani', 8, 'Musica', 'defecto', 'Trustwhorthy', 8, 'Objective', 7,
'Well-written', 5, 'Complete', 6, 'Academico', NOW()),
('marcos.suiffet', 9, 'Libro', 'defecto', 'Trustwhorthy', 7, 'Objective', 3,
'Well-written', 4, 'Complete', 7, 'Academico', NOW());
```

Cuando se agregan recursos evaluados por parte de los usuarios a la base de datos, se debe setear en la tabla de recursos, en la columna 'evaluado' el valor '1' para los identificadores de recursos que intervienen en dicha adición.

Cuando la acción anterior se realiza a través del ingreso de recursos evaluados utilizando el sistema recomendador (se carga un archivo xml por parte del administrador), el sistema recomendador realiza las operaciones necesarias sobre la base de datos de forma de dejar los datos consistentes.

3.3 Componente lógica

A continuación se expone lo realizado en la componente lógica del Sistema Recomendador, mostrando las clases involucradas y las funcionalidades de éstas. Además se muestra la arquitectura de la componente lógica y cómo se realiza la interacción con la Persistencia y con la Interfaz Gráfica. Se muestra y explica claramente la secuencia de como se extraen las recomendaciones empleando el algoritmo de recomendación con Friendsourcing Semántico.

El Sistema Recomendador en su componente lógica está formado por una clase principal llamada Recomendador.java que es la encargada de gestionar los pedidos provenientes de la Interfaz Gráfica (recomendaciones, evaluar/puntuar recursos, etc.).

Dentro de la componente lógica se encuentran las clases:

- ManejoArchivos.java: se encarga de las operaciones de I/O al Sistema Recomendador (cargar la red social, agregar recursos, etc.)
- ParserXML.java: validada archivos xml y parsea los archivos indicados para devolver estructuras adecuadas a la clase ManejoArchivos.java (no es accedida directamente por la clase Recomendador.java)
- ConexiónBD.java: opera sobre la base de datos del Sistema Recomendador.
- RelacionSemantica.java: opera sobre el modelo de la Red Social y el modelo de Relaciones.
- AreaTematicaSemantica.java: opera sobre el modelo de las Áreas Temáticas.

Si bien la clase Recomendador.java tiene visibilidad directa sobre los modelos de Red Social, de Relaciones, y de Áreas Temáticas, se optó por realizar las clases RelacionSemantica.java y AreaTematicaSemantica.java para que las mismas se encarguen de operar sobre los modelos y hacer mas legible a la clase Recomendador.java.

Y además, esta distribución de responsabilidades de las clases descriptas anteriormente, se realizó no solo respetando los requerimientos funcionales del Sistema Recomendador, sino también contemplando los requerimientos no funcionales del mismo.

3.3.1 Arquitectura e Interacción de la componente lógica

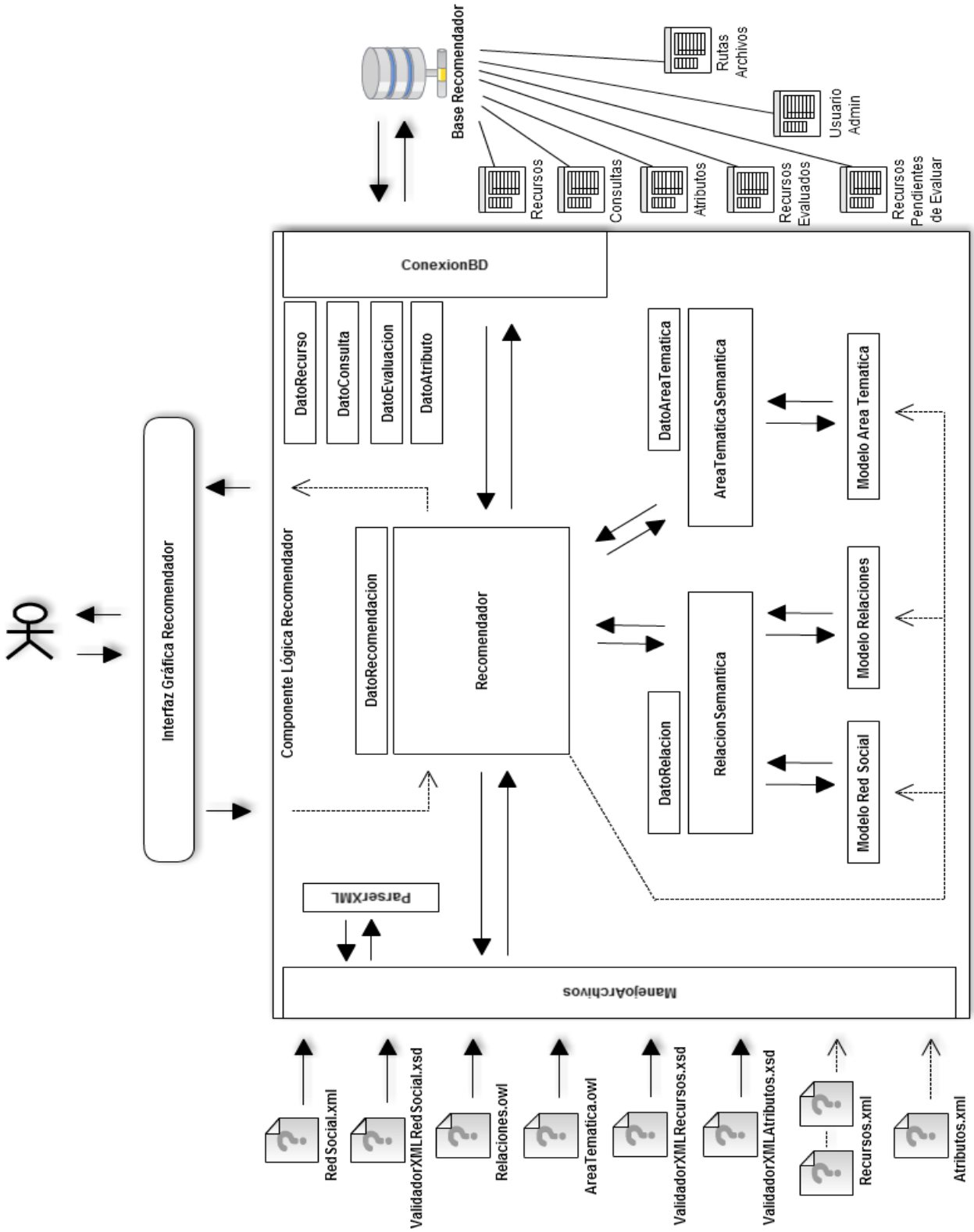


Figura 6 - Arquitectura e Interacción de la componente lógica

3.3.2 Secuencia de obtención de las recomendaciones

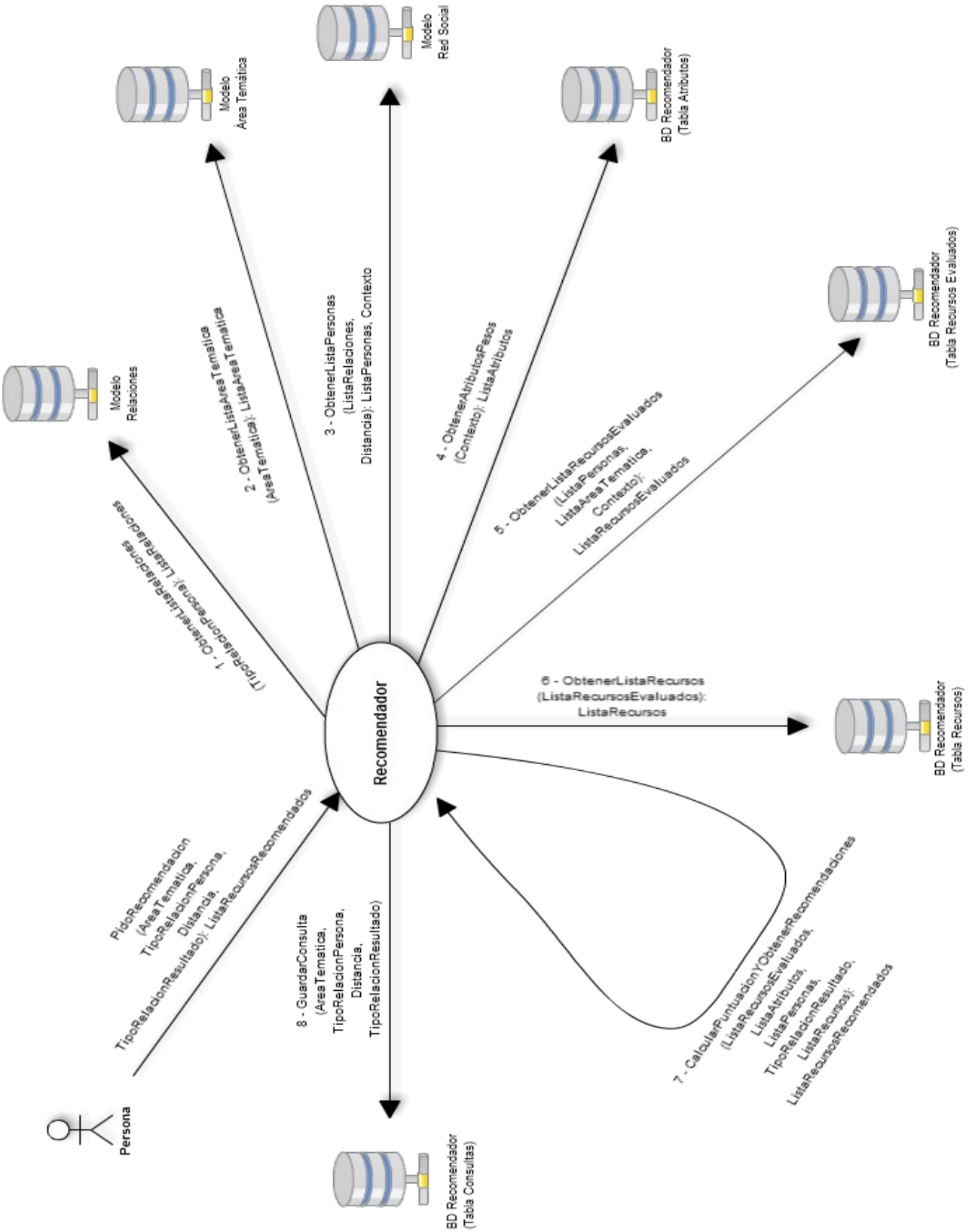


Figura 7 - Secuencia de obtención de las recomendaciones

3.3.3 Clases involucradas en la componente lógica

3.3.3.1 Recomendador.java

Esta clase es empleada para obtener las recomendaciones para un usuario de la red social, utilizando el algoritmo de Friendsourcing Semántico para tal fin. Estas recomendaciones son extraídas a partir de los recursos evaluados por parte de los usuarios de la red social. Utiliza a las clases ManejoArchivos.java, ConexionBD.java, RelacionSemantica.java y AreaTematicaSemantica.java, así como a los modelos que contienen a la red social, y a las ontologías de vocabulario para las relaciones y para las áreas temáticas.

Aparte de las recomendaciones, esta clase provee al usuario que solicita las mismas, recursos que aún no han sido evaluados por los usuarios de la red social y que pueden ser de interés para el mismo.

También provee operaciones para que puedan ser ejecutadas únicamente por el administrador del sistema.

La interfaz gráfica accede a la componente lógica del Sistema Recomendador a través de esta clase.

Variables que utiliza la clase Recomendador:

Esta clase Recomendador mantiene para su uso los siguientes modelos(Model de Jena):

- Model modeloRelaciones = **null**;
- Model modeloRedSocial = **null**;
- Model modeloAreaTematica = **null**;

Inicialización de la clase Recomendador:

Para inicializar el Recomendador, es necesario pasarle como parámetros las rutas donde se encuentran los archivos que contienen:

- La ontología de vocabulario de las relaciones
- La ontología de vocabulario de las áreas temáticas
- La red social
- El validador de la red social

Para este propósito, utiliza a la clase ManejoArchivos.java, quien devuelve los

modelos para las ontologías y la red social:

- **Recomendador**(String rutaRelaciones, String rutaRedSocial, String rutaValidadorRedSocial, String rutaAreaTematica)

```
ManejoArchivos manejadorArchivos = new ManejoArchivos();
modeloRelaciones = manejadorArchivos.getModeloRelaciones(rutaRelaciones);
modeloRedSocial = manejadorArchivos.getModeloRedSocial(rutaRedSocial, rutaValidadorRedSocial);
modeloAreaTematica = manejadorArchivos.getModeloAreaTematica(rutaAreaTematica);
```

Funciones incluidas en esta clase:

- **Necesarias para obtener las recomendaciones los usuarios (se detalla el código):**

- ArrayList<DatoRecomendacion> **getRecomendaciones**(String areaTematica, String idUsuario, String relacionPersona, int distancia, String relacionResultado)

```
AreaTematicaSemantica ats = new AreaTematicaSemantica();
RelacionSemantica rs = new RelacionSemantica();
ConexionBD connBD = new ConexionBD();
ArrayList<DatoRecomendacion> listaRecomendados = new ArrayList<DatoRecomendacion>();

// Obtengo la lista de Áreas Temáticas (contiene la misma así como también las subclases de areaTematica)
ArrayList<String> listaAreaTematica = ats.getListaAreasTematicas(modeloAreaTematica, areaTematica);

// Obtengo Lista de Relaciones del Usuario que pide la recomendación con los demás usuarios de la red social
ArrayList<DatoRelacion> listaRelacionesUsuario = rs.getListaFinalUsuariosRelacion(modeloRelaciones, modeloRedSocial, relacionPersona, idUsuario, distancia);

// Obtengo el dominio o contexto de la red social
String contexto = rs.getDominioRedSocial(modeloRedSocial);

// Obtengo Lista de Atributos correspondientes al dominio de la red social
ArrayList<DatoAtributo> listaAtr = connBD.getListaAtributosContexto(contexto);
```

```
ArrayList<String> listaAtributos = new ArrayList<String>();
for(DatoAtributo da:listaAtr) {
	String atributo = da.getNomAtributo();
	listaAtributos.add(atributo);
}

// Obtengo Lista de Recursos evaluados
ArrayList<DatoEvaluacion> listaEval = new ArrayList<DatoEvaluacion>();
for(DatoRelacion dr:listaRelacionesUsuario) {
	for(String at:listaAreaTematica) {
		ArrayList<DatoEvaluacion> listaEvalAux =
connBD.getListaRecursosEvaluados(dr.getIdUsuario(), at, contexto);
		listaEval.addAll(listaEvalAux);
	}
}

// Obtengo relaciones (y subrelaciones) que se esperan en la recomendación por parte del usuario
ArrayList<String> listaRelacionesEsperada = rs.getListaRelaciones(modeloRelaciones,
relacionResultado);

// Obtengo las Recomendaciones
for(DatoEvaluacion de:listaEval) {

	// Datos del recurso
	int idRec = de.getIdRecurso();
	DatoRecurso rec = connBD.getRecursoId(idRec);
	String tituloRec = rec.getTitulo();
	String descRec = rec.getDescripcion();
	String urlRec = rec.getURL();
	String fechaRec = rec.getFechaRec();

	// Datos del usuario
	String idUsu = de.getIdUsuario();
	String nomUsu = rs.getNombreUsuarioID(listaRelacionesUsuario, idUsu);

	// Calculo ponderadores semánticos de la puntuación
	int valor1 = rs.getValor1(listaRelacionesUsuario, idUsu);

	String tipoVinculoEnRecursoEvaluado = de.getTipoVinculo();
	int valor2 = rs.getPuntosValor2(listaRelacionesEsperada, tipoVinculoEnRecursoEvaluado);
	int valor3 = rs.getPuntosValor3(listaRelacionesEsperada, tipoVinculoEnRecursoEvaluado);

	// Obtengo la fecha en que se evaluó el recurso
	String fechaEval = de.getFechaEval();

	// Obtengo los valores de los atributos y sus respectivos pesos
	float pesoAtributo1 = 0;
```

```
        float pesoAtributo2 = 0;
        float pesoAtributo3 = 0;
        float pesoAtributo4 = 0;
        int puntajeAtributo1 = de.getPuntaje1();
        String nombreAtributo1 = de.getNombreAtributo1();
        if(listaAtributos.contains(nombreAtributo1)){
                pesoAtributo1 = connBD.getPesoAtributoContexto(nombreAtributo1, contexto);
        }

        int puntajeAtributo2 = de.getPuntaje2();
        String nombreAtributo2 = de.getNombreAtributo2();
        if(listaAtributos.contains(nombreAtributo2)){
                pesoAtributo2 = connBD.getPesoAtributoContexto(nombreAtributo2, contexto);
        }

        int puntajeAtributo3 = de.getPuntaje3();
        String nombreAtributo3 = de.getNombreAtributo3();
        if(listaAtributos.contains(nombreAtributo3)){
                pesoAtributo3 = connBD.getPesoAtributoContexto(nombreAtributo3, contexto);
        }

        int puntajeAtributo4 = de.getPuntaje4();
        String nombreAtributo4 = de.getNombreAtributo4();
        if(listaAtributos.contains(nombreAtributo4)){
                pesoAtributo4 = connBD.getPesoAtributoContexto(nombreAtributo4, contexto);
        }

        // Obtengo la puntuación del recurso a recomendar
        float puntuacion = (puntajeAtributo1*pesoAtributo1 + puntajeAtributo2*pesoAtributo2 +
puntajeAtributo3*pesoAtributo3 + puntajeAtributo4*pesoAtributo4 + valor1 + valor2 + valor3);

        // Creo la Recomendación y la agrego a la lista de Recomendaciones
        DatoRecomendacion dato = new DatoRecomendacion(tituloRec, descRec, urlRec,
fechaRec, nomUsu, fechaEval, puntuacion);
        listaRecomendados.add(dato);
}

//Ordeno las recomendaciones por su puntuación de mayor a menor
Collections.sort(listaRecomendados);

//Elimino las recomendaciones duplicadas de menor puntuación
listaRecomendados = eliminarRecomendacionesRepetidas(listaRecomendados);

//Agrego la consulta a las consultas del usuario
connBD.addConsulta(idUsuario, areaTematica, contexto, relacionPersona, distancia,
relacionResultado);
```

```
//Devuelvo el resultado
return listaRecomendados;
```

Función auxiliar:

```
ArrayList<DatoRecomendacion>
eliminarRecomendacionesRepetidas(ArrayList<DatoRecomendacion>
listaRecomendacionOrdenada){

        ArrayList<DatoRecomendacion> listaRecomendadosFinal = new
ArrayList<DatoRecomendacion>();
        Set<String> recomendacion = new HashSet<String>();

        for(DatoRecomendacion item : listaRecomendacionOrdenada) {
                if (recomendacion.add(item.getTitulo())) {
                                listaRecomendadosFinal.add(item);
                }
        }
        return listaRecomendadosFinal;
}
```

- ArrayList<DatoRecurso> **getRecursosSinEvaluar**(Model modeloAreaTematica, String areaTematica, String contexto)

```
ArrayList<DatoRecurso> listaRecursosNoEval = new ArrayList<DatoRecurso>();
ConexionBD connBD = new ConexionBD();
AreaTematicaSemantica ats = new AreaTematicaSemantica();
ArrayList<String> listaAreaTematica = ats.getListaAreasTematicas(modeloAreaTematica,
areaTematica);
for(String at:listaAreaTematica) {
        ArrayList<DatoRecurso> listaEvalAux = connBD.getListaRecursoSinEvaluar(at, contexto);
        listaRecursosNoEval.addAll(listaEvalAux);
        }
return listaRecursosNoEval;
```

- **Necesarias para evaluar un recurso los usuarios**(cabezales)**:**
- String **getIdUsuario**(String nomUsuario)
- String **getDominioRedSocial**()

- int **getIdRecurso**(String titulo, String descripcion, String urlRec, String areaTematica, String contexto)
- ArrayList<DatoAtributo> **getListaAtributosContexto**(String contexto)
- void **addRecursoEvaluado**(String idUsuario, int idRecurso, String areaTematica, String contexto, String nom_atributo1, int puntaje1, String nom_atributo2, int puntaje2, String nom_atributo3, int puntaje3, String nom_atributo4, int puntaje4, String tipo_vinculo)

- **Necesarias para ver los recursos que han evaluado los usuarios**(cabezales)**:**

- ArrayList<DatoEvaluacion> **getListaRecursosEvaluadosUsuario**(String idUsuario, String contexto)
- DatoRecurso **getRecursoId**(int idRec)

- **Necesarias para ver las consultas (pedido de recomendaciones) que hay hecho los usuarios**(cabezales)**:**

- ArrayList<DatoConsulta> **getListaConsultasUsuario**(String idUsuario, String contexto)

- **Necesarias para que el administrador pueda ingresar recursos desde un archivo al sistema** (cabezales)**:**

- int **addRecursosDesdeArchivo**(String rutaRecursos, String rutaValidadorRecursos, String contexto, Recomendador recomendador)
- void **addRecurso**(String titulo, String descripcion, String urlRec, String areaTematica, String contexto)

- **Necesarias para que el administrador pueda setear los pesos de los atributos de calidad** (cabezales)**:**

- boolean **setAtributosDesdeArchivo**(String rutaAtributos, String

rutaValidadorAtributos, String contexto, Recomendador recomendador)

- boolean **setAtributos**(String nom_atributo1, float peso1, String nom_atributo2, float peso2, String nom_atributo3, float peso3, String nom_atributo4, float peso4, String contexto)

- **Necesarias ver recursos en un browser** (cabezales)**:**
- String getUrlIdRecurso(**int** idRec, String contexto)
- **void** abrirUrlEnBrowser(String url)

3.3.3.2 ManejoArchivos.java

Esta clase es utilizada para cargar las fuentes de información de la componente lógica(la red social y las ontologías de vocabulario para las relaciones y para las áreas temáticas) del Sistema Recomendador a través de modelos de Jena.

Para el caso de la red social, se delega el trabajo de cargar la red social en un modelo a la clase ParserXML.java quién realiza la validación de la red social para crear el modelo posteriormente y devolverlo a la clase ManejoArchivos.java.

Para el caso de adicionar recursos al sistema, así como el caso de setear los atributos de calidad de los recursos del sistema, también se le delega parte del trabajo a la clase ParserXML.java.

Funciones incluidas en esta clase(cabezales):

- Model **getModeloRelaciones**(String rutaRelaciones)
- Model **getModeloRedSocial**(String rutaRedSocial, String rutaValidadorRedSocial)
- Model **getModeloAreaTematica**(String rutaAreaTematica)
- int **addRecursosDesdeArchivo**(String rutaRecursos, String rutaValidadorRecursos, String contexto, Recomendador recomendador, ManejoArchivos manejadorArchivos)
- void **addRecurso(**String titulo, String descripcion, String urlRec, String areaTematica, String contexto, Recomendador recomendador)

- int **getIdRecurso**(String titulo, String descripcion, String urlRec, String areaTematica, String contexto, Recomendador recomendador)
- boolean **setAtributosDesdeArchivo**(String rutaAtributos, String rutaValidadorAtributos, String contexto, Recomendador recomendador, ManejoArchivos manejadorArchivos)
- boolean **setAtributos**(String nom_atributo1, float peso1, String nom_atributo2, float peso2, String nom_atributo3, float peso3, String nom_atributo4, float peso4, String contexto, Recomendador recomendador)

3.3.3.3 ParserXML.java

La presente clase tiene como propósito:

- Validar que el archivo provisto para la red social(RedSocial.xml) esté correctamente estructurado. Para ello utiliza el archivo ValidadorXMLRedSocial.xsd para realizar la validación.
- Validar que el archivo provisto para agregar recursos al sistema (ejemplo: Recursos.xml) esté correctamente estructurado. Para ello utiliza el archivo ValidadorXMLRecursos.xsd para realizar la validación.
- Validar que el archivo provisto para la la configuración de los atributos de calidad de los recursos (Atributos.xml) esté correctamente estructurado. Para ello utiliza el archivo ValidadorXMLAtributos.xsd para realizar la validación.
- Crear el modelo(Model de Jena) para la Red Social (parsea el archivo RedSocial.xml) y se lo envía a la clase ManejoArchivos.java.
- Parsear los archivos de recursos y de atributos, extrayendo la información necesaria y enviándola a la clase ManejoArchivos.java.

¿ Cómo es que se realiza la creación del Modelo de la Red Social ?:

(Si el archivo RedSocial.xml es válido)

Utiliza los siguientes NS:

- rwsfUri = "http://rwsf/"

- relacionUri = "http://www.semanticweb.org/rwsf/ontologies/2015/4/relaciones#"

(donde rwsf es la abreviación de “recommender with sematic friendsourcing”)

Lo primero que se carga es el dominio, por ejemplo si es www.facebook.com, se le carga al modelo:

- model.setNsPrefix("rwsf", "http://rwsf/ww.facebook.com/)

Lo que sigue es crear los recursos del modelo y relacionarlos a través de propiedades para luego ser agregados al modelo. Sea el siguiente fragmento del archivo RedSocial.xml:

```
...
<Usuario>
          <Identificador>diego.bastiani</Identificador>
          <Nombre>Diego Bastiani</Nombre>
          <ListaRelacionamiento>
               <Vinculo>
                    <Identificador>marcos.suiffet</Identificador>
                    <Nombre>Marcos Suiffet</Nombre>
                    <TipoVinculo>amigo</TipoVinculo>
               </Vinculo>
...
```

Se crean los siguientes recursos (por ejemplo utilizando el dominio anterior):

- Resource r1 = model.createResource(http://rwsf/www.facebook.com/diego.bastiani)
- Resource r2 = model.createResource(http://rwsf/www.facebook.com/marcos.suiffet)

Y se agrega la siguiente propiedad:

- Property p = model.createProperty(relacionUri,"amigo")

Para finalmente agregarlos al modelo a los recursos relacionados mediante

propiedades (no solo a través de la propiedad “p”, sino a través de propiedades de RDF y FOAF):

- model.add(r1, RDF.type, FOAF.Person)
- model.add(r1, FOAF.accountName, "diego.bastiani")
- model.add(r1, FOAF.name, "Diego Bastiani")
- model.add(r2, RDF.type, FOAF.Person)
- model.add(r2, FOAF.accountName, "marcos.suiffet")
- model.add(r2, FOAF.name, "Marcos Suiffet")
- model.add(r1, p, r2)
- model.add(r1, FOAF.knows, r2);

De esta manera se va creando el modelo (grafo RDF) que será consultado cuando sea necesario por la clase RelacionSemantica.java.

Funciones incluidas en esta clase(cabezales):

- boolean **validateXMLSchema**(String xsdPath, String xmlPath)
- Model **crearGrafoRedSocialRDF**(String rutaArchivo)
- int **agregarRecursos**(String rutaArchivo, String dominio, Recomendador recomendador, ManejoArchivos manejadorArchivos)
- boolean **setAtributos**(String rutaArchivo, String dominio, Recomendador recomendador, ManejoArchivos manejadorArchivos)

3.3.3.4 RelacionSemantica.java

Esta clase es utilizada para extraer todas las relaciones(subclases) que son subrelaciones de una relación en particular (en términos de estructuras, sería todo el árbol que contiene a la relación en cuestión como raíz, para luego devolverlo en una lista ordenada según la aplicación del algoritmo DFS (búsqueda en profundidad)

sobre el árbol involucrado).

También es utilizada para realizar consultas sobre el grafo RDF que contiene a la estructura de la Red Social. Cuando se requieren extraer información sobre las relaciones, se devuelven a través de un tipo de datos denominado DatoRelacion.

Tanto el vocabulario de ontología para las relaciones como la estructura de la Red Social son cargados sobre una estructura denominada Modelo (de Jena) y la misma adopta la forma de un grafo RDF, por lo tanto se emplea SPARQL para realizar consultas sobre el mismo y obtener los resultados esperados.

Se emplea también como colaboradora en el armado de la puntuación de las recomendaciones (ponderador semántico), dado que el algoritmo de Friendsourcing Semántico se basa en el significado de las relaciones y esta clase tiene total acceso a las estructuras que las contienen, es por ello que el aporte semántico en la puntuación lo realiza dicha clase.

El formato SPARQL para realizar las consultas sobre el grafo RDF:

Se distinguen a continuación cuando son consultas realizadas sobre el modelo de Relaciones o sobre el modelo de la Red Social.

```
// A continuación se muestra una consulta sobre el grafo rdf de las relaciones
String consulta =
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX rel: <http://www.semanticweb.org/rwsf/ontologies/2015/4/relaciones#> " +
        // En la variable resultado se guarda lo extraído al ejecutar la consulta,
        // relación es un parámetro de entrada del cuál se quiere extraer las subclases del
        // mismo
        "SELECT ?resultado " +
                "WHERE { ?resultado (rdfs:subClassOf)* at:"+relacion+" }";


// A continuación se muestra una consulta sobre el grafo rdf de la red social
String consulta =
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
```

```
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX rel: <http://www.semanticweb.org/rwsf/ontologies/2015/4/relaciones#> " +
        // nsDomino es el espacio de nombres para el recomendador en un dominio determinado
        "PREFIX user: <"+nsDominio+"> " +
        "PREFIX foaf: <http://xmlns.com/foaf/0.1/> " +
        // En este ejemplo se extra el nombre de un usuario con identificador “id” en la red social
        // Se devuelve en la variable “resultado” el nombre del mismo.
        "SELECT * " +
                "WHERE { ?persona foaf:name ?resultado. " +
                "                ?persona foaf:accountName \""+id+"\" }";


/* Creo la consulta */
Query query = QueryFactory.create(consulta);


/* Ejecuto la consulta sobre el modeloRelaciones y obtengo los resultados */
QueryExecution qe = QueryExecutionFactory.create(query, modeloRelaciones);

ResultSet results = qe.execSelect();


/* O ejecuto la consulta sobre el modeloRedSocial si es el caso y obtengo los resultados */
QueryExecution qe = QueryExecutionFactory.create(query, modeloRedSocial);

ResultSet results = qe.execSelect();


/* Itero sobre el resultado(es una lista) y  realizo las operaciones pertinentes */
for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution();

        RDFNode ontUri = soln.get("resultado") ; //obtengo una URI


        // Al resultado debo asignarle un tipo de datos: String, int, etc.
        // si es un String procedo de la siguiente manera:
        String resul = ontUri.toString();

        // ej: “http://www.semanticweb.org/rwsf/ontologies/2015/4/relaciones#Amistad” si es sobre el
        // modelo de relaciones o "Diego Bastiani" si es sobre el modelo de la red social (foaf ya me da
        // el nombre)


        // En el caso de las relaciones, la URI la pasé a String, pero yo no quiero la URI, quiero el
        // nombre de la relación entonces aplico operaciones sobre cadena de caracteres
        result.substring(result.lastIndexOf("#") + 1));
        //ej obtenido: “Amistad”


        // Y el resultado lo voy agregando a una lista o alguna estructura adecuada como ser
```

```
        // DatoRelacion
        }

/*Libero la memoria*/
qe.close();

// Luego devuelvo la consulta en un lista o estructura adecuada
```

Funciones incluidas en esta clase(cabezales):

ModeloRelaciones:

- ArrayList<String> **getListaRelaciones**(Model modeloRelaciones, String vinculo)

ModeloRedSocial:

- String **getIdUsuario**(Model modeloRedSocial, String nombreUsuario)
- String **getDominioRedSocial**(Model modeloRedSocial)

Ambos modelos:

- ArrayList<DatoRelacion> **getListaUsuariosRealacion**(Model modeloRelaciones, Model modeloRedSocial, String vinculo, String idUsuario, int distancia)
- ArrayList<DatoRelacion> **getListaFinalUsuariosRealacion**(Model modeloRelaciones, Model modeloRedSocial, String vinculo, String idUsuario, int distancia)

Puntuación de las recomendaciones:

- int **getPuntosValor1**(ArrayList<DatoRelacion> listaRel, String idUsuario)
- int **getPuntosValor2**(ArrayList<String> listaRelaciones, String vinculo)
- int **getPuntosValor3**(ArrayList<String> listaRelaciones, String vinculo)

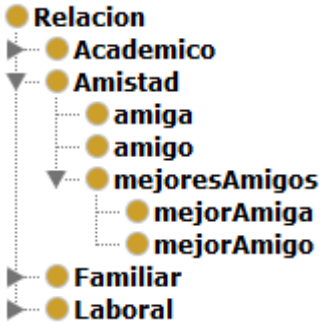
Otras:

- String **getNombreUsuarioID**(ArrayList<DatoRelacion> listaRel, String idUsuario)

Ejemplo sobre modelo de relaciones:

Problema:

“Obtener las subrelaciones de la relación Amistad”



Resultado:

{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}

Ejemplo sobre modelo de la Red Social:

Problema:

“Obtener el id del usuario con nombre='Diego Bastiani'”

```
<?xml version="1.0" encoding="UTF-8"?>
<RedSocial>
     <Dominio>www.example.com</Dominio>
     <Usuario>
          <Identificador>diego.bastiani</Identificador>
          <Nombre>Diego Bastiani</Nombre>
          <ListaRelacionamiento>
               <Vinculo>
                    <Identificador>marcos.suiffet</Identificador>
                    <Nombre>Marcos Suiffet</Nombre>
                    <TipoVinculo>amigo</TipoVinculo>
               </Vinculo>
```

```
                <Vinculo>
                    <Identificador>marcos.suiffet</Identificador>
                    <Nombre>Marcos Suiffet</Nombre>
                    <TipoVinculo>estudiante-curso</TipoVinculo>
                </Vinculo>
                ...
            </ListaRelacionamiento>
        </Usuario>
        ...
</RedSocial>
```

Resultado:

{diego.bastiani}

Ejemplo sobre colaboración en las puntuaciones de las recomendaciones:

Supongamos:

DatoRelacion1={Marcos Suiffet, marcos.suiffet, amigo, 2}

DatoRelacion2={Diego Bastiani, diego.bastiani, mejorAmigo, 1}

(DatoRelacion:{nombre usuario, id usuario, relación, distancia})

Y supongamos que tenemos las subrelaciones de Amistad (ejemplo sobre el modelo de Relaciones anterior):

{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}

Ejemplos sobre los DatoRelacion:

getPuntosValor1({DatoRelacion1, DatoRelacion2}, diego.bastiani) = 0

getPuntosValor1({DatoRelacion1, DatoRelacion2}, marcos.suiffet) = 2

Ejemplos sobre la lista de relaciones y subrelaciones:

getPuntosValor2(*{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}*, Amistad) = 1

getPuntosValor2(*{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga,*

mejorAmigo}, Empleado) = 0

getPuntosValor2(*{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}*, amigo) = 1

getPuntosValor3(*{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}*, Amistad) = 1

getPuntosValor3(*{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}*, Empleado) = 0

getPuntosValor3(*{Amistad, amiga, amigo, mejoresAmigos, mejorAmiga, mejorAmigo}*, amigo) = 0

3.3.3.5 AreaTematicaSemantica.java

Esta clase se emplea para extraer todas las áreas temáticas(subclases) que son subáreas temáticas de un área temática en particular (en términos de estructuras, sería todo el árbol que contiene a el área temática en cuestión como raíz, para luego devolverlo en una lista ordenada según la aplicación del algoritmo DFS (búsqueda en profundidad) sobre el árbol involucrado).

El vocabulario de ontología para las áreas temáticas es cargado sobre una estructura denominada Modelo (de Jena) y la misma adopta la forma de un grafo RDF, por lo tanto se emplea SPARQL para realizar consultas sobre el mismo y obtener los resultados esperados.

<u>El formato SPARQL para realizar las consultas sobre el grafo RDF:</u>

```
// Formulo la consulta
String consulta =
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
        "PREFIX at: <http://www.semanticweb.org/rwsf/ontologies/2015/4/area_tematica#> " +
        // En la variable resultado se guarda lo extraído al ejecutar la consulta,
        // areaTematica es un parámetro de entrada del cuál se quiere extraer las subclases del
        // mismo
        "SELECT ?resultado " +
                "WHERE { ?resultado (rdfs:subClassOf)* at:"+areaTematica+" }";
```

```
/* Creo la consulta */
Query query = QueryFactory.create(consulta);


/* Ejecuto la consulta sobre el modeloAreaTematica y obtengo los resultados */
QueryExecution qe = QueryExecutionFactory.create(query, modeloAreaTematica);
ResultSet results = qe.execSelect();


/* Itero sobre el resultado(es una lista) y  realizo las operaciones pertinentes */
for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution();
        RDFNode ontUri = soln.get("resultado") ; //obtengo una URI


        // Al resultado debo asignarle un tipo de datos: String, int, etc.
        // si es un String procedo de la siguiente manera:
        String subcat = ontUri.toString();
        // ej: “http://www.semanticweb.org/rwsf/ontologies/2015/4/area_tematica#Musica”


        // La URI la pasé a String, pero yo no quiero la URI, quiero el nombre del área temática
        // entonces aplico operaciones sobre cadena de caracteres
        subcat.substring(subcat.lastIndexOf("#") + 1));
        //ej obtenido: “Musica”


        // Y el resultado lo voy agregando a una lista o alguna estructura adecuada
        }


/*Libero la memoria*/
qe.close();
// Luego devuelvo la consulta en un lista o estructura adecuada
```

Funciones incluidas en esta clase(cabezales):

- ArrayList<String> **getListaAreasTematicas**(Model modeloAreaTematica, String areaTematica)

Ejemplo:

Problema:

“Obtener las subáreas temáticas del área temática Educación”

Resultado:

{Educacion, Escuela, EscuelaPrivada, EscuelaPublica, Liceo, LiceoPrivado, LiceoPublico, Universidad, UniversidadPrivada, UniversidadPublica}

3.3.3.6 ConexionBD.java

Esta clase es utilizada para realizar todas las operaciones requeridas sobre la base de datos del Sistema Recomendador (sobre una base MySQL). Se muestra primeramente como se establece la conexión a la base de datos y luego se muestran las operaciones realizadas sobre la misma; éstas operaciones se incluyen dentro de las funciones que se muestran mas adelante, diferenciadas según sea la tabla en que se opere.

Para realizar la conexión se utilizaron los siguientes parámetros:

```
private String url = "jdbc:mysql://localhost:3306/recomendador";
private String username = "root";
private String password = "Acá va el pwd de la base";
```

El formato que se estableció para operar sobre la base de datos es el siguiente:

```
try (Connection connection = (Connection) DriverManager.getConnection(url, username, password)) {
    java.sql.Statement s = connection.createStatement();
    String sql = "Acá va la operación sobre la base de datos";

    /* Si la operación es consulta: */
        // Se realiza la consulta. Los resultados se guardan en el ResultSet rs
        java.sql.ResultSet rs = ((java.sql.Statement) s).executeQuery (sql);
```

```
            // Se recorre el ResultSet, y se operan con los datos extraídos de la consulta
            while (rs.next()) {

            ...
            }

    /* Si la operación es de inserción: */
            s.executeUpdate(sql);

    /* Finalmente cierro la conexión con la base de datos */
    connection.close();
    s.close();

    /* Opcional: Si retorno algo lo realizo aquí */

    } catch (SQLException e) {
            throw new IllegalStateException("Acá va el mensaje de error", e);
            }
```

Se realizan dos tipos de operaciones sobre la base de datos del recomendador:

- SELECT: Para realizar consultas sobre las tablas.
- INSERT: Para realizar modificaciones o agregar datos en las tablas.

Las funciones realizadas sobre las tablas de la base son las siguientes (cabezales):

- *Recursos:*
  - ArrayList<DatoRecurso> **getListaRecursos()**
  - DatoRecurso **getRecursoId**(int idRecursoIN)
  - int **getIdRecurso**(String titulo, String descripcion, String urlRec, String areaTematica, String contexto)
  - ArrayList<DatoRecurso> **getListaRecursoSinEvaluar**(String areaTematicaIN, String contextoIN)
  - void **addRecurso**(String titulo, String descripcion, String urlRec, String areaTematica, String contexto)

- *Recursos Evaluados:*
  - ArrayList<DatoEvaluacion> **getListaTodosRecursosEvaluados**()
  - ArrayList<DatoEvaluacion> **getListaRecursosEvaluados**(String idUsuarioIN, String areaTematicaIN, String contextoIN)
  - ArrayList<DatoEvaluacion> **getListaRecursosEvaluadosUsuario**(String idUsuarioIN, String contextoIN)
  - void **addRecursoEvaluado**(String idUsuario, int idRecurso, String areaTematica, String contexto, String nom_atributo1, int puntaje1, String nom_atributo2, int puntaje2, String nom_atributo3, int puntaje3, String nom_atributo4, int puntaje4, String tipo_vinculo)

- *Consultas de Usuarios:*
  - ArrayList<DatoConsulta> **getListaConsultasUsuario**(String idUsuarioIN, String contextoIN)
  - void **addConsulta**(String idUsuario, String areaTematica, String contexto, String relacionPersona, int distancia, String relacionResultado)

- Atributos:
  - String **consultarAtributos**()
  - ArrayList<DatoAtributo> **getListaAtributosContexto**(String contexto)
  - float **getPesoAtributoContexto**(String nombreAtributo, String contexto)
  - boolean **setAtributos**(String nom_atributo1, float peso1, String nom_atributo2, float peso2, String nom_atributo3, float peso3, String nom_atributo4, float peso4)

3.3.3.7 Tipos de Datos

Se crearon los siguientes tipos de datos para representar:

- Recomendaciones
- Evaluaciones de los recursos
- Consultas de usuarios

- Atributos
- Recursos
- Relaciones
- Área Temática

Todos los tipos de datos incluyen: los constructores(vacío y con datos), consultas (get) y modificadores (set), así como la impresión en pantalla(toString), y que no serán detallados.

3.3.3.7.1 DatoRecomendacion.java

Dado que las recomendaciones se muestran ordenadas de mayor a menor puntuación, es necesario poder ordenar estos datos, por ello este tipo de datos debe implementar las funciones de la clase “Comparable” necesarias para tal fin.

```
public class DatoRecomendacion implements Comparable<Object>
```

Los atributos de este tipo de datos son los siguientes:

```
private String titulo;
private String descripcion;
private String url;
private String fechaRecurso;
private String nombreUsuario;
private String fechaEvaluadoUsuario;
private float puntuacion;
```

Además es necesario sobrecargar las funciones para comparar las puntuaciones de las recomendaciones entre los tipos de datos de DatoRecomendacion.

```
@Override
public boolean equals(Object object) {
   boolean isEqual= false;
   if (object != null && object instanceof DatoRecomendacion)
   {
      isEqual = (this.puntuacion == ((DatoRecomendacion) object).puntuacion);
   }
   return isEqual;
}

@Override
public int hashCode() {
```

```
   return ((this.titulo.length())*(this.nombreUsuario.length()));
}

@Override
public int compareTo(Object compareRecomendacion) {
   float comparePuntuacion=((DatoRecomendacion)
compareRecomendacion).getPuntuacion();
  return Float.compare(comparePuntuacion,this.puntuacion);
}
```

3.3.3.7.2 DatoEvaluación.java

Los atributos de este tipo de datos son los siguientes:

```
private String idUsuario;
private int idRecurso;
private String areaTematica;
private String contexto;
private String nom_atributo1;
private int puntaje1;
private String nom_atributo2;
private int puntaje2;
private String nom_atributo3;
private int puntaje3;
private String nom_atributo4;
private int puntaje4;
private String tipo_vinculo;
private String fecha_eval;
```

3.3.3.7.3 DatoConsulta.java

Los atributos de este tipo de datos son los siguientes:

```
private String idUsuario;
private String areaTematica;
private String contexto;
private String relacionPersona;
private int distancia;
private String relacionResultado;
private String fecha_consulta;
```

3.3.3.7.4 DatoAtributo.java

Los atributos de este tipo de datos son los siguientes:

```
private String nomAtributo;
private String contexto;
private float peso;
```

3.3.3.7.5 DatoRecurso.java

Los atributos de este tipo de datos son los siguientes:

```
private int idRecurso;
private String titulo;
private String descripcion;
private String url;
private String fecha_rec;
private String areaTematica;
private int evaluado; // 0 no, 1 si
private String contexto;
```

3.3.3.7.6 DatoRelacion.java

Los atributos de este tipo de datos son los siguientes:

```
private String nombreUsuario;
private String idUsuario;
private String relacion;
private int distancia;
```

3.3.3.7.7 DatoAreaTematica.java

El tipo de datos usado en este caso es String. No se creó un tipo de datos en particular para una área temática.

3.4 Persistencia

A continuación se expone lo realizado para la Persistencia (en una base de datos). Esto incluye la creación de la base de datos para el Sistema Recomendador así como la creación de las tablas necesarias para posteriormente ser cargadas con datos, necesarios para el funcionamiento del Sistema Recomendador.

3.4.1 BaseRecomendador.sql

Este archivo es el encargado de crear la base de datos del Sistema Recomendador.
Su contenido es:

```
create database recomendador;
```

3.4.2 TablaRecurso.sql

Este archivo es el encargado de crear la tabla para los recursos en la base de datos del Sistema Recomendador.

Su contenido es:

```
create table recomendador.recurso(
idRecurso INT NOT NULL AUTO_INCREMENT,
titulo VARCHAR(100) NOT NULL,
descripcion VARCHAR(2500) NOT NULL,
url VARCHAR(450) NOT NULL,
fecha_rec DATETIME,
areaTematica VARCHAR(50) NOT NULL,
evaluado INT NOT NULL DEFAULT 0,
contexto VARCHAR(20) NOT NULL,
PRIMARY KEY ( idRecurso )
);
```

3.4.3 TablaAtributo.sql

Este archivo es el encargado de crear la tabla para los atributos de calidad de los recursos (según el dominio o red social en el que se esté) en la base de datos del Sistema Recomendador.

Su contenido es:

```
create table recomendador.atributo(
nomAtributo VARCHAR(30) NOT NULL,
contexto VARCHAR(20) NOT NULL,
peso FLOAT DEFAULT 0,
PRIMARY KEY ( nomAtributo,contexto )
);
```

3.4.4 TablaEvaluacionRecurso.sql

Este archivo es el encargado de crear la tabla para los recursos evaluados por los usuarios en la base de datos del Sistema Recomendador.

Su contenido es:

```
create table recomendador.evaluacionesrec(
idUsuario VARCHAR(50) NOT NULL,
idRecurso INT NOT NULL,
areaTematica VARCHAR(50) NOT NULL,
contexto VARCHAR(20) NOT NULL,
nom_atributo1 VARCHAR(30),
puntaje1 INT DEFAULT 0,
nom_atributo2 VARCHAR(30),
puntaje2 INT DEFAULT 0,
nom_atributo3 VARCHAR(30),
puntaje3 INT DEFAULT 0,
nom_atributo4 VARCHAR(30),
puntaje4 INT DEFAULT 0,
tipo_vinculo VARCHAR(30) NOT NULL,
fecha_eval DATETIME,
FOREIGN KEY (idRecurso) REFERENCES recomendador.recurso(idRecurso)
);
```

3.4.5 TablaConsultaUsuario.sql

Este archivo es el encargado de crear la tabla para los recursos en la base de datos del Sistema Recomendador.

Su contenido es:

```
create table recomendador.consulta_usuario(
idUsuario VARCHAR(50) NOT NULL,
areaTematica VARCHAR(50) NOT NULL,
contexto VARCHAR(20) NOT NULL,
relacionPersona VARCHAR(30) NOT NULL,
distancia INT NOT NULL,
relacionResultado VARCHAR(30) NOT NULL,
fecha_consulta DATETIME
);
```

3.4.6 Tabla RecursosPendEvaluar.sql

Este archivo es el encargado de crear la tabla para los recursos pendientes de evaluar de un usuario en la base de datos del Sistema Recomendador.

Su contenido es:

```
create table recomendador.recursos_pendevaluar(
```

```
idUsuario VARCHAR(50) NOT NULL,
idRecurso INT(11) NOT NULL,
titulo VARCHAR(100) NOT NULL,
descripcion VARCHAR(2500) NOT NULL,
url VARCHAR(450) NOT NULL,
fecha_rec DATETIME,
areaTematica VARCHAR(50) NOT NULL,
evaluado INT NOT NULL DEFAULT 0,
contexto VARCHAR(20) NOT NULL,
relacionResultado VARCHAR(30) NOT NULL,
PRIMARY KEY ( idUsuario,idRecurso )
);
```

3.4.7 Tabla RutaArchivos.sql

Este archivo es el encargado de crear la tabla para las rutas de los archivos que contienen fuentes de información para el Sistema Recomendador.

Su contenido es:

```
create table recomendador.ruta_archivos(
archivo VARCHAR(30) NOT NULL,
ruta VARCHAR(100) NOT NULL,
PRIMARY KEY (archivo)
);
```

3.4.8 Tabla Usuarios.sql (Usuario Administrador)

Este archivo es el encargado de crear la tabla para el usuario administrador, el cual contiene la password para acceder al sistema.

Su contenido es:

```
create table recomendador.usuarios(
idUsuario VARCHAR(10) NOT NULL,
pwd VARCHAR(30) NOT NULL,
PRIMARY KEY ( idUsuario )
);
```

3.5 Interfaz gráfica

A continuación se expone lo realizado en la interfaz gráfica del Sistema Recomendador, mostrando las clases involucradas y las funcionalidades de éstas.

Como motor, se utilizo el servidor de aplicaciones JBoss Application Server 7, en el cual es deployado el proyecto del Sistema Recomendador en su respectivo archivo war.

Para el desarrollo de la interfaz web, se utilizó el framework de Java JSF, en el cual para esto se utilizó el cliente Primefaces. Este es una suite de componentes JSF de código abierto con varias extensiones.

Además se uso CSS3 para personalizar el estilo de las páginas web.

3.5.1 Pautas para el desarrollo de la interfaz gráfica

El desarrollo de la solución comienza con un link de interés para el acceso a la solución del recomendador (index.xhtml). Esto es para poder acoplarlo a cualquier solución ya implementada. El acceso se hace a través de un link primefaces accediendo a la dirección de la página web maestra (pages/recomendador_actual.xhtml). En nuestro caso, para poder simular la aplicación nativa, escribimos en un textbox el nombre de usuario (que sera pasado por la aplicación nativa, mediante un bean llamado UsuarioLogueado, el cual solamente tiene el nombre del mismo) y luego accedemos a la aplicación mediante un botón Acceder Recomendador.

Figura 8 - Página de acceso al Sistema Recomendador

Una vez accedido a este, se podrá ver un set de acciones. Las mismas son: Búsqueda, Puntar Recurso, Historial Consultas Realizadas, Historial Recursos Evaluados y Salir. Las mismas podrán ser accedidas a través de un link a la izquierda de la página principal, y mediante botones, en el centro de la página, como se muestra en la siguiente imagen:

Figura 9 - Página de inicio del Sistema Recomendador

La navegación entre las páginas es la siguiente:

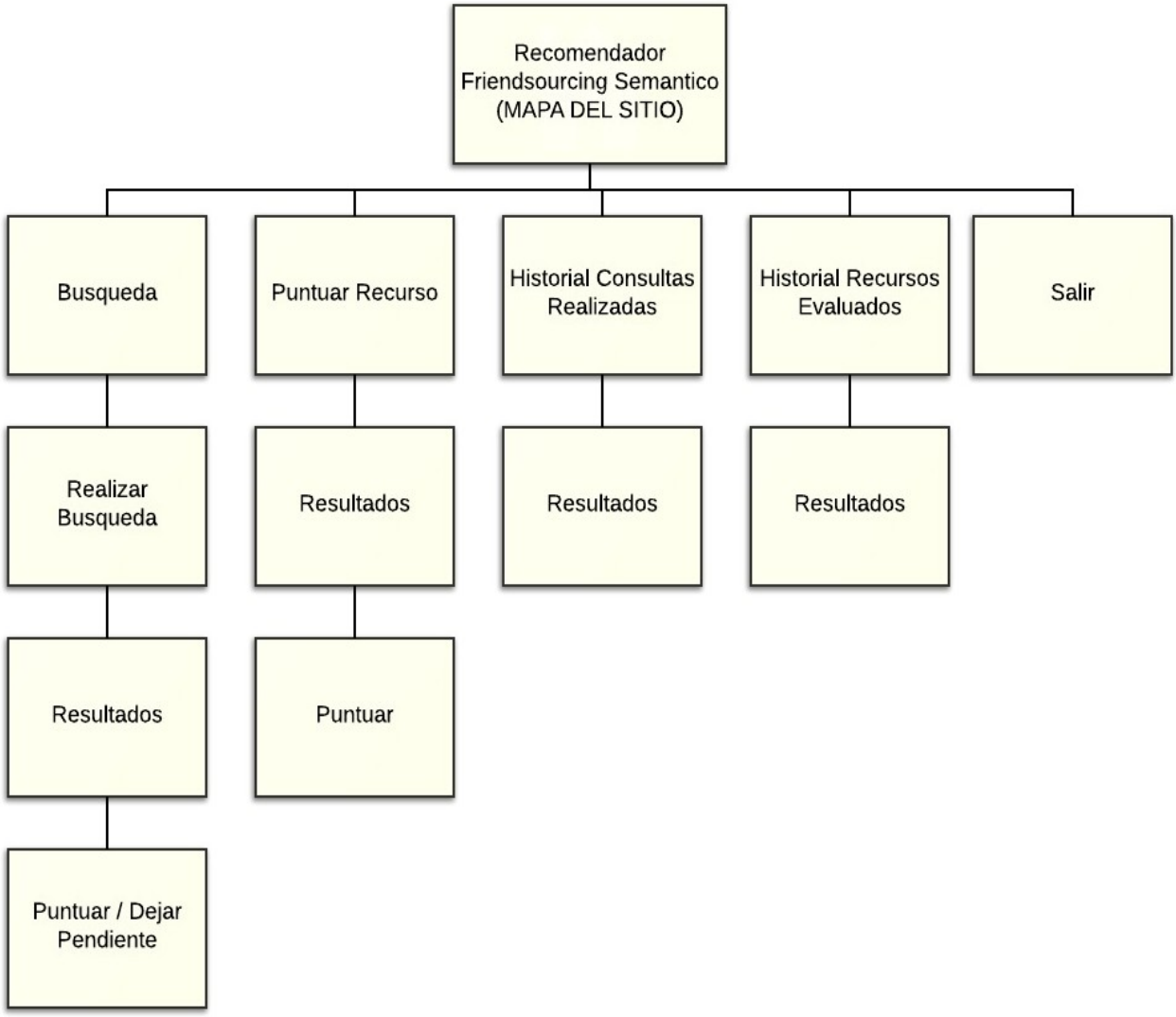


Figura 10 - Mapa de Navegación entre Páginas del Sitio

3.5.2 Diseño y desarrollo de la interfaz gráfica

Tal como se comentó anteriormente, el Sistema Recomendador tiene 5 acciones definidas.

Los propósitos de los mismos son:

- Búsqueda: Permite realizar la búsqueda requerida por parte del usuario, esto es, solicitar las recomendaciones al Sistema Recomendador. Una vez hecha, le da la opción al usuario de seleccionar el ítem de su interés y poder puntuarlo en el momento o dejarlo para puntuar posteriormente, como también, la posibilidad de visualizar el recurso dada la url del mismo y además, visualizar su

descripción. Los ítems que no se puntúan y se dejan para una posterior puntuación, se podrán ver en el punto Puntuar Recurso. Cabe mencionar, que los ítems seleccionados y que tengan su respectiva acción, dejaran de ser vistos en la página de resultados para el usuario.

- Puntuar Recurso: Obtiene los recursos pendientes de puntuar por el usuario. Le permite al usuario, puntuar el mismo, visualizar el recurso dada su url, como también poder visualizar su descripción.
- Historial de Consultas Realizadas: Permite visualizar las últimas búsquedas realizadas
- Historial de Recursos Evaluados: Permite ver los recursos evaluados.
- Salir: Sale del Sistema Recomendador.

El diseño y navegación del sistema, no se hace a través de múltiples páginas web, sino que se desarrolló sobre una única página “recomendador_actual.xhtml”. Los distintos componentes de búsqueda y resultados, se implementaron mediante el componente de Primefaces p:dialog. Este me permite superponer otros elementos en la página principal permitiéndonos dejarla siempre visible.

A continuación un breve resumen de la estructura de la página:

`recomendador_actual.xhtml:`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui">

<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>Recomendador</title>
        <style type="text/css">
                .notShow {
                        display:none;
                }

                body {
                        font: 100%/1.4 Verdana, Arial, Helvetica, sans-serif;
                        background: #42413C;
                        margin: 0;
                        padding: 0;
                        color: #000;
```

```
                }
...
...
...
...
                .button1 {
                        height:200px; width: 200px;  background-image: url("../images/buscar.jpeg");
        border-style:solid; border-width: 1px;
                        display:block!important;
                        -webkit-transition:-webkit-transform 0.5s ease-out;
                        -moz-transition:-moz-transform 0.5s ease-out;
                        -o-transition:-o-transform 0.5s ease-out;
                        -ms-transition:-ms-transform 0.5s ease-out;
                        transition:transform 0.5s ease-out;
                }
                .button1:HOVER {
                        -moz-transform: scale(1.05);
                        -webkit-transform: scale(1.05);
                        -o-transform: scale(1.05);
                        -ms-transform: scale(1.05);
                        transform: scale(1.05)
                }
                .zoomIt{
                        display:block!important;
                        -webkit-transition:-webkit-transform 0.5s ease-out;
                        -moz-transition:-moz-transform 0.5s ease-out;
                        -o-transition:-o-transform 0.5s ease-out;
                        -ms-transition:-ms-transform 0.5s ease-out;
                        transition:transform 0.5s ease-out;
                }
                .zoomIt:hover{
                        -moz-transform: scale(1.05);
                        -webkit-transform: scale(1.05);
                        -o-transform: scale(1.05);
                        -ms-transform: scale(1.05);
                        transform: scale(1.05)
                }
        </style>
</h:head>

<body>
<div class="container">
  <div class="header"><img src="../images/imagen_top.jpg" width="961" height="145"/><!--
end .header --></div>
  <div class="sidebar1">
    <h:form id="idForm1">
    <ul class="nav">
```

```
    <li>
          <p:commandLink value="Busqueda" update....."/>
    </li>
    <li>
          <p:commandLink value="Puntuar Recurso" action...../>
    </li>
    <li>
          <p:commandLink value="Historial Consultas Realizadas" action...../>
    </li>
    <li>
          <p:commandLink value="Historial Recursos Evaluados" action...../>
    </li>
         <li>
               <p:commandLink value="SALIR" action...../>
         </li>
  </ul>
  </h:form>
</div>
<div class="content">
  <table width="750" border="0">
    <tr>
      <td align="center">
                    <h:outputLink >
                <p:commandButton   .... title="Busqueda"/>
                    </h:outputLink>
      </td>
      <td align="center">
                    <p:commandLink >
                <p:commandButton .... title="Puntuar Recurso"/>
             </p:commandLink>
      </td>
      <td align="center">
                    <p:commandLink >
                <p:commandButton ... title="Historial Consultas Realizadas"/>
             </p:commandLink>
      </td>
    </tr>
    <tr>
      <td align="center">
                    <p:commandLink ">
                <p:commandButton ... title="Historial Recursos Evaluados"/>
        </p:commandLink>
      </td>
      <td align="center">
                    <p:commandLink >
                <p:commandButton .... title="Salir"/>
        </p:commandLink>
```

```
      </td>
      <td> </td>
     </tr>
    </table>
  </div>
  <div class="footer">
    <p></p>
  </div>
</div>

<h:form id="dlgForm">

        <p:dialog header="Busqueda de Recursos" >
        ...
        ...
        ...
        </p:dialog>

        <p:dialog header="Resultados de Busqueda">
                <p:dataTable id="recomendaciones" selectionMode="single" scrollable="true"
scrollHeight="150" resizableColumns="true">
                        ...
                </p:dataTable>

                <p:dataTable id="recomendacionesSEvaluar" selectionMode="single"
scrollable="true" scrollHeight="300" resizableColumns="true">
                        ...
                </p:dataTable>
        </p:dialog>

        <p:dialog  header="Recomendacion Info" >
        ...
        </p:dialog>

        <p:dialog header="Recomendacion Sin Evaluar Info">
        ...
        </p:dialog>

        <p:dialog header="Puntuar Recurso" >
         ...
        </p:dialog>

        <p:dialog header="Recurso Info" >
        ...
        </p:dialog>

        <p:dialog header="Historia Consultas Realizadas" >
```

```
            <p:dataTable id="historiales" selectionMode="single" scrollable="true"
scrollHeight="600" resizableColumns="true">
                ...
            </p:dataTable>
        </p:dialog>

        <p:dialog header="Historia Recursos Evaluados" >
            <p:dataTable id="historialesRE" selectionMode="single" scrollable="true"
scrollHeight="450" resizableColumns="true">
                ...
            </p:dataTable>
    </p:dialog>
</h:form>

<script type="text/javascript">

        function handleRecomendacionDialogOut() {
            PF('recomendacionDialog').hide();
            PF('dlgRB').show();
        }
...
...
        function handleBusquedaOut() {
            PF('dlg').hide();
        }
</script>

</body>
</html>
```

3.5.2.1 Búsqueda

El usuario podrá realizar la búsqueda según sus preferencias. En el mismo podrá seleccionar:

- Relación sobre Búsqueda del Usuario (relación en la red social).
- Distancia Máxima (1 o 2).
- Área Temática.
- Relación Esperada de la Recomendación (relación en los recursos evaluados).
- Recordar Selección Anterior.

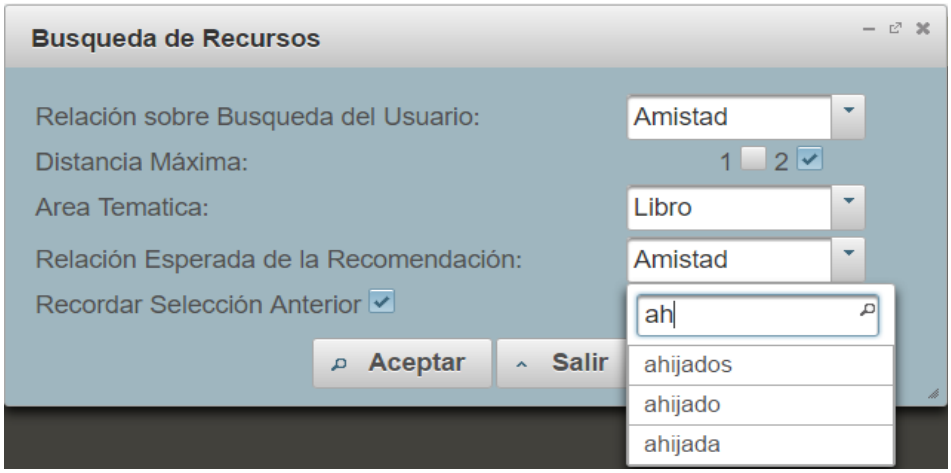


Figura 11 - Formulario para la Solicitud de Recomendaciones

Se deja como ayuda, para los input de Relaciones y de Área Temática, el menú con el universo de relaciones/áreas, permitiendo seleccionar una directamente, o realizar una búsqueda coincidiendo siempre el primer patrón del ítem a buscar. Cabe mencionar que para la totalidad de relaciones, como de áreas temáticas, se dejó operativa esta opción, pero el sistema una vez que tiene este valor, lo cambia manualmente al valor que corresponda según el campo, Relación para el caso de Relaciones, y AreaTematica para el caso de Áreas Temáticas. Estos 2 valores, son las superclases y por ende los totalizadores de ambos conjuntos.

Se utilizó el patrón MVC para separar la interfaz de usuario de la lógica de negocio.

Se usan beans para el manejo, pasaje de información y comunicación entre cliente-servidor. Los scope usados para los distintos beans son ApplicationScope y SessionScope. Este último es para la interfaz gráfica, mientras que los ApplicationScope, son usados por el bean Recomendador, quien es que tiene la totalidad de la lógica y algunos Services de la interfaz gráfica, ya que se mantienen a lo largo de la aplicación y no tienen sentido volver a recargarlos.

Esta página es manejada por un bean llamado RecomendacionService. La firma del mismo es la siguiente:

```
@ManagedBean(name = "recomendacionService")
@SessionScoped
public class RecomendacionServices implements Serializable

    @ManagedProperty(value="#{recomendadorBean}")
    private Recomendador rService;
    @ManagedProperty(value="#{abSelectOneMenuView}")
    private AyudaBusquedaSelectOneMenuView abSOMenuView;
    @ManagedProperty(value="#{dtRContextMenuView}")
    private RecomendacionContextMenuView contextMV;
```

El mismo tiene instanciado a varios beans:

- AyudaBusquedaSelectOneMenuView: es la ayuda de los distintas valores de Relaciones y Áreas Temáticas mostrados
- Recomendador: usado para obtener los valores de la última búsqueda
- RecomendacionContextMenuView: usado para obtener y mostrar los valores de la búsqueda deseada por el usuario.

Todos los campos de la misma, son atributos propios del bean, cada vez que se referencian y se usan, son accedidos por los getters y los setters de este. Los campos que presentan una ayuda tienen el siguiente formato en la página:

```
<p:selectOneMenu id="ID" value="#{recomendacionService.ATRIBUTO_AYUDA}"
panelStyle="width:180px effect="fade" var="var" style="width:160px" filter="true"
filterMatchMode="startsWith">
	<f:selectItems value="#{abSelectOneMenuView.AYUDA_QUE_CORRESPONDA" var="var"
	itemLabel="#{var}" itemValue="#{var}" />
</p:selectOneMenu>
```

Como se puede apreciar, el valor seleccionado, se guarda directamente en el bean en su respectivo atributo. El bean ayuda, es usado solamente para la muestra de valores correspondiente al campo en cuestión.

El checkbox Recordar Selección Anterior, lo que hace es instanciar al Recomendador, y consultarle los valores de la última búsqueda realizada por el usuario y presentarlos, además de actualizar los valores de los campos mostrados por la página:

```
<form id="form1" name="form1" method="post" action="">
	<p:outputLabel value="Recordar Selección Anterior " />
	<p:selectBooleanCheckbox value="#{recomendacionService.selAnterior}"
id="selAnterior">
	<p:ajax update="relUsuario"
listener="#{recomendacionService.updateDataFSelAnterior}"/>
	<p:ajax update="distUno"
listener="#{recomendacionService.updateDataFSelAnterior}"/>
	<p:ajax update="distDos"
listener="#{recomendacionService.updateDataFSelAnterior}"/>
	<p:ajax update="relEsperada"
listener="#{recomendacionService.updateDataFSelAnterior}"/>
	<p:ajax update="categoria"
listener="#{recomendacionService.updateDataFSelAnterior}"/>
	</p:selectBooleanCheckbox>
</form>
```

El botón Salir, lo que hace es reiniciar los atributos del bean a su estado vacío,

borrarlos valores de la recomendación anterior, y además, cerrar la página vía javaScript:

```
<p:commandButton id="busquedaSalir" value="Salir"
actionListener="#{recomendacionService.busquedaSalir}" icon="ui-icon-logout"
ajax="false" oncomplete="handleBusquedaOut()"/>
```

```
public void busquedaSalir() {
	contextMV.destroy();
	this.relUsuario="TODOS";
	this.distUno=false;
	this.distDos=false;
	this.distancia=1;
	this.areaTematica="TODOS";
	this.relEsperada="TODOS";
	this.selAnterior=false;
}
```

```
<script type="text/javascript">
	function handleBusquedaOut() {
		PF('dlg').hide();
	}
</script>
```

El botón Aceptar, tiene un actionListener busquedaAceptar, el cual asocia los distintos atributos del bean dtRContextMenuView, así como también ejecutar el método init. Este lo que hace, es referenciar al Recomendador, y ejecutar el proceso de recomendación dado los datos de entrada (usuario, contexto(red social), relación búsqueda, distancia, área temática, relación resultado). Una vez ejecutado esto, se llama al dialogo para visualizar las recomendaciones.

```
public void busquedaAceptar() {

contextMV.setRelUsuario(relUsuario.equalsIgnoreCase("TODOS")?"Relacion":relUsuario);
	contextMV.setDistUno(distUno);
	contextMV.setDistDos(distDos);
contextMV.setAreaTematica(areaTematica.equalsIgnoreCase("TODOS")?"AreaTematica":areaTem
atica);

contextMV.setRelEsperada(relEsperada.equalsIgnoreCase("TODOS")?"Relacion":relEsperada);
	contextMV.init();
}
```

```
<p:commandLink action="#{recomendacionService.busquedaAceptar}" update="dlgRB"
oncomplete="PF('dlgRB').show()">
	<p:commandButton id="busquedaAceptar" value="Aceptar" icon="ui-icon-search" />
</p:commandLink>
```

El bean dtRContextMenuView, es el encargado de mantener los datos para la visualización y puntuar/dejar pendiente de puntuar -> recomendaciones/recursos sin evaluaciones.

```
@ManagedBean(name="dtRContextMenuView")
```

```
@SessionScoped
public class RecomendacionContextMenuView
       private List<DatoRecomendacion> recomendaciones;
       private List<DatoRecomendacion> recomendacionesSEvaluar;
       private DatoRecomendacion selectedRecomendacion;
       private DatoRecomendacion selectedRecomendacionSE;

       @ManagedProperty(value="#{recomendadorBean}")
       private Recomendador rService;
```

El mismo tiene referencia al Recomendador, para inicializarse con la recomendación y con los recursos sin evaluaciones, puntuar, o dejar pendientes de puntuación recomendaciones/recursos. También 2 colecciones el cual contiene las recomendaciones obtenidas, como así también los recursos que aun no fueron evaluados. Estos valores son los que se obtienen por el algoritmo de recomendación dado los valores obtenidos de la página anterior (usuario, contexto(red social), relación sobre búsqueda, área temática y relación esperada para la recomendación). Y por ultimo, los elementos seleccionados de ambas colecciones.

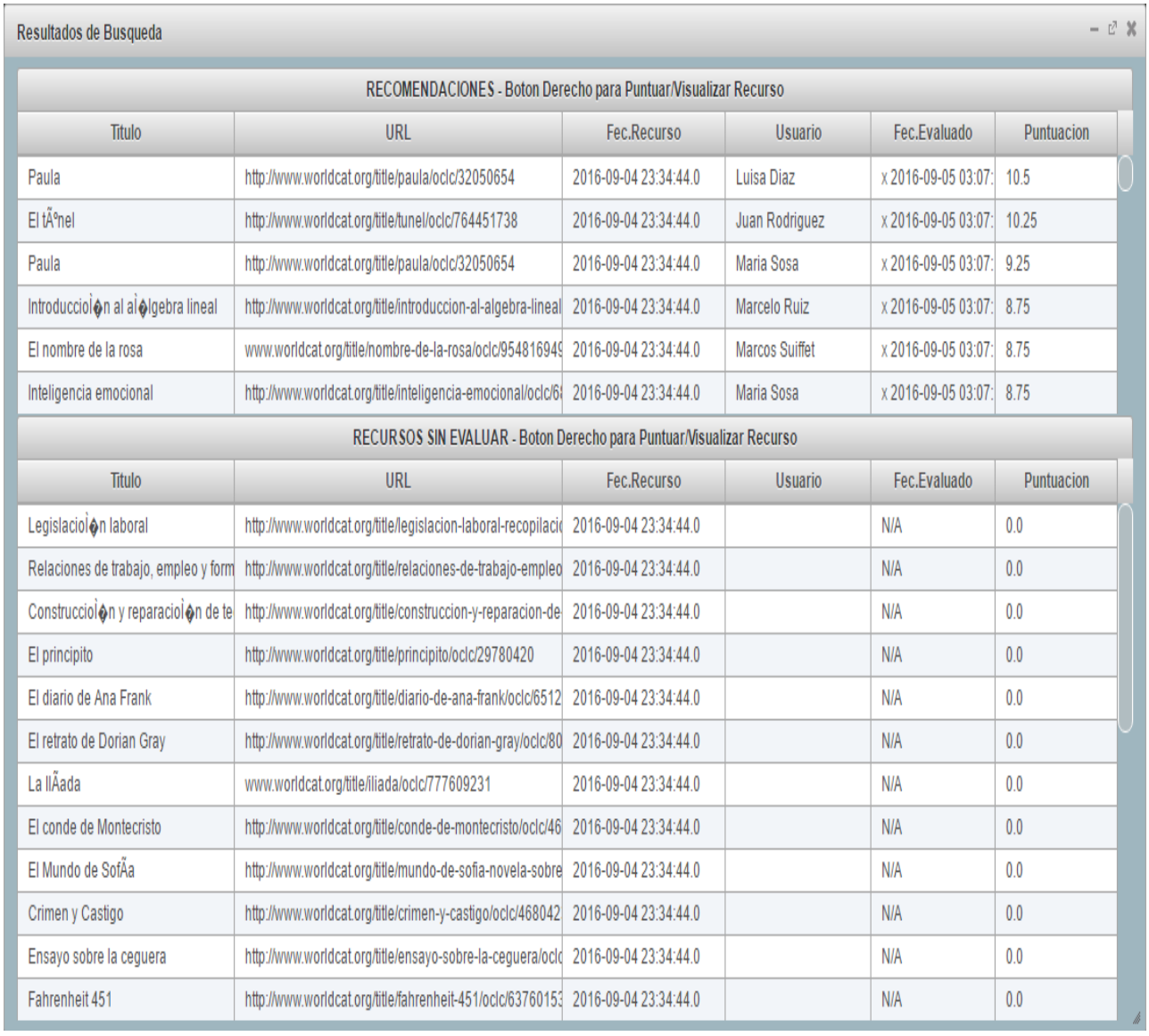
```
    public void init() {
       List<DatoRecomendacion> r = createRecomendaciones();
       List<DatoRecomendacion> rse = createRecomendacionesSinEvaluar();

       recomendaciones = new ArrayList<DatoRecomendacion>();
        recomendaciones.addAll(r);
        recomendacionesSEvaluar = new ArrayList<DatoRecomendacion>();
        recomendacionesSEvaluar.addAll(rse);
    }

   public List<DatoRecomendacion> createRecomendaciones() {
       List<DatoRecomendacion> list = new ArrayList<DatoRecomendacion>();
       list = rService.getRecomendaciones(areaTematica,
rService.getIdUsuario(usrLogueado.getUsuario()), relUsuario, (distUno==true?1:2),
relEsperada);
       return list;
   }

   public List<DatoRecomendacion> createRecomendacionesSinEvaluar() {
       List<DatoRecomendacion> list2 = new ArrayList<DatoRecomendacion>();
       list2 = rService.getRecursosSinEvaluar(areaTematica,
rService.getDominioRedSocial());
        return list2;
   }
```

Una vez hecha la búsqueda, el sistema desplegara en la página web antes mencionada los resultados:



Resultados de Busqueda

RECOMENDACIONES - Boton Derecho para Puntuar/Visualizar Recurso

Titulo	URL	Fec.Recurso	Usuario	Fec.Evaluado	Puntuacion
Paula	http://www.worldcat.org/title/paula/oclc/32050654	2016-09-04 23:34:44.0	Luisa Diaz	x 2016-09-05 03:07:	10.5
El tÃºnel	http://www.worldcat.org/title/tunel/oclc/764451738	2016-09-04 23:34:44.0	Juan Rodriguez	x 2016-09-05 03:07:	10.25
Paula	http://www.worldcat.org/title/paula/oclc/32050654	2016-09-04 23:34:44.0	Maria Sosa	x 2016-09-05 03:07:	9.25
Introduccioì¿½n al aì¿½lgebra lineal	http://www.worldcat.org/title/introduccion-al-algebra-lineal	2016-09-04 23:34:44.0	Marcelo Ruiz	x 2016-09-05 03:07:	8.75
El nombre de la rosa	www.worldcat.org/title/nombre-de-la-rosa/oclc/954816949	2016-09-04 23:34:44.0	Marcos Suiffet	x 2016-09-05 03:07:	8.75
Inteligencia emocional	http://www.worldcat.org/title/inteligencia-emocional/oclc/6	2016-09-04 23:34:44.0	Maria Sosa	x 2016-09-05 03:07:	8.75

RECURSOS SIN EVALUAR - Boton Derecho para Puntuar/Visualizar Recurso

Titulo	URL	Fec.Recurso	Usuario	Fec.Evaluado	Puntuacion
Legislacioì¿½n laboral	http://www.worldcat.org/title/legislacion-laboral-recopilaci	2016-09-04 23:34:44.0		N/A	0.0
Relaciones de trabajo, empleo y form	http://www.worldcat.org/title/relaciones-de-trabajo-empleo	2016-09-04 23:34:44.0		N/A	0.0
Construccioì¿½n y reparacioì¿½n de te	http://www.worldcat.org/title/construccion-y-reparacion-de	2016-09-04 23:34:44.0		N/A	0.0
El principito	http://www.worldcat.org/title/principito/oclc/29780420	2016-09-04 23:34:44.0		N/A	0.0
El diario de Ana Frank	http://www.worldcat.org/title/diario-de-ana-frank/oclc/6512	2016-09-04 23:34:44.0		N/A	0.0
El retrato de Dorian Gray	http://www.worldcat.org/title/retrato-de-dorian-gray/oclc/80	2016-09-04 23:34:44.0		N/A	0.0
La lÃada	www.worldcat.org/title/iliada/oclc/777609231	2016-09-04 23:34:44.0		N/A	0.0
El conde de Montecristo	http://www.worldcat.org/title/conde-de-montecristo/oclc/46	2016-09-04 23:34:44.0		N/A	0.0
El Mundo de SofÃa	http://www.worldcat.org/title/mundo-de-sofia-novela-sobre	2016-09-04 23:34:44.0		N/A	0.0
Crimen y Castigo	http://www.worldcat.org/title/crimen-y-castigo/oclc/468042	2016-09-04 23:34:44.0		N/A	0.0
Ensayo sobre la ceguera	http://www.worldcat.org/title/ensayo-sobre-la-ceguera/oclc	2016-09-04 23:34:44.0		N/A	0.0
Fahrenheit 451	http://www.worldcat.org/title/fahrenheit-451/oclc/63760153	2016-09-04 23:34:44.0		N/A	0.0

Figura 12 - Despliegue de Recomendaciones Obtenidas

En el mismo, se visualizan como primera instancia las recomendaciones hechas por el sistema, ordenadas de mayor a menor dada las puntuaciones obtenidas, como así también, aquellos recursos que nunca fueron evaluados pero que cumplen con la condición de búsqueda. Los datos a mostrar son: Titulo, URL, Fecha Recomendación/Recurso, Fecha de Evaluado y su puntuación obtenida. Si nos paramos sobre el que estemos interesado, lo seleccionamos y apretamos el botón derecho del mouse, aparecerá 3 opciones, puntuar el recurso seleccionado. visualización del mismo en un browser (dada su url), y visualización de la

descripción del mismo:

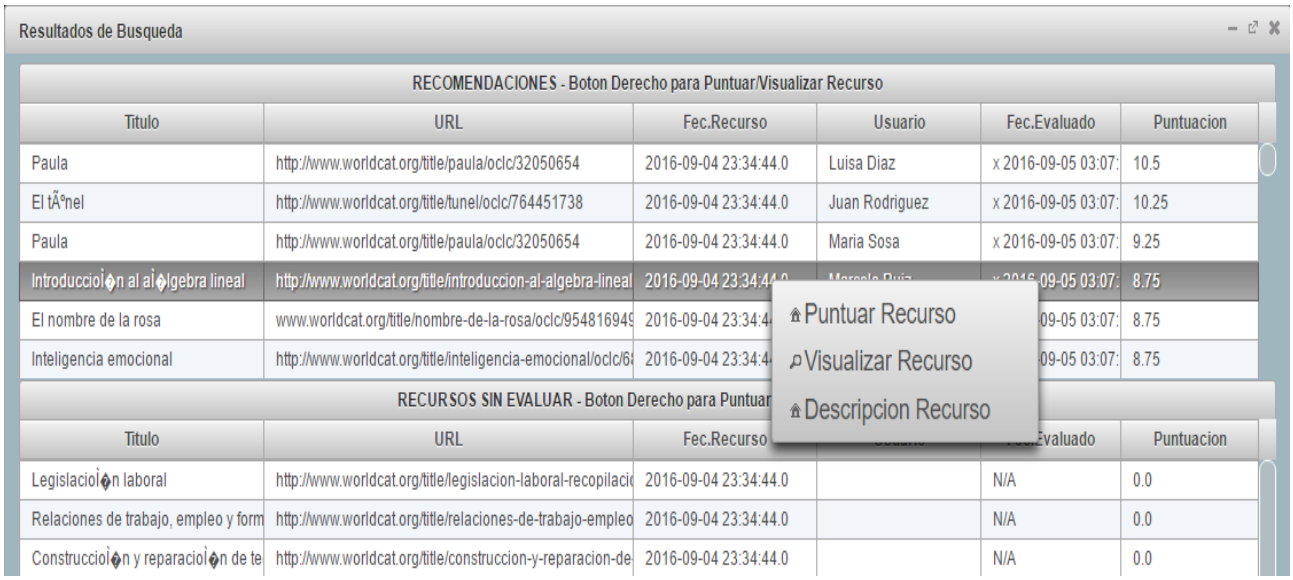


Figura 13 - Acciones sobre las Recomendaciones obtenidas

A continuación se mostrara las 3 acciones sobre los recursos:

Visualizar Recurso:

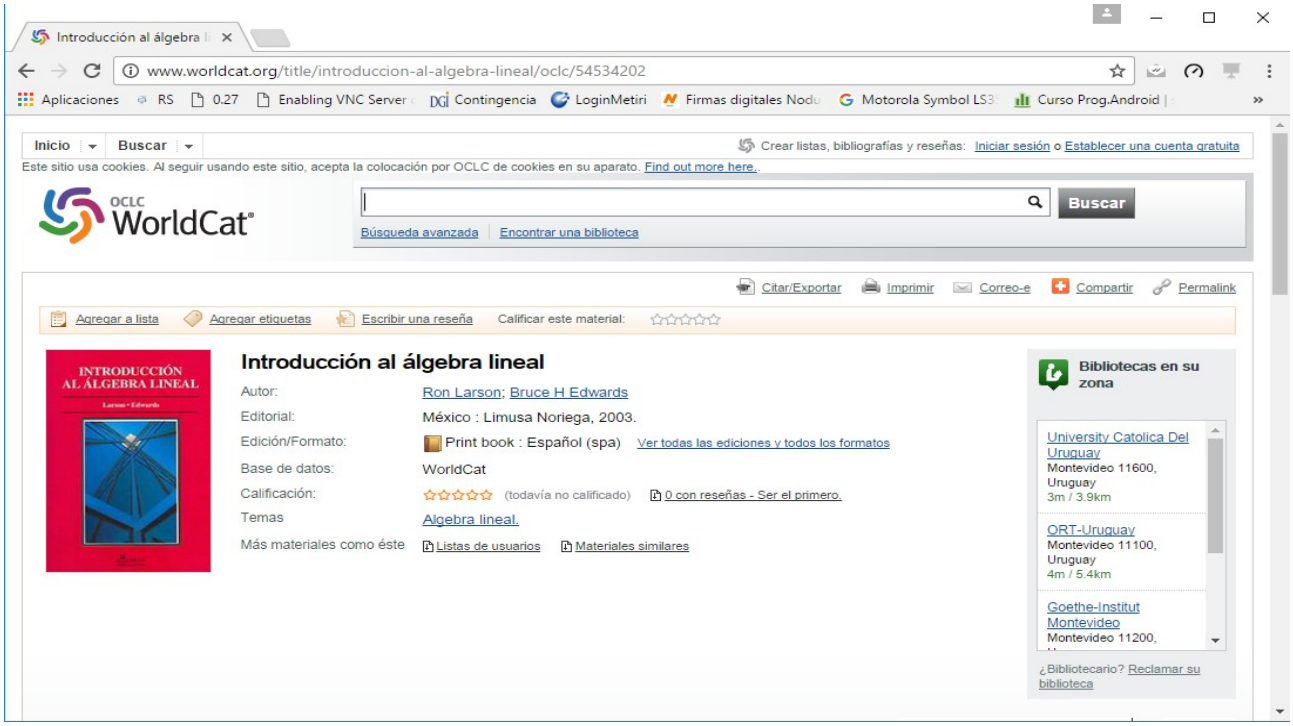


Figura 14 - Visualización del recurso en Navegador Web I

Descripción Recurso:

Figura 15 - Descripción detallada del Recurso I

Puntuar Recurso:

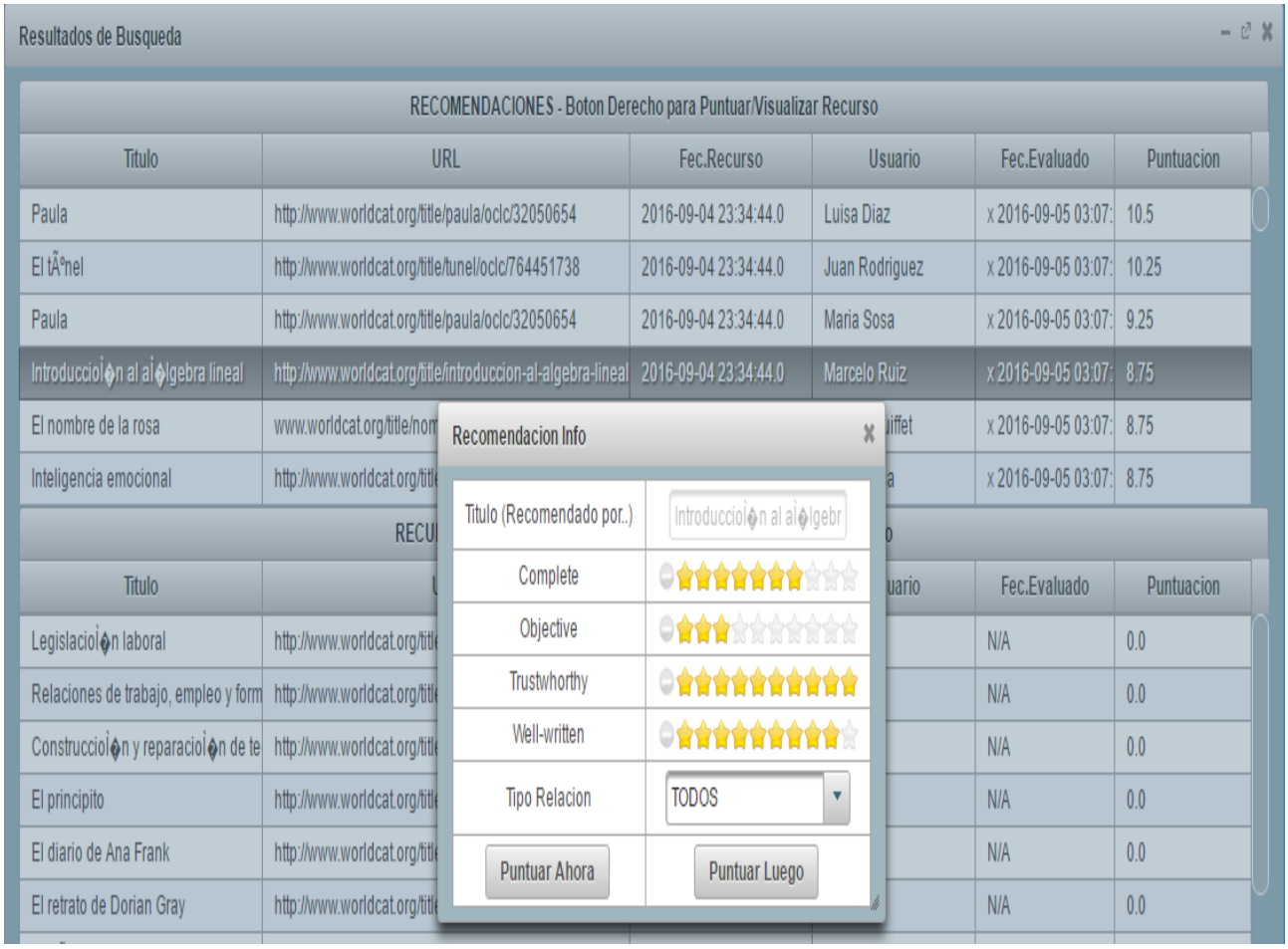


Figura 16 - Evaluación de un Recurso

Aquí tenemos 2 opciones; la primera es realizar la puntuación deseada, y luego puntuar el recurso (botón Puntuar Ahora), y la otra es dejarlo para puntuar luego (botón Puntuar Luego). Ambas opciones, borran del resultado de búsqueda del Sistema Recomendador, el recurso evaluado o próximo a evaluar. Esta opciones existen para las 2 grillas de datos, recomendaciones como recursos sin evaluar. Un dato importante a la hora de puntuar, es que se le solicita al usuario sobre que relación se debiera recomendar ese recurso, por defecto TODOS (Toda relación, que es equivalente a la clase Relacion en la ontología de vocabulario para las relaciones):

```
public void puntuarRecursoAhora(){
	puntuarRecurso(selectedRecomendacion);
	recomendaciones.remove(selectedRecomendacion);
	selectedRecomendacion = null;
}

public void puntuarRecursoLuego(){
	puntuarLuego(selectedRecomendacion);
	recomendaciones.remove(selectedRecomendacion);
	selectedRecomendacion = null;
}

public void puntuarRecursoAhoraSE(){
	recomendacionesSEvaluar.remove(selectedRecomendacionSE);
	puntuarRecurso(selectedRecomendacionSE);
	selectedRecomendacionSE = null;
}

public void puntuarRecursoLuegoSE(){
	recomendacionesSEvaluar.remove(selectedRecomendacionSE);
	puntuarLuego(selectedRecomendacionSE);
	selectedRecomendacionSE = null;
}


public void puntuarRecurso (DatoRecomendacion aPuntuarAhora) {
	if (aPuntuarAhora.getRelacion().trim().equalsIgnoreCase(""))

	rService.addRecursoEvaluado(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getIdRecurso(aPuntuarAhora.getTitulo(), aPuntuarAhora.getDescripcion(),
aPuntuarAhora.getUrl(), areaTematica, rService.getDominioRedSocial()), areaTematica,
rService.getDominioRedSocial(), aPuntuarAhora.getNom_atributo1(),
aPuntuarAhora.getPuntaje1(), aPuntuarAhora.getNom_atributo2(),
aPuntuarAhora.getPuntaje2(), aPuntuarAhora.getNom_atributo3(),
aPuntuarAhora.getPuntaje3(), aPuntuarAhora.getNom_atributo4(),
aPuntuarAhora.getPuntaje4(), relEsperada);
	else

	rService.addRecursoEvaluado(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getIdRecurso(aPuntuarAhora.getTitulo(), aPuntuarAhora.getDescripcion(),
aPuntuarAhora.getUrl(), areaTematica, rService.getDominioRedSocial()), areaTematica,
rService.getDominioRedSocial(), aPuntuarAhora.getNom_atributo1(),
aPuntuarAhora.getPuntaje1(), aPuntuarAhora.getNom_atributo2(),
aPuntuarAhora.getPuntaje2(), aPuntuarAhora.getNom_atributo3(),
```

```
aPuntuarAhora.getPuntaje3(), aPuntuarAhora.getNom_atributo4(),
aPuntuarAhora.getPuntaje4(), aPuntuarAhora.getRelacion());
}

public void puntuarLuego (DatoRecomendacion aPuntuarLuego) {
      rService.addRecursoAEvaluar(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getIdRecurso(aPuntuarLuego.getTitulo(), aPuntuarLuego.getDescripcion(),
aPuntuarLuego.getUrl(), areaTematica, rService.getDominioRedSocial()), areaTematica,
rService.getDominioRedSocial(), aPuntuarLuego.getNom_atributo1(),
aPuntuarLuego.getPuntaje1(), aPuntuarLuego.getNom_atributo2(),
aPuntuarLuego.getPuntaje2(), aPuntuarLuego.getNom_atributo3(),
aPuntuarLuego.getPuntaje3(), aPuntuarLuego.getNom_atributo4(),
aPuntuarLuego.getPuntaje4(), relEsperada);
}
```

3.5.2.2 Puntuar Recurso

El usuario podrá puntuar los recursos que no hayan sido puntuados en el momento. Para esto deberá seleccionar la opción puntuar recursos, el cual desplegará el set de recursos pendientes de puntuar. El sistema en este caso, llamara al dialogo recursos_pendientes_puntuar, el cual es administrado por el Bean dtRContextMenuViewRecurso:

```
      <p:dialog      header="Puntuar      Recurso"      id="dlgPR"      widgetVar="dlgPR"
minimizable="true" maximizable="true" styleClass="puntuarRecurso">
      <p:ajax event="close" listener="#{dtRContextMenuViewRecurso.recursoExit}"
oncomplete="handlePuntuarRecursoOut()" update="dlgPR"/>

          <p:contextMenu for="recursos">
              <p:menuitem value="Puntuar Recurso" update="recursoDetail" icon="ui-
icon-home" oncomplete="PF('recursoDialog').show()"/>
              <p:menuitem value="Visualizar Recurso" icon="ui-icon-search"
actionListener="#{dtRContextMenuViewRecurso.actionRecursoListenerRedirect}"
ajax="false" target="_blank"/>
              <p:menuitem value="Recurso Descripcion" update="recursoDDetail"
icon="ui-icon-home" oncomplete="PF('recursoDDialog').show()"/>
          </p:contextMenu>


          <p:dataTable id="recursos" var="recurso"
value="#{dtRContextMenuViewRecurso.recursos}" rowKey="#{recurso.key}"
                       selection="#{dtRContextMenuViewRecurso.selectedRecurso}"
selectionMode="single" scrollable="true" scrollHeight="250" resizableColumns="true">
              <f:facet name="header">
                  Boton derecho para ver Opciones
              </f:facet>
              <p:column headerText="Titulo" style="width:80px;">
                  <h:outputText value="#{recurso.titulo}" />
              </p:column>
              <p:column headerText="Descripcion" style="width:150px;">
                  <h:outputText value="#{recurso.descripcion}" />
              </p:column>
              <p:column headerText="URL" style="width:150px;">
                  <h:outputText value="#{recurso.url}" />
```

```
                </p:column>
                <p:column headerText="Fec.Recurso" style="width:75px;">
                    <h:outputText value="#{recurso.fecha_rec}" />
                </p:column>
            </p:dataTable>
    </p:dialog>

@ManagedBean(name="dtRContextMenuViewRecurso")
@SessionScoped
public class RecursoContextMenuView
    private List<DatoRecurso> recursos;
    private DatoRecurso selectedRecurso;

@ManagedProperty(value="#{recursoService}")
    private RecursoServices service;
```

Este bean, tiene como atributos, una colección de Recursos pendientes de puntuar, el cual son las mostradas en el dialogo en cuestión, el Recurso seleccionado para poder tomar acciones luego, como también el respectivo bean Service, encargado de obtener los recursos a puntuar por el usuario.

Cuando es ejecutado el botón o link para visualizar los recursos, este tiene un método el cual es ejecutado como action, el cual inicializa el bean, y obtiene los recursos pendientes de puntuar por el usuario a través del Service. Este lo único que hace, es realizar una llamada al método del Recomendador que solicita estos datos, por lo que solamente actúa como intermediario.

```
<p:commandLink    value="Puntuar    Recurso"    action="#{dtRContextMenuViewRecurso.init}"
update=":dlgForm:dlgPR" oncomplete="PF('dlgPR').show()" styleClass="zoomIt"/>

    public void init() {
       recursos = service.createRecursos();
    }
```

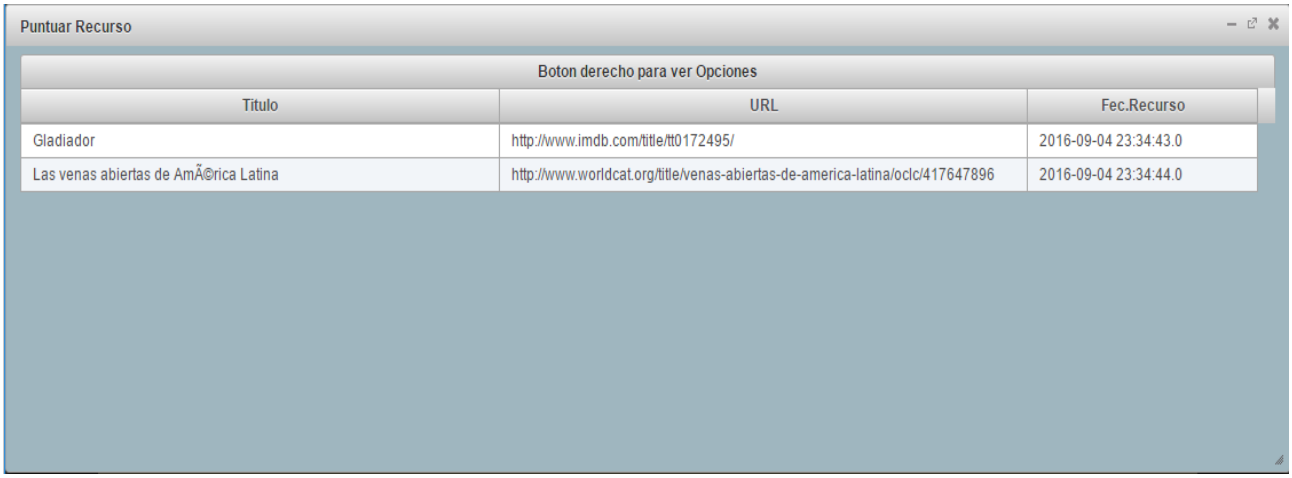


Figura 17 - Recursos pendientes de Evaluación

Luego que este es ejecutado, es abierto el diálogo de puntuar recurso, visualizando los valores correspondientes. Esta página cuenta con las mismas opciones que la de resultado de búsqueda, Visualizar Recurso, Puntuar Recurso y Descripción del Recurso, el cual se activará para el recurso seleccionado. Para esto, se selecciona uno, y se presiona el botón derecho del mouse, tal cual muestra la siguiente imagen:

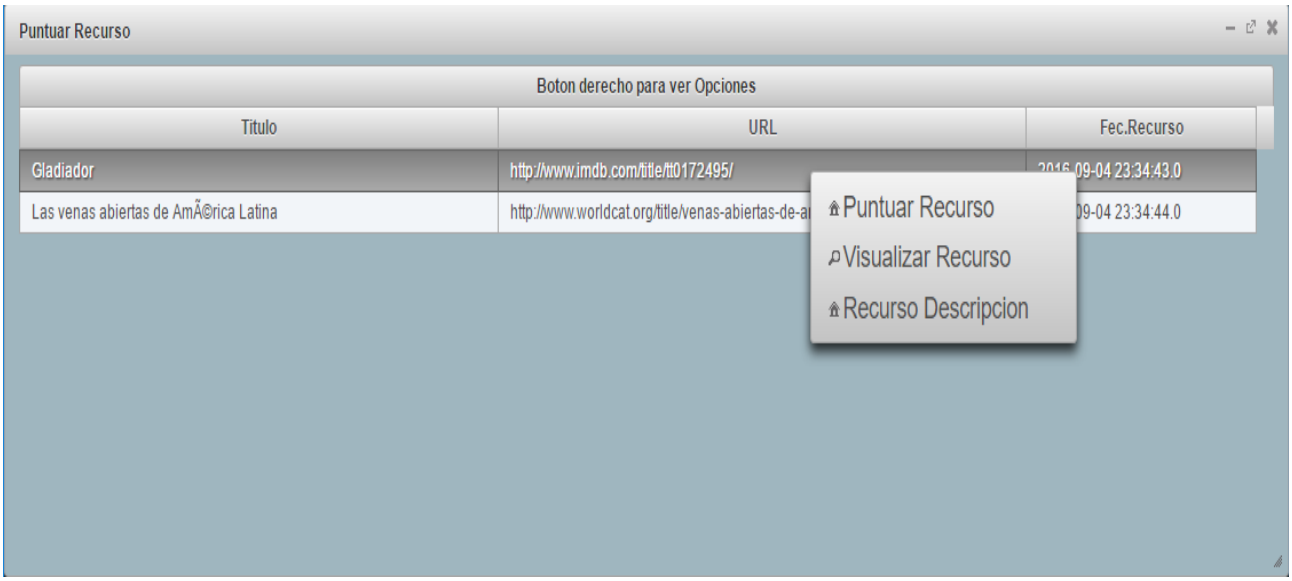


Figura 18 - Acciones sobre los Recurso pendientes de Evaluar

Para Visualizar Recurso, se abre un browser y nos lleva a la url del recurso (ídem caso anterior).

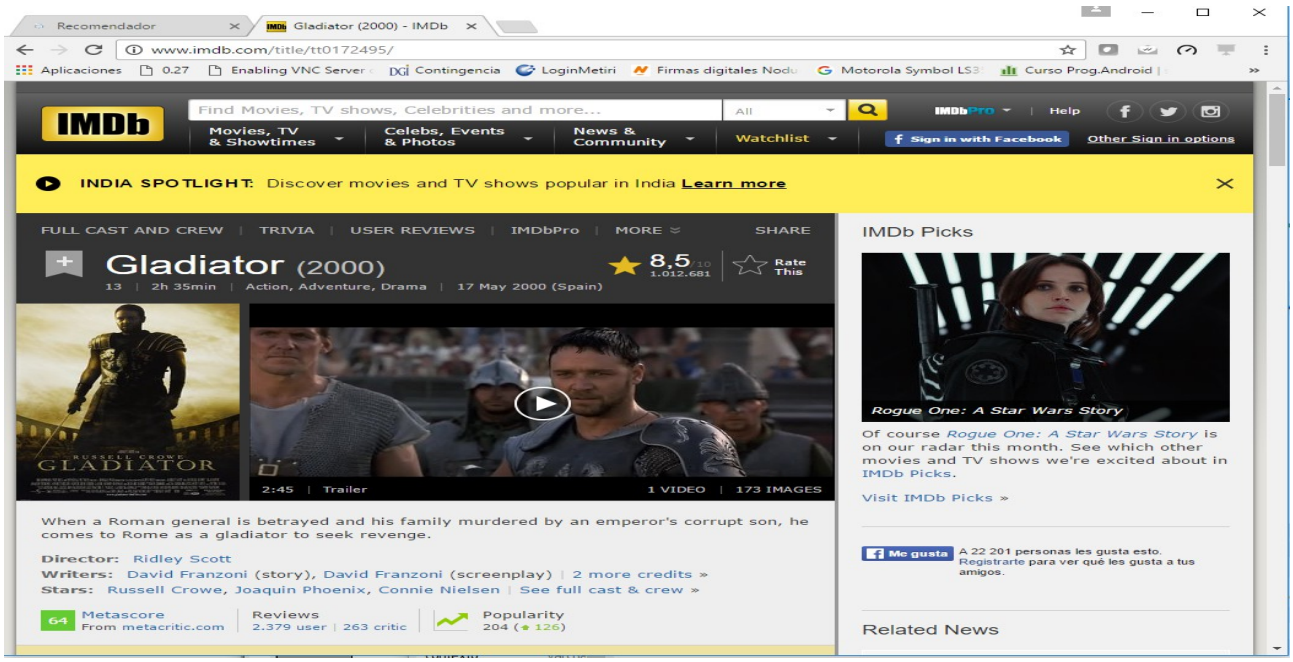


Figura 19 - Visualización del recurso en Navegador Web II

Para visualizar la descripción de un recurso, se abre un dialogo conteniendo los datos de este:

Figura 20 - Descripción detallada del Recurso II

Para el caso de Puntuar Recurso, se habilita el diálogo para puntuar dado los atributos definidos según el contexto de la red:

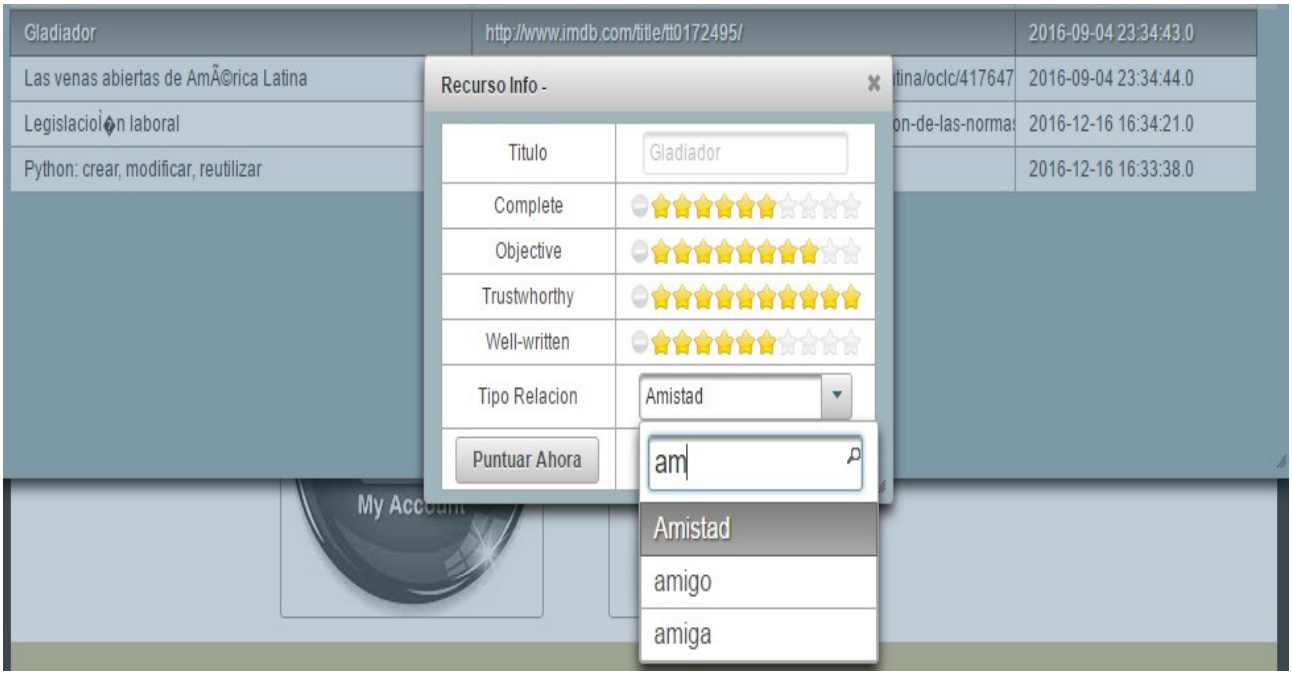


Figura 21 - Evaluación de un Recurso pendiente

Aquí solo tendremos la opción de puntuar o cancelar para puntuar otro recurso.
A continuación la firma del bean Service correspondiente al evaluar Recursos

pendientes recursoService. El cometido de este, no es mas que referenciar al Recomendador, y llamarlo para obtener la lista de recursos pendientes de puntuar por parte de este usuario, como así también, la puntuación otorgada por el usuario al recurso en cuestión, y luego la eliminación de este sobre la tabla de recursos pendientes de evaluar por el mismo.

```
@ManagedBean(name = "recursoService")
@ApplicationScoped
public class RecursoServices implements Serializable {
    @ManagedProperty(value="#{recomendadorBean}")
    private Recomendador rService;

       public List<DatoRecurso> createRecursos() {
              List<DatoRecurso> list = new ArrayList<DatoRecurso>();
              list =
rService.getRecursosPendientesEval(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getDominioRedSocial());

         return list;
       }

       public void puntuarRecurso (DatoRecurso aPuntuarAhora) {

       rService.addRecursoEvaluado(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getIdRecurso(aPuntuarAhora.getTitulo(), aPuntuarAhora.getDescripcion(),
aPuntuarAhora.getUrl(), aPuntuarAhora.getAreaTematica(),
rService.getDominioRedSocial()), aPuntuarAhora.getAreaTematica(),
rService.getDominioRedSocial(), aPuntuarAhora.getNom_atributo1(),
aPuntuarAhora.getPuntaje1(), aPuntuarAhora.getNom_atributo2(),
aPuntuarAhora.getPuntaje2(), aPuntuarAhora.getNom_atributo3(),
aPuntuarAhora.getPuntaje3(), aPuntuarAhora.getNom_atributo4(),
aPuntuarAhora.getPuntaje4(), aPuntuarAhora.getRelacion());

       rService.removeRecursoAEvaluar(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getIdRecurso(aPuntuarAhora.getTitulo(), aPuntuarAhora.getDescripcion(),
aPuntuarAhora.getUrl(), aPuntuarAhora.getAreaTematica(),
rService.getDominioRedSocial()), aPuntuarAhora.getAreaTematica(),
rService.getDominioRedSocial(), aPuntuarAhora.getRelacion());
       }
```

3.5.2.3 Historial de Consultas Realizadas

El usuario tendrá la posibilidad de ver las consultas hechas a lo largo del tiempo, es decir, las solicitudes de recomendaciones que ha efectuado el mismo. Para esto, deberá seleccionar la opción Historial de Consultas realizadas. El mismo será atendido por el dialogo historial_busqueda. La misma es administrada por el Bean dtRContextMenuViewHistorial.

```
      <p:dialog header="Historia Consultas Realizadas" id="dlgHCR" widgetVar="dlgHCR"
minimizable="true" maximizable="true" styleClass="hConsultasRealizadas">
      <p:ajax event="close" listener="#{dtRContextMenuViewHistorial.historialExit}"
oncomplete="handleHConsultasRealizadasOut()" update="dlgRB"/>

          <p:dataTable id="historiales" var="historial"
value="#{dtRContextMenuViewHistorial.historiales}" rowKey="#{historial.idUsuario}"
                       selection="#{dtRContextMenuViewHistorial.selectedHistorial}"
selectionMode="single" scrollable="true" scrollHeight="600" resizableColumns="true">
              <f:facet name="header">
                  Historial de Busqueda
              </f:facet>
              <p:column headerText="Area Tematica" style="width:100px;">
                  <h:outputText value="#{historial.areaTematica}" />
              </p:column>
              <p:column headerText="Relacion Persona" style="width:100px;">
                  <h:outputText value="#{historial.relacionPersona}" />
              </p:column>
              <p:column headerText="Distancia" style="width:80px;">
                  <h:outputText value="#{historial.distancia}" />
              </p:column>
              <p:column headerText="Relacion Resultado" style="width:100px;">
                  <h:outputText value="#{historial.relacionResultado}" />
              </p:column>
              <p:column headerText="Fecha Consulta" style="width:100px;">
                  <h:outputText value="#{historial.fechaConsulta}" />
              </p:column>
          </p:dataTable>
    </p:dialog>
```

```
@ManagedBean(name="dtRContextMenuViewHistorial")
@SessionScoped
public class HistorialContextMenuView
    private List<DatoConsulta> historiales;

    @ManagedProperty(value="#{recursoService}")
    private RecursoServices service;
```

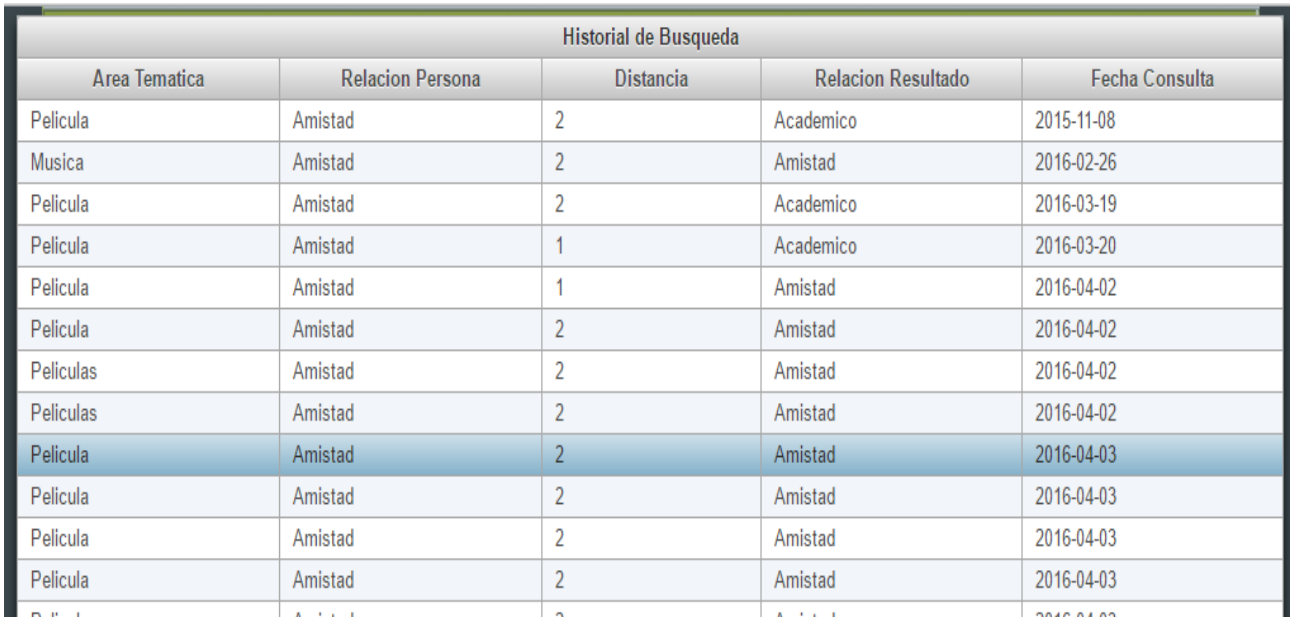
Este bean, tiene como atributos, una colección de Consultas, el cual son las mostradas en el dialogo en cuestión, como también el respectivo bean Service, encargado de obtener las consultas realizadas por el usuario.

Como en los casos anteriores, el mismo tiene un método el cual es ejecutado como action, cuando el botón es apretado, el cual inicializa el bean, y obtiene las consultas realizadas por el usuario a través del Service, que al igual que los casos anteriores, no hace mas que llamar a un servicio de Recomendador.

```
<p:commandLink value="Historial Consultas Realizadas"
        action="#{dtRContextMenuViewHistorial.init}" update=":dlgForm:dlgHCR"
        oncomplete="PF('dlgHCR').show()" styleClass="zoomIt"/>

    public void init() {
```

```
        historiales = service.createHistoriales();
    }
```



Historial de Busqueda

Area Tematica	Relacion Persona	Distancia	Relacion Resultado	Fecha Consulta
Pelicula	Amistad	2	Academico	2015-11-08
Musica	Amistad	2	Amistad	2016-02-26
Pelicula	Amistad	2	Academico	2016-03-19
Pelicula	Amistad	1	Academico	2016-03-20
Pelicula	Amistad	1	Amistad	2016-04-02
Pelicula	Amistad	2	Amistad	2016-04-02
Peliculas	Amistad	2	Amistad	2016-04-02
Peliculas	Amistad	2	Amistad	2016-04-02
Pelicula	Amistad	2	Amistad	2016-04-03
Pelicula	Amistad	2	Amistad	2016-04-03
Pelicula	Amistad	2	Amistad	2016-04-03
Pelicula	Amistad	2	Amistad	2016-04-03

Figura 22 - Historial de Solicitudes de Recomendación realizadas por el Usuario

A continuación la firma del bean Service correspondiente al historial de Recursos Evaluados historialService.

```
@ManagedBean(name = "historialService")
@ApplicationScoped
public class HistorialServices implements Serializable {
    @ManagedProperty(value="#{recomendadorBean}")
    private Recomendador rService;

public List<DatoConsulta> createHistoriales() {
            List<DatoConsulta> list = new ArrayList<DatoConsulta>();
            list =
rService.getListaConsultasUsuario(rService.getIdUsuario(usrLogueado.getUsuario()),
rService.getDominioRedSocial());

        return list;

    }
```

3.5.2.4 Historial de Recursos Evaluados

El usuario podrá consultar todos los recursos evaluados a lo largo del tiempo, para esto deberá de acceder a la opción Historial de Recursos Evaluados. El mismo será atendido por el dialog historial_recursos_evaluados. La misma es administrada por el Bean dtRContextMenuViewHistorialRE

```
      <p:dialog header="Historia Recursos Evaluados" id="dlgHRE" widgetVar="dlgHRE"
minimizable="true" maximizable="true" styleClass="hRecursosEvaluados">
      <p:ajax event="close" listener="#{dtRContextMenuViewHistorialRE.historialREExit}"
oncomplete="handleHRecursosEvaluadosOut()" update="dlgRB"/>

          <p:dataTable id="historialesRE" var="historialRE"
value="#{dtRContextMenuViewHistorialRE.historiales}" rowKey="#{historialRE.key}"
                       selection="#{dtRContextMenuViewHistorialRE.selectedHistorialRE}"
selectionMode="single" scrollable="true" scrollHeight="450" resizableColumns="true">
              <f:facet name="header">
                  Historial de Recursos Evaluados
              </f:facet>
              <p:column headerText="Id Usr" style="width:80px;">
                  <h:outputText value="#{historialRE.idUsuario}" />
              </p:column>
              <p:column headerText="Id Rso" style="width:20px;">
                  <h:outputText value="#{historialRE.idRecurso}" />
              </p:column>
              <p:column headerText="Titulo" style="width:70px;">
                  <h:outputText value="#{historialRE.titulo}" />
              </p:column>
              <p:column headerText="Area Tematica" style="width:80px;">
                  <h:outputText value="#{historialRE.areaTematica}" />
              </p:column>
              <p:column headerText="Contexto" style="width:60px;">
                  <h:outputText value="#{historialRE.contexto}" />
              </p:column>
              <p:column headerText="#{dtRContextMenuViewHistorialRE.nombreAtributo1}"
style="width:70px;">
                  <h:outputText value="#{historialRE.puntaje1}" />
              </p:column>
              <p:column headerText="#{dtRContextMenuViewHistorialRE.nombreAtributo2}"
style="width:70px;">
                  <h:outputText value="#{historialRE.puntaje2}" />
              </p:column>
              <p:column headerText="#{dtRContextMenuViewHistorialRE.nombreAtributo3}"
style="width:70px;">
                  <h:outputText value="#{historialRE.puntaje3}" />
              </p:column>
              <p:column headerText="#{dtRContextMenuViewHistorialRE.nombreAtributo4}"
style="width:70px;">
                  <h:outputText value="#{historialRE.puntaje4}" />
              </p:column>
              <p:column headerText="Tipo Vinculo" style="width:70px;">
                  <h:outputText value="#{historialRE.tipoVinculo}" />
              </p:column>
              <p:column headerText="Fecha Eval" style="width:70px;">
                  <h:outputText value="#{historialRE.fechaEval}" />
              </p:column>
          </p:dataTable>


    </p:dialog>

@ManagedBean(name="dtRContextMenuViewHistorialRE")
@ViewScoped
```

```
public class HistorialContextMenuViewRE
    private List<DatoEvaluacion> historialesRE;

    @ManagedProperty(value="#{historialREService}")
    private HistorialREServices service;
    @ManagedProperty(value="#{recomendadorBean}")
    private Recomendador rService;
```

Este bean, tiene como atributos, una colección de Evaluaciones, el cual son las mostradas en el dialogo en cuestión, como también los bean de Recomendador y el respectivo Service, encargados de obtener los nombre de los atributos según el contexto de la red, como el de obtener las puntuaciones de los recursos evaluados por el usuario.

```
      public String getNombreAtributo1() {
            return rService.getNombreAtributo1();
      }
      public String getNombreAtributo2() {
            return rService.getNombreAtributo2();
      }
      public String getNombreAtributo3() {
            return rService.getNombreAtributo3();
      }
      public String getNombreAtributo4() {
            return rService.getNombreAtributo4();
      }
```

Como en los casos anteriores, el mismo tiene un método el cual es ejecutado como action, cuando el botón es apretado, el cual inicializa el bean, y obtiene los recursos evaluados por el usuario a través del Service, que al igual que los casos anteriores, no hace mas que llamar a un servicio de Recomendador.

```
<p:commandLink value="Historial Recursos Evaluados"
      action="#{dtRContextMenuViewHistorialRE.init}"update=":dlgForm:dlgHRE"
      oncomplete="PF('dlgHRE').show()" styleClass="zoomIt"/>

    public void init() {
      historialesRE = service.createHistoriales();
    }
```

La siguiente imagen muestra lo obtenido en pantalla al realizar la consulta para obtener los recursos evaluados para un usuario:

Id Usr	Id Rsc	Titulo	Area Temática	Contexto	Complete	Objective	Trustwhorthy	Well-written	Tipo Vinculo	Fecha Eval
diego.bastiani	18	Mujer Bonita	PeliculaRomantica	defecto	6	5	8	8	amiga	2016-09-05 03:
diego.bastiani	1	Forrest Gump	PeliculaDramatica	defecto	10	10	9	9	mejoresAmigos	2016-09-05 03:
diego.bastiani	7	El Caballero de	PeliculaDeAccion	defecto	6	5	10	7	Amistad	2016-09-05 03:
diego.bastiani	12	Toy Story	PeliculaAnimada	defecto	5	9	6	9	Familiar	2016-09-05 03:
diego.bastiani	48	El alquimista	LibroDeNovelas	defecto	8	6	8	5	Familiar	2016-09-05 03:
diego.bastiani	93	Quí mica: la ci	LibroDeQuimica	defecto	10	8	10	6	estudiante-curs	2016-09-05 03:
diego.bastiani	105	Php: paso a pas	LibroDePrograma	defecto	7	10	10	8	Academico	2016-09-05 03:
diego.bastiani	45	Don Quijote de l	LibroDeNovelas	defecto	8	7	9	6	Relacion	2016-09-05 03:

Figura 23 - Historial de Recursos Evaluados por el Usuario

A continuación la firma del bean Service correspondiente al historial de Recursos Evaluados historialREService.

```
@ManagedBean(name = "historialREService")
@ApplicationScoped
public class HistorialREServices implements Serializable {
    @ManagedProperty(value="#{recomendadorBean}")
    private Recomendador rService;

    public List<DatoEvaluacion> createHistoriales() {
            List<DatoEvaluacion> list = new ArrayList<DatoEvaluacion>();
            list =
rService.getListaRecursosEvaluadosUsuario(rService.getIdUsuario(usrLogueado.getUsuario(
)), rService.getDominioRedSocial());

        return list;
    }
```

3.5.2.5 Salir

Será utilizada para que el usuario salga del Sistema Recomendador.

3.5.3 Diseño y desarrollo de la interfaz gráfica del Administrador

Esta interfaz, esta diseñada para poder editar las parametrizaciones del Recomendador.

Se accede a través de la página web http://localhost:8080/ProyectoWeb/index.jsf

En la misma, estará el icono de login, el cual al clickearlo, aparecerá el acceso del administrador.

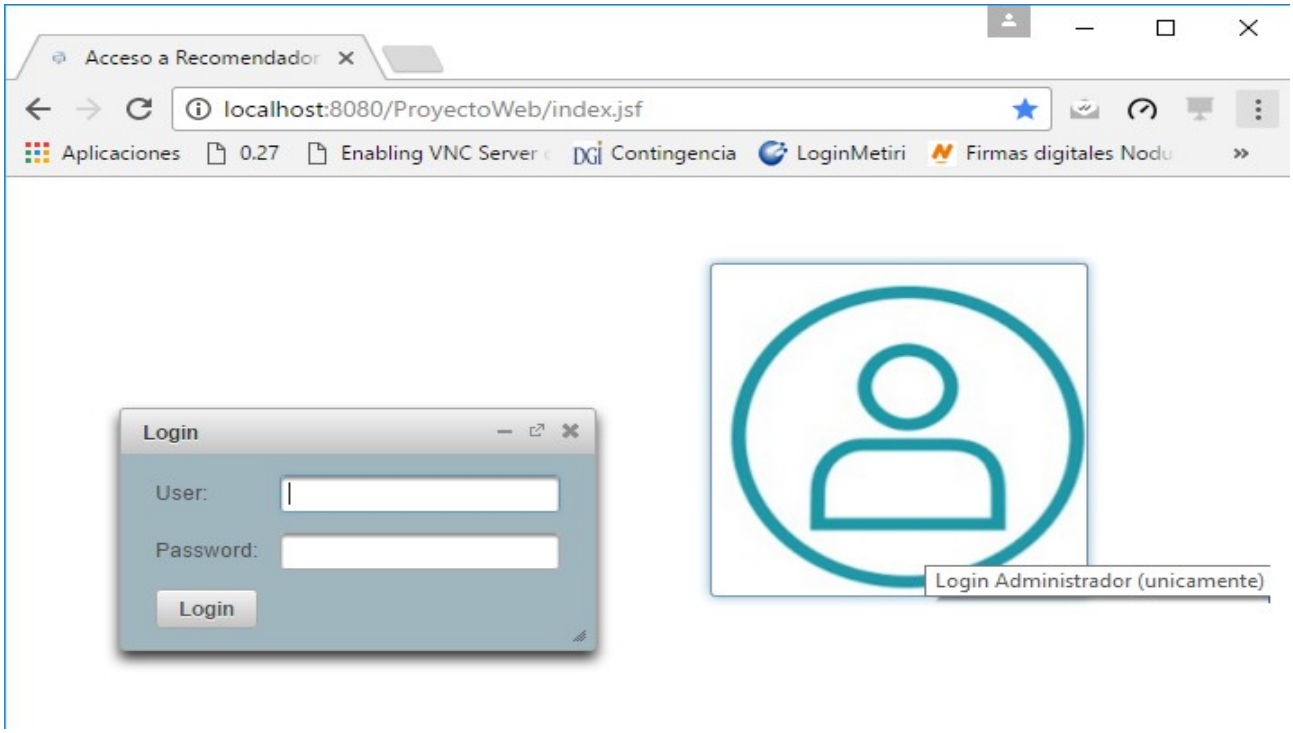


Figura 24 - Acceso y Login del Administrador del Recomendador

Para acceder al mismo, se deberá de introducir, usuario: admin y clave: admin. Posteriormente se podrá modificar esta contraseña.

Una vez accedido a este, se podrá ver un set de acciones. Las mismas son: Setear Ruta de Archivos, Setear Peso Atributos, Ingresar Recursos, Cambiar Contraseña y Salir. Estas pueden ser accedidos mediante un link a la izquierda de la página principal, como también mediante botones en el centro de la página.

Figura 25 - Página de inicio de Administrador

A continuación se detalla una breve descripción de las acciones en cuestión:

- Setear Ruta de Archivos: Permite editar las rutas de los archivos vitales del recomendador. Estos son grabados en la tabla ruta_archivos de la base de datos del recomendador, el cual tiene para cada uno de estos, la ubicación del mismo. Los mismos son: Área Temática, Red Social, Relaciones y el xml de la Red Social.
- Setear Peso de Atributos: Permite editar los pesos de los atributos relevantes para la red social en cuestión.
- Ingreso Recursos: Permite Ingresar masivamente los recursos del recomendador dado un archivo xml.
- Cambiar Contraseña: Permite modificar al usuario administrador la contraseña de acceso al sistema.
- Salir: Sale del Administrador del Recomendador.

Tal como el Recomendador, el diseño y navegación del sistema, no se hace a través de múltiples páginas web, sino que se desarrolló sobre una única página “config_admin.xhtml”. Los distintos componentes de parametrización e inserción de resultados, se implementaron mediante el componente de Primefaces p:dialog. Este me permite superponer otros elementos en la página principal permitiéndonos dejarla siempre visible.

A continuación un breve resumen de la estructura de la página:

config_admin.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui">

<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>Administrador</title>
        <style type="text/css">
                .notShow {
                        display:none;
                }

                body {
                        font: 100%/1.4 Verdana, Arial, Helvetica, sans-serif;
                        background: #42413C;
                        margin: 0;
                        padding: 0;
                        color: #000;
                }
…
…
…
                .button1 {
                        height:200px; width: 200px;          background-image: url("../images/archivos.jpg");
        border-style:solid; border-width: 1px;
                        display:block!important;
                        -webkit-transition:-webkit-transform 0.5s ease-out;
                        -moz-transition:-moz-transform 0.5s ease-out;
                        -o-transition:-o-transform 0.5s ease-out;
                        -ms-transition:-ms-transform 0.5s ease-out;
                        transition:transform 0.5s ease-out;
                }
                .button1:HOVER {
                        -moz-transform: scale(1.05);
                        -webkit-transform: scale(1.05);
                        -o-transform: scale(1.05);
                        -ms-transform: scale(1.05);
                        transform: scale(1.05)
                }
        </style>
</h:head>
```

```
<body>
<div class="container">
  <div class="header"><img src="../images/imagen_top.jpg" width="961" height="145"/><!-- end .header --></div>
  <div class="sidebar1">
    <h:form id="idForm1">
    <ul class="nav">
      <li>

                        <p:commandLink value="Setear Ruta Archivos" ...../>
      </li>
      <li>
                <p:commandLink value="Setear Atributos y Peso"..../>
      </li>
      <li>
                <p:commandLink value="Insertar Recursos" ...../>
      </li>
              <li>
                        <p:commandLink value="SALIR" action="#{usrAdmin.exit}"...../>
              </li>
    </ul>
    </h:form>
  </div>
  <div class="content">
    <table width="750" border="0">
      <tr>
        <td align="center">
                            <p:commandLink action="#{adminContextMenuView.init}"...>
                        <p:commandButton...title="Setear Ruta Archivos: Setea la ruta de los archivos pertinentes a la
aplicación"/>
                    </p:commandLink>
        </td>
        <td align="center">
                            <p:commandLink action="#{adminContextMenuView.init}"...>
                        <p:commandButton...title="Setear Atributos y Peso: Permite modificar los atributos relevantes según
la red y su respectivo peso"/>
                    </p:commandLink>
        </td>
        <td align="center">
                            <h:outputLink onclick="PF('dlgIR').show();">
                        <p:commandButton...title="Insertar Recursos: Permite insertar recursos a través de un xml
predefinido"/>
                            </h:outputLink>
        </td>
      </tr>
      <tr>
        <td align="center">
                            <h:outputLink onclick="PF('dlgCC').show();">
                        <p:commandButton...title="Cambiar Contrasena"/>
                            </h:outputLink>
        </td>
        <td align="center">
                            <p:commandLink action="#{usrAdmin.exit}">
                        <p:commandButton... title="Salir"/>
            </p:commandLink>
        </td>
        <td> </td>
```

```
    </tr>
   </table>
  </div>
  <div class="footer">
   <p></p>
  </div>
</div>

<h:form id="dlgForm">

   <p:dialog header="Rutas Archivos" id="dlgRA" widgetVar="dlgRA" minimizable="true" maximizable="true"
styleClass="archivo">
          <p:ajax event="close" listener="#{adminContextMenuView.busquedaExit}"
oncomplete="handleHRutaArchivosOut()" update="dlgRA"/>
…
…
…
   </p:dialog>


   <p:dialog header="Peso Atributos" id="dlgPA" widgetVar="dlgPA" minimizable="true" maximizable="true"
styleClass="archivo">
          <p:ajax event="close" listener="#{adminContextMenuView.busquedaExit}"
oncomplete="handleHPesoAtributosOut()" update="dlgPA"/>

             <table width="600" border="0" align="center" >
…
…
…
       </table>
   </p:dialog>


   <p:dialog header="Insertar Recursos" id="dlgIR" widgetVar="dlgIR" minimizable="true" maximizable="true"
styleClass="archivo">
          <p:ajax event="close" listener="#{adminContextMenuView.busquedaExit}"
oncomplete="handleHInsertarRecursosOut()" update="dlgIR"/>

             <table width="600" border="0" align="center" >
…
…
…

             </table>
   </p:dialog>
```

3.5.3.1 Setear Ruta de Archivos

El administrador podrá setear las rutas de los archivos relevantes para la aplicación.

Figura 26 - Dialogo para setear rutas de los archivos

Aquí se podrá cambiar las rutas de los archivos en cuestión. Para obtener los valores actuales, se realiza bajo un commandlink, llamando a la acción action del bean que atiende la página, AdminContextMenuView.

```
<p:commandLink action="#{adminContextMenuView.init}" >
        <p:commandButton id="cbRutaArch"  title="Setear Ruta Archivos: Setea la ruta de los archivos pertinentes a la aplicacion"/>
</p:commandLink>
```

```
@ManagedBean(name="adminContextMenuView")
@SessionScoped
public class AdminContextMenuView implements Serializable {

   @ManagedProperty(value="#{recomendadorBean}")
   private Recomendador rService;

   private String areaTematica;
   private String redSocial;
   private String relaciones;
   private String xmlValRedSocial;
   private String xmlValRecursos;
```

El mismo tiene instanciado el bean Recomendador, el cual es usado para obtener los datos en cuestión (en este caso las rutas de los archivos).
Los campos a usar son atributos del bean, el cual alojan los valores de los 5 archivos, antes de editarlos (configuración predeterminada). Para la obtención de los mismos, es usado un DataType AdminContent, el cual tiene los 5 valores de las rutas de los

archivos. Una vez obtenido, se los setea en los atributos del bean.

```
public void obtenerDatosArchivos() {
        try {
                AdminContent ac = rService.getDatosArchivos();
                areaTematica = ac.getAreaTematica();
                redSocial = ac.getRedSocial();
                relaciones = ac.getRelaciones();
                xmlValRedSocial = ac.getXmlValRedSocial();
                xmlValRecursos = ac.getXmlValRecursos();
                System.out.println("ADM"+areaTematica+","+redSocial+","+relaciones+","+xmlRedSocial);
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

En cambio, para la actualización de las rutas, no es necesario usar el DataType AdminContent, dado que solamente se pasaran como parámetro, las 5 rutas. Y este siempre actualizará las 5. Aquí hay un pequeño control de que las 5 campos estén completos, ya que si no, no deja grabar el formulario. En caso contrario, se despliega un error por un mensaje de la página.

```
public void grabarDatosArchivos() {
        try {
                RequestContext context = RequestContext.getCurrentInstance();
                FacesMessage message = null;
        boolean loggedIn = false;

        if(xmlRecursos != null && xmlRedSocial != null && areaTematica != null && redSocial != null &&
relaciones != null) {
          loggedIn = true;
          message = new FacesMessage(FacesMessage.SEVERITY_INFO, "Datos Archivos", "Realizada
Correctamente");
        } else {
          loggedIn = false;
          message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Datos Archivos", "Debe de
completar todos los campos");
        }

                if (!loggedIn)
                        FacesContext.getCurrentInstance().addMessage(null, message);

                if (loggedIn) {
                        rService.setDatosArchivos(areaTematica, redSocial, relaciones, xmlRedSocial);
                        FacesContext.getCurrentInstance().addMessage(null, message);
                }
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

3.5.3.2 Setear Peso de Atributos

El administrador podrá setear el peso de los atributos de la/s red/es en la que trabaja.

Figura 27 - Dialogo para setear peso de los atributos

Aquí se podrá cambiar el peso de los atributos, teniendo la única consideración, de que la suma de los pesos, sea igual a 1. El llamado de los valores actuales, se realiza bajo un commandlink, llamando a la acción action del bean que atiende la página, AdminContextMenuView (el mismo que para la opción Setear Rutas Archivos).

```
<p:commandLink action="#{adminContextMenuView.init}" >
        <p:commandButton id="cbRutaArch"  title="Setear Atributos y Peso: Permite modificar los atributos
relevantes según la red y su respectivo peso"/>
</p:commandLink>
```

```
@ManagedBean(name="adminContextMenuView")
@SessionScoped
public class AdminContextMenuView implements Serializable {

   @ManagedProperty(value="#{recomendadorBean}")
   private Recomendador rService;

        private String nAtr1;
        private String nAtr2;
        private String nAtr3;
        private String nAtr4;

        private double vAtr1;
        private double vAtr2;
        private double vAtr3;
        private double vAtr4;
```

El mismo tiene instanciado el bean Recomendador, el cual es usado para obtener los datos en cuestión (en este caso los nombres y pesos de los atributos de la Red Social). Los campos a usar son atributos del bean, el cual alojan los valores de los (hasta) 4 atributos como los (hasta) 4 pesos respectivamente. Para la obtención de los mismos, es usado un DataType AdminContent, el cual tiene los 4 valores de los atributos y pesos. Una vez obtenido, se los setea en los atributos del bean. Aquí también esta la

opción de que cada vez que se cambia el contexto de la red social, se cambian los atributos automáticamente.

```
public void obtenerDatosAtributos(String contexto) {
        try {
                AdminContent ac = rService.getDatosAtributos(contexto);
                nAtr1=ac.getnAtr1();
                nAtr2=ac.getnAtr2();
                nAtr3=ac.getnAtr3();
                nAtr4=ac.getnAtr4();
                vAtr1=ac.getvAtr1();
                vAtr2=ac.getvAtr2();
                vAtr3=ac.getvAtr3();
                vAtr4=ac.getvAtr4();

        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

En cambio, para la actualización de las atributos, no es necesario usar el DataType AdminContent, dado que solamente se pasaran como parámetro, los 4 atributos/pesos. Y como el caso anterior, siempre actualizará los 4. Aquí hay pequeños controles, como el ser que los 4 campos tengan valores, y también que la suma de los mismos sea igual a 1. En caso afirmativo, procederá a la actualización de los mimos, en caso negativo, saldrá mensaje de error.

```
public void grabarDatosAtributos(String contexto) {
        try {
                RequestContext context = RequestContext.getCurrentInstance();
                FacesMessage message = null;
    boolean loggedIn = false;

    if((vAtr1+vAtr2+vAtr3+vAtr4)==1) {
                        if((vAtr1+vAtr2+vAtr3+vAtr4)==1) {
                    loggedIn = true;
                message = new FacesMessage(FacesMessage.SEVERITY_INFO, "Peso Atributos",
"Realizada Correctamente");
                } else {
                  loggedIn = false;
                  message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Peso Atributos",
"La suma de los pesos de los atributos debe de ser igual a 1");
                }
        } else {
           loggedIn = false;
           message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Peso Atributos", "La suma
de los pesos de los atributos debe de ser igual a 1");
        }

                if (!loggedIn)
                        FacesContext.getCurrentInstance().addMessage(null, message);

                if (loggedIn) {
```

```
                                Double v1=new Double(vAtr1), v2=new Double(vAtr2), v3=new Double(vAtr3),
v4=new Double(vAtr4);
                                rService.setDatosAtributos(contexto, nAtr1, v1, nAtr2, v2, nAtr3, v3, nAtr4, v4);
                                FacesContext.getCurrentInstance().addMessage(null, message);
                                //context.execute("dlgCC.hide()");
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

3.5.3.3 Ingresar recursos masivamente

El administrador podrá ingresar masivamente recursos.

Figura 28 - Dialogo para ingresar Masivamente Recursos

Aquí se podrá elegir el archivo xml con la información de los recursos que se necesiten ingresar masivamente, para que posteriormente se ingresen a la base de datos. El llamado a la grabación, se hace con el botón de Guardar, el cual se realiza bajo un commandlink, llamando a la acción action del bean que atiende la página, AdminContextMenuView (el mismo que las opciones anteriores).

```
<p:commandLink action="#{adminContextMenuView.grabarDatosArchivos()}">
        <p:commandButton id="datosIRecursosAceptar" value="Guardar" icon="ui-icon-search" />
</p:commandLink>
```

```
@ManagedBean(name="adminContextMenuView")
@SessionScoped
public class AdminContextMenuView implements Serializable {

   @ManagedProperty(value="#{recomendadorBean}")
   private Recomendador rService;

        private String rutaRecursos;
```

El mismo tiene instanciado el bean Recomendador, el cual es usado para ejecutar y grabar la información de los recursos dados en cuestión (aquí solo necesitamos la ruta del archivo, todo el resto lo hace el bean Recomendador, pero para esto, como el comando usado de primefaces es un fileUpload, debemos de grabar el archivo en

alguna carpeta del servidor, para luego poder tomar la ruta del mismo).
Los campos a usar son atributos del bean, el cual aloja la ruta del archivo xml de los recursos.
Para la inserción de la información de los recursos, hay pequeños controles tales como que el archivo no sea null. En caso afirmativo, se graba el archivo subido al servidor, y se pasa la ruta para poder grabar los recursos, y en caso negativo, saltará el error correspondiente.

```
	public void insertarRecursos() {
		try {
			RequestContext context = RequestContext.getCurrentInstance();
			FacesMessage message = null;
			boolean loggedIn = false;

			if(file != null) {

				String fileName  = file.getFileName();
				ServletContext servletContext = (ServletContext)
FacesContext.getCurrentInstance().getExternalContext().getContext();
				rutaRecursos = servletContext.getRealPath("") + File.separator + "upload" +
File.separator+ fileName;

				File destFile = new File(rutaRecursos);
				FileUtils.copyInputStreamToFile(file.getInputstream(), destFile);


			if(rutaRecursos != null) {
			   loggedIn = true;
			   message = new FacesMessage(FacesMessage.SEVERITY_INFO, "Insertar Recursos",
"Realizada Correctamente");
			} else {
			   loggedIn = false;
			   message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Insertar
Recursos", "Debe de completar los campos");
			}

				if (!loggedIn)
					FacesContext.getCurrentInstance().addMessage(null, message);

				if (loggedIn) {
					rService.addRecursosDesdeArchivo(rutaRecursos, xmlRecursos, contexto);
					FacesContext.getCurrentInstance().addMessage(null, message);
					//context.execute("dlgCC.hide()");
				}
			}
			else {
		loggedIn = false;
		message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Insertar Recursos", "No
fue seleccionado ningun archivo");

		if (!loggedIn)
		  FacesContext.getCurrentInstance().addMessage(null, message);
```

```
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

3.5.3.4 Cambiar Contraseña

El administrador podrá modificar la contraseña de acceso al sistema.

Figura 29 - Dialogo para modificar la contraseña del administrador

Aquí se podrá modificar la contraseña del administrador, pero para esto, se deberá de ingresar la contraseña vieja, y la nueva, repitiendo esta 2 veces para control. El llamado a la grabación, se hace con el botón de Guardar, el cual se realiza bajo un commandlink, llamando a la acción action del bean que atiende la página, AdminContextMenuView (el mismo que las opciones anteriores).

```
<p:commandLink action="#{adminContextMenuView.guardarContrasena()}">
        <p:commandButton id="datosCCAceptar" value="Guardar" icon="ui-icon-search" />
</p:commandLink>
```

```
@ManagedBean(name="adminContextMenuView")
@SessionScoped
public class AdminContextMenuView implements Serializable {

   @ManagedProperty(value="#{recomendadorBean}")
   private Recomendador rService;

   private String pwdOld;
   private String pwdNew;
   private String pwdNewR;
```

El mismo tiene instanciado el bean Recomendador, el cual es usado para ejecutar y guardar la nueva contraseña del administrador. Para esto, se hace un control previo de que la contraseña vieja coincida con al ingresada, como también las contraseñas nuevas coincidan. En caso que alguna de estos anteriores no sean correctas, se

desplegará un mensaje de error, indicando el problema. En caso afirmativo, se actualizará la pwd del administrador.

```
public void guardarContrasena() {
try {
RequestContext context = RequestContext.getCurrentInstance();
FacesMessage message = null;
boolean loggedIn = false;


if(pwdOld != null && pwdNew != null && pwdNewR != null) {
if(pwdOld.equals(rService.getPwdAdmin())) {
if(pwdNew.equals(pwdNewR)) {
if(pwdNew.length() <= 30) {
loggedIn = true;
message = new FacesMessage(FacesMessage.SEVERITY_INFO, "Cambio Contrasena", "Realizada Correctamente");
} else {
loggedIn = false;
message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Cambio Contrasena", "La nueva contrasena supera los 30 caracteres, ingresela nuevamente");
}
} else {
loggedIn = false;
message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Cambio Contrasena", "Las contrasenas nuevas no coinciden");
}
} else {
loggedIn = false;
message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Cambio Contrasena", "La actual contrasena no coincide con la registrada");
}
} else {
loggedIn = false;
message = new FacesMessage(FacesMessage.SEVERITY_WARN, "Error - Cambio Contrasena", "Debe de completar todos los campos");
}

if (!loggedIn)
FacesContext.getCurrentInstance().addMessage(null, message);

if (loggedIn) {
rService.setPwdAdmin(pwdNew);
FacesContext.getCurrentInstance().addMessage(null, message);
//context.execute("dlgCC.hide()");
}
} catch (Exception e) {
e.printStackTrace();
}
}
```

3.5.3.5 Salir

Será utilizada para que el usuario administrador salga del Sistema Recomendador.

4 – Casos de estudio

A continuación se describen los casos de prueba generados para testear el Sistema Recomendador con Friendsourcing Semántico al brindar recomendaciones a usuarios, también se explica el escenario en que fueron ejecutados los casos de prueba, qué resultados se obtuvieron y cómo se interpretan dichos resultados. Con estas pruebas se busca mostrar el esperado y correcto funcionamiento del algoritmo.

A lo largo de esta sección, se hará mención al archivo (hoja de datos) Datos.ods, donde se fueron registrando todos los cálculos realizados a lo largo del testing sobre el Sistema Recomendador.

4.1 Datos utilizados en las pruebas

Para la realización de las distintas pruebas se utilizaron los siguientes datos, los mismos se encuentran disponibles en las hojas “RelacionesEntreUsuarios”, “Recursos” y “RecursosEvaluados” del archivo Datos.ods:

- Cantidad de usuarios en la red social: 15
- Relaciones y subrelaciones entre usuarios en la Red Social por subredes:
  - Amistad (12 usuarios):
    - Relaciones efectivas: 54
    - Relaciones posibles: 132
    - Cubrimiento: 40,91 %
  - Familiar (11 usuarios):
    - Relaciones efectivas: 48
    - Relaciones posibles: 110
    - Cubrimiento: 43,63 %
  - Académico (8 usuarios):
    - Relaciones efectivas: 46
    - Relaciones posibles: 56
    - Cubrimiento: 82,14 %
  - Laboral (8 usuarios):
    - Relaciones efectivas: 38

    - Relaciones posibles: 56
    - Cubrimiento: 67,86 %
- Relaciones y subrelaciones entre usuarios en la Red Social a nivel global tomando en cuenta los 15 usuarios que la compone:
    - Amistad (15 usuarios):
        - Relaciones efectivas: 54
        - Relaciones posibles: 210
        - Cubrimiento: 25,71 %
    - Familiar (15 usuarios):
        - Relaciones efectivas: 48
        - Relaciones posibles: 210
        - Cubrimiento: 22,86 %
    - Académico (15 usuarios):
        - Relaciones efectivas: 46
        - Relaciones posibles: 210
        - Cubrimiento: 21,90 %
    - Laboral (15 usuarios):
        - Relaciones efectivas: 38
        - Relaciones posibles: 210
        - Cubrimiento: 18,10 %
    - Todas (15 usuarios):
        - Relaciones efectivas: 186
        - Relaciones posibles: 840
        - Cubrimiento: 22,14 %
- Cantidad de recursos en la red: 107
    - Recursos evaluados: 62
    - Recursos sin evaluar: 45
- Número promedio de conexiones por usuario: 12,4 (no implica que sean todas conexiones con usuarios distintos, pueden haber mas de una conexión con un

mismo usuario, lo que no se permite es que hayan 2 o mas conexiones con un mismo usuario a través de una relación dentro de un mismo grupo de relaciones).

- Número promedio de evaluaciones de recursos por usuario: 8
- Cantidad total de evaluaciones de recursos que hicieron los usuarios: 120
- Promedio de la cantidad de veces que fue evaluado un mismo recurso: 1,875
  - Máximo: 4
  - Mínimo: 1
- Las valuaciones de recursos se hicieron sobre los cuatro atributos de calidad:
  - trustwhorthy (confiable), peso: 0,25
  - objective (objetivo), peso: 0,25
  - complete (completo), peso: 0,25
  - well-written (bien escrito), peso: 0,25

Los IDs de recursos relevantes según la relación como preferencia son los siguientes:

Recursos relevantes para relación amigo:

Películas: 19 -> [1, 2, 4, 5, 7, 9, 11, 14, 15, 16, 17, 20, 22, 24, 25, 30, 31, 36, 39]

Libros: 13 -> [45, 47, 52, 53, 54, 56, 60, 64, 68, 69, 72, 81, 82]

Recursos relevantes para relación amiga:

Películas: 13 -> [2, 4, 7, 16, 17, 18, 20, 22, 25, 30, 31, 36, 39]

Libros: 12 -> [45, 47, 52, 53, 55, 56, 61, 68, 69, 72, 81, 82]

Recursos relevantes para relación Amistad:

Películas: 20 -> [1, 2, 4, 5, 7, 9, 11, 14, 15, 16, 17, 18, 20, 22, 24, 25, 30, 31, 36, 39]

Libros: 15 -> [45, 47, 52, 53, 54, 55, 56, 60, 61, 64, 68, 69, 72, 81, 82]

Recursos relevantes para relación Familiar:

Películas: 12 -> [2, 6, 12, 16, 18, 20, 21, 22, 25, 31, 38, 39]

Libros: 12 -> [43, 45, 47, 48, 52, 53, 55, 62, 68, 69, 72, 81]

Recursos relevantes para relación estudiante-curso:

Películas: 0 -> []

Libros: 13 -> [84, 86, 88, 90, 92, 93, 97, 99, 100, 102, 103, 105, 106]

Recursos relevantes para relación profesor-curso:

Películas: 0 -> []

Libros: 1 -> [96]

Recursos relevantes para relación Académico:

Películas: 0 -> []

Libros: 14 -> [84, 86, 88, 90, 92, 93, 96, 97, 99, 100, 102, 103, 105, 106]

LibroDeAprendizaje: 14 -> [84, 86, 88, 90, 92, 93, 96, 97, 99, 100, 102, 103, 105, 106]

Recursos relevantes para relación colegaDeTrabajo:

Películas: 0 -> []

Libros: 12 -> [45, 47, 52, 53, 55, 68, 69, 72, 75, 77, 78, 81]

Recursos relevantes para relación Laboral:

Películas: 0 -> []

Libros: 17 -> [43, 45, 47, 52, 53, 55, 63, 68, 69, 72, 73, 75, 77, 78, 80, 81, 82]

LibroDeTrabajo: 5 -> [73, 75, 77, 78, 80]

Recursos relevantes para toda Relación:

Películas: 24 -> [1, 2, 4, 5, 6 ,7 ,9, 11, 12, 14, 15, 16, 17, 18, 20, 21, 22, 24, 25, 30, 31, 36, 38, 39]

Libros: 38 -> [43, 45, 47, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63, 64, 68, 69, 72, 73, 75, 77, 78, 80, 81, 82, 84, 86, 88, 90, 92, 93, 96, 97, 99, 100, 102, 103, 105, 106]

Para la Red Social se utiliza el archivo RedSocial.xml. La Red Social simulada se compone de 15 usuarios y de las relaciones que se dan entre éstos. Se puede ver a la Red Social como la unión de subredes, como son Amistad, Familiar, Académica y Laboral, las cuales están integradas por algunos de los 15 integrantes de la Red Social global. Un usuario puede conectarse a un mismo usuario hasta cuatro veces, pero se impone que si entre dos usuarios existen dos o mas relaciones entre ellos, las relaciones deben pertenecer a distintos ámbitos o grupos relacionales (Amistad, Familiar, Académico y/o Laboral). Es decir, que entre dos usuarios no puede haber por ejemplo una relación de “amigo” y de “mejorAmigo”, ya que pertenecen al mismo grupo relacional.

Para los recursos se utiliza el archivo Recursos.xml. En el mismo se incluyen Libros y Películas. Los recursos fueron obtenidos de dos fuentes de información mundialmente conocidas, para los libros los recursos fueron extraídos de WorldCat[44] y para las películas los recursos fueron extraídos de IMDb[45].

Para los atributos se utiliza el archivo Atributos.xml

Para los recursos evaluados se ejecuta el archivo cargaEvaluaciones.sql, directamente sobre la base de datos.

A continuación se muestran las subredes sociales (Amistad, Familiar, Académica y Laboral, en ese orden) incluidas en la red social:

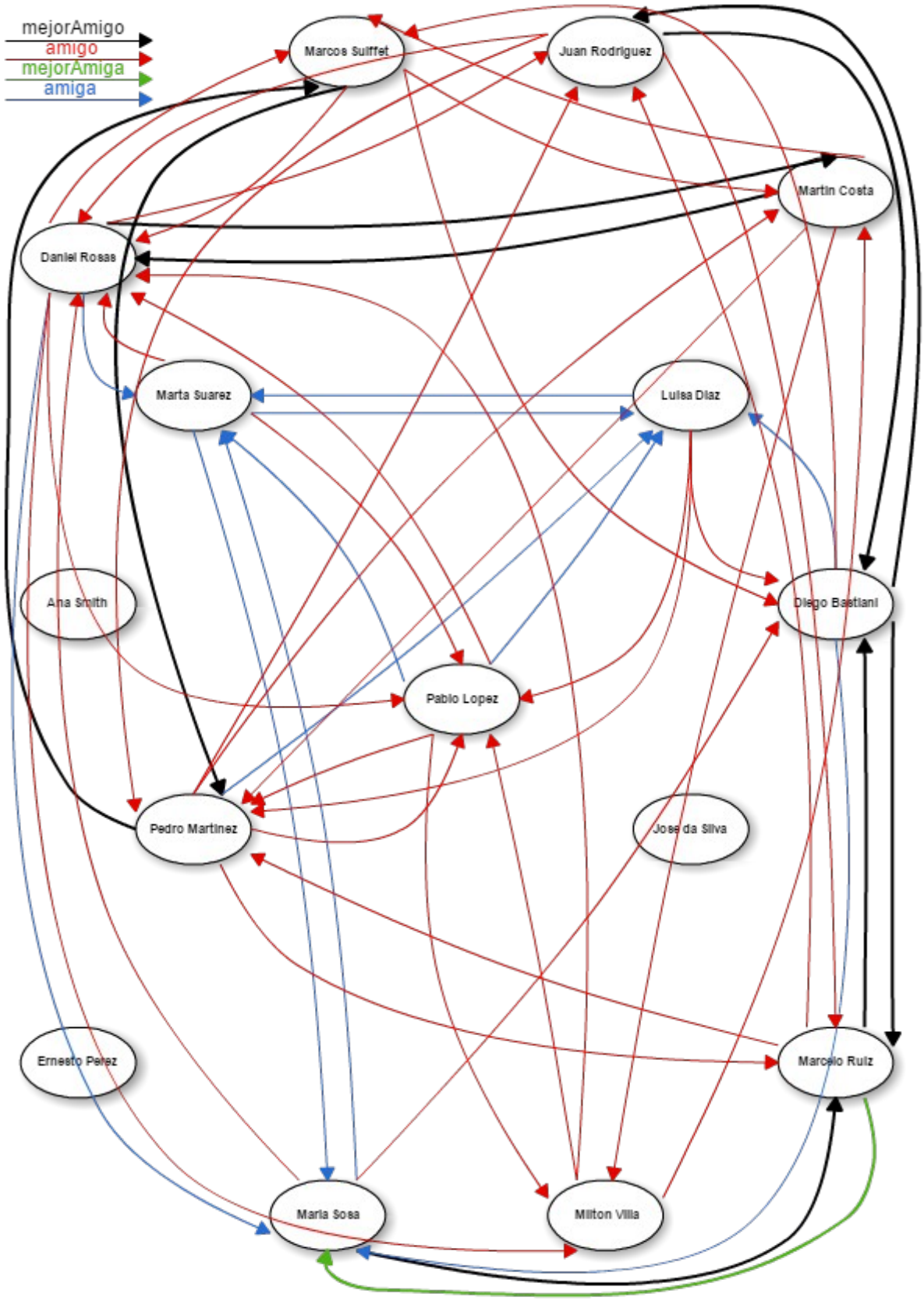


Figura 30 - Relaciones de Amistad en la Red Social Simulada

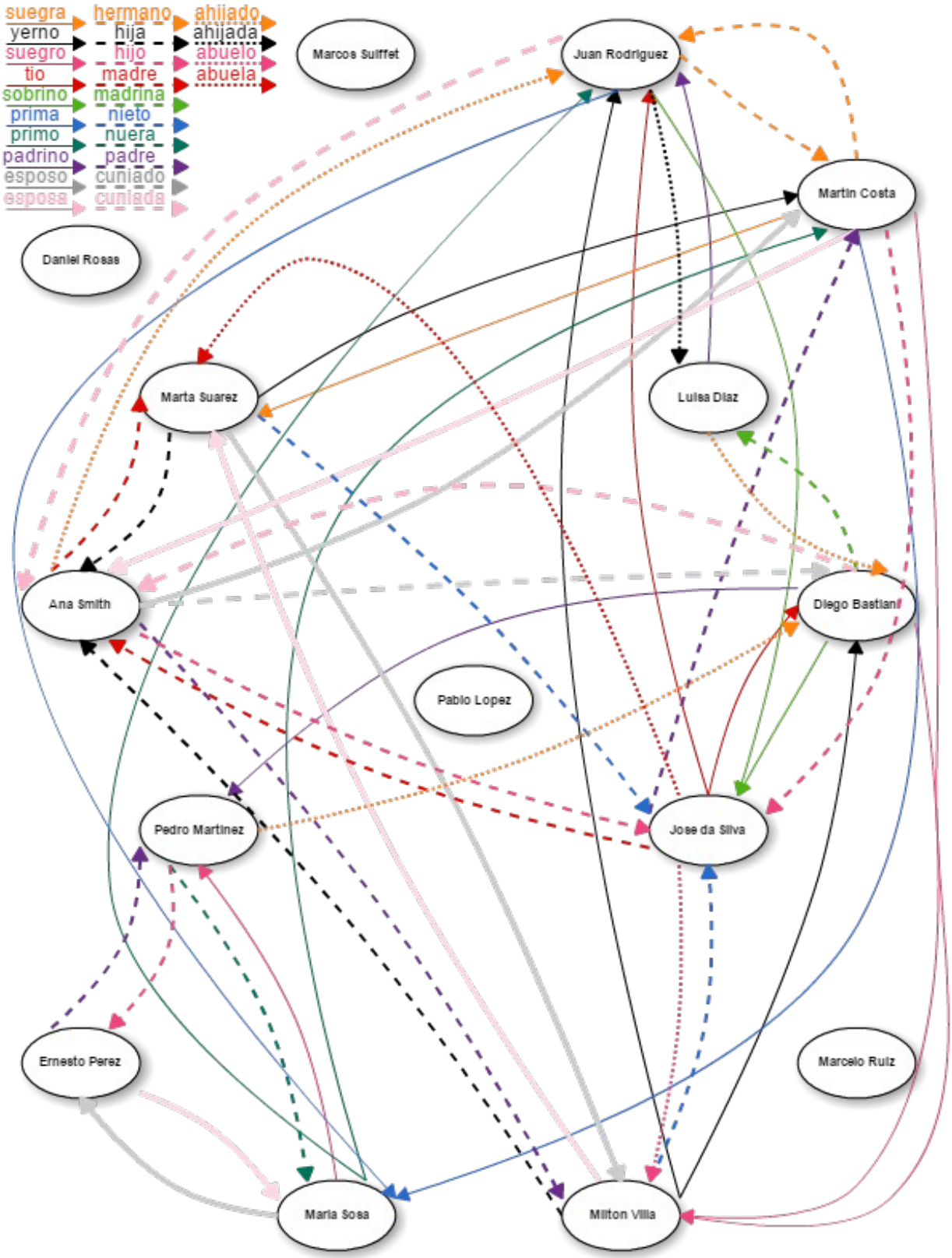


Figura 31 - Relaciones Familiares en la Red Social Simulada

Figura 32 - Relaciones Académicas en la Red Social Simulada

Figura 33 - Relaciones Laborales en la Red Social Simulada

4.2 Métodos de evaluación

Se plantea la necesidad de probar qué tan acertados son los resultados obtenidos en nuestro Sistema Recomendador con Friendsourcing Semántico. Para ello, existen diversas técnicas y métricas [46][47], que ayudan a medir que tan eficiente es nuestro algoritmo de recomendación. Entre ellas, las que utilizamos son las siguientes:

4.2.1 Precisión (Precision) y Exhaustividad (Recall)

Para evaluar los sistemas de recomendación, también se suelen utilizar métricas de recuperación de información, más precisamente precisión y exhaustividad. Precisión(precision) es la proporción de recomendaciones principales que son buenas recomendaciones, y exhaustividad(recall) es la proporción de buenas recomendaciones que aparecen en las recomendaciones principales.

$$\text{Precisión} = RR/Rd$$

$$\text{Exhaustividad} = RR/Rt$$

Donde:

- **RR** es el total de ítems relevantes recuperados.
- **Rd** es el total de ítems recuperados.
- **Rt** es el total de ítems relevantes.

4.2.2 Error Cuadrático Medio o RMSE

Es una técnica simple pero robusta para determinar la exactitud en las recomendaciones. En nuestro caso se utiliza para medir el error cuadrático medio de la Precisión(Precision) y de la Exhaustividad(Recall).

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (\hat{r}_{ui} - r_{ui})^2}$$

Donde:

- El sistema genera recomendaciones $\hat{\mathbf{r}}_{\mathbf{ui}}$ para un set de datos $\boldsymbol{T}$ de pares usuario-ítem ***(u,i)*** para el cual la recomendación real $\mathbf{r}_{\mathbf{ui}}$ es conocida.

4.2.3 F-Measure

Es la media armónica de la precisión(precision) y de la exhaustividad(recall). Es una única medida que combina precisión vs exhaustividad.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \qquad ,\beta^2 = \frac{1-\alpha}{\alpha}$$

Tomando:

$$\alpha = 1/2 \qquad \beta = 1$$

Resulta la fórmula de F balanceada para que los pesos tanto de Precision como de Recall sean equilibradas:

$$F_{\beta=1} = \frac{2PR}{P+R}$$

4.2.4 Mean Reciprocal Rank (MRR)

Consideramos la posición en la lista de ítems recomendados, del primer ítem relevante.

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i}$$

Donde:

- **rank_i** es el ranking del primer ítem relevante en la i-ésima consulta y **Q** es la cantidad de consultas realizadas.

4.3 Casos de Prueba

Se generaron 50 casos de prueba, los mismos utilizados para solicitar recomendaciones al sistema. Se buscaron para los mismos, combinaciones entre relaciones en la red social, distancia entre usuarios (1 o 2), áreas temáticas, y relaciones preferentes en el recurso. Los casos de exigencia mayor corresponden a los IDs de casos de prueba 33 a 36, ya que se aplica sobre todas las relaciones del usuarios, y aún mas exigentes los IDs de casos de prueba 37 y 38, que además de aplicarlos sobre todas las relaciones del usuario, busca recursos sobre todas las áreas temáticas posibles. Los IDS de casos de prueba 39 a 50, fueron generados para mostrar cómo influye la relación preferente en los recursos, cuando el sistema genera la puntuación de los recursos recomendados.
Se eligieron 4 usuarios al azar que ejecuten los IDs de casos de prueba 1 a 38 (siempre que sea posible, es decir, que si no tienen una relación de Amistad en la red social el usuario, no va a solicitarle al Sistema Recomendador, recomendaciones sobre dicha relación(Amistad), en casos como este, se descarta el caso de prueba para el usuario), y dos usuarios para ejecutar los IDs de casos de pruebas 39 a 50, donde uno ejecuta los IDs de casos de prueba 39 a 41 y 45 a 47, y el otro usuario ejecuta los IDs de casos de prueba 42 a 44 y 48 a 50 (lo explicado anteriormente también es válido aquí).

Para las relaciones dentro de la red social fueron seleccionadas: amigo, Amistad, Familiar, estudiante-curso, Académico, colegaDeTrabajo, Laboral y Todas (todas las relaciones).

Para las áreas temáticas fueron seleccionadas: Película, Libro, LibroDeTrabajo, LibroDeAprendizaje y Todas (todas las áreas temáticas).

Para las distancias fueron seleccionados: 1 y 2.

Para las relaciones preferibles en los recursos fueron seleccionados: amigo, amiga, Amistad, Familiar, estudiante-curso, profesor-curso, Académico, colegaDeTrabajo, Laboral y Todas (toda relación).

A continuación se muestran en detalle los casos de prueba, obtenidos de la hoja “Casos de Prueba” del archivo Datos.ods:

ID Caso de Prueba	Relación en la red	Área Temática	Distancia	Relación en el recurso
1	amigo	Pelicula	1	amigo
2	amigo	Pelicula	2	amigo
3	amigo	Libro	1	amigo
4	amigo	Libro	2	amigo
5	Amistad	Pelicula	1	Amistad
6	Amistad	Pelicula	2	Amistad
7	Amistad	Libro	1	Amistad
8	Amistad	Libro	2	Amistad
9	Familiar	Pelicula	1	Familiar
10	Familiar	Pelicula	2	Familiar
11	Familiar	Libro	1	Familiar
12	Familiar	Libro	2	Familiar
13	estudiante-curso	Pelicula	1	estudiante-curso
14	estudiante-curso	Pelicula	2	estudiante-curso
15	estudiante-curso	Libro	1	estudiante-curso
16	estudiante-curso	Libro	2	estudiante-curso
17	Academico	Pelicula	1	Academico
18	Academico	Pelicula	2	Academico
19	Academico	Libro	1	Academico
20	Academico	Libro	2	Academico
21	Academico	LibroDeAprendizaje	1	Academico
22	Academico	LibroDeAprendizaje	2	Academico
23	colegaDeTrabajo	Pelicula	1	colegaDeTrabajo
24	colegaDeTrabajo	Pelicula	2	colegaDeTrabajo
25	colegaDeTrabajo	Libro	1	colegaDeTrabajo
26	colegaDeTrabajo	Libro	2	colegaDeTrabajo
27	Laboral	Pelicula	1	Laboral
28	Laboral	Pelicula	2	Laboral
29	Laboral	Libro	1	Laboral
30	Laboral	Libro	2	Laboral
31	Laboral	LibroDeTrabajo	1	Laboral
32	Laboral	LibroDeTrabajo	2	Laboral
33	Todas	Pelicula	1	Todas
34	Todas	Pelicula	2	Todas
35	Todas	Libro	1	Todas
36	Todas	Libro	2	Todas
37	Todas	Todas	1	Todas
38	Todas	Todas	2	Todas
39	Academico	Libro	1	Academico
40	Academico	Libro	1	estudiante-curso
41	Academico	Libro	1	profesor-curso
42	Amistad	Pelicula	1	Amistad
43	Amistad	Pelicula	1	amigo
44	Amistad	Pelicula	1	amiga
45	Academico	Libro	2	Academico
46	Academico	Libro	2	estudiante-curso
47	Academico	Libro	2	profesor-curso
48	Amistad	Pelicula	2	Amistad
49	Amistad	Pelicula	2	amigo
50	Amistad	Pelicula	2	amiga

Figura 34 - Casos de Prueba

4.4 Resultado obtenidos

Sobre todas los casos de pruebas se controla el tiempo que el Sistema Recomendador demora en brindar las recomendaciones, así como en los casos que sea posible se calcula: Precision, Recall, F-Measure, MRR y RMSE.

Luego con los resultados anteriormente obtenidos, se calcula los promedios generales, los promedios generales para cuando las recomendaciones se brindan entre usuarios directamente relacionados (distancia 1), y entre usuarios directa e indirectamente relacionados (distancia 2).

Los usuarios seleccionados e IDs de casos de pruebas corridos fueron:

- Juan Rodriguez: 1-12, 23-38 (28 casos de prueba).
- Diego Bastiani: 1-22, 33-38 (28 casos de prueba).
- José da Silva: 9-38 (30 casos de prueba).
- Pablo López: 1-8, 13-38 (34 casos de prueba).
- Marcos Suiffet: 39-41, 45-47 (6 casos de prueba).
- Maria Sosa: 42-44, 48-50 (6 casos de prueba).

En la hoja “Ejecución Casos de Prueba” del archivo Datos.ods, se registran los casos de prueba corridos para cada uno de los usuarios anteriormente citados, donde el resultado son las recomendaciones de recursos brindados por el sistema, indicando la cantidad de recursos recomendados y la lista de IDs de recursos con sus respectivas puntuaciones de mayor a menor.

En la hoja “Cálculos Extraídos de las Ejecuciones” del archivo Datos.ods, se registran los casos de prueba corridos para cada uno de los usuarios anteriormente citados, pero calculando para cada uno de los casos de prueba de cada usuario lo siguiente:

- Tiempo de Ejecución (ms).
- Cantidad de Recursos Obtenidos en la Recomendación.
- Cantidad de Recursos Relevantes.
- Cantidad de Recursos Relevantes Obtenidos en la Recomendación.
- Precision (%).
- Recall (%).

- MRR (valor entre 0 y 1 inclusives).

En la hoja “Promedios Calculados” del archivo Datos.ods, se registran los promedios registrados por cada caso de prueba. En este se registran para cada caso de prueba lo siguiente:

- Cantidad de usuarios que ejecutaron el caso de prueba.
- Promedio de Tiempo de Ejecución (ms).
- Promedio de Precision (%).
- Promedio de Recall (%).
- Media Armónica (F-Measure) (%).
- RMSE de Precision (%).
- RMSE de Recall (%).
- Promedio MRR.

A continuación se muestran los resultados promedios obtenidos para cada caso de prueba:

ID	#Us	Tiempo (ms)	Pre (%)	Rec (%)	F-M (%)	RMSE (Pre)	RMSE (Rec)	MRR
1	3	3151,00	90,91	40,35	55,89	7,42	13,81	1,00
2	3	4768,00	84,95	73,68	78,92	4,88	15,49	1,00
3	3	2729,00	25,41	23,08	24,19	18,94	16,62	0,67
4	3	5057,67	31,36	46,15	37,34	3,77	16,62	0,71
5	3	3208,33	86,63	65,00	74,27	0,73	4,08	1,00
6	3	6900,67	85,09	95,00	89,77	1,83	4,08	1,00
7	3	3516,00	42,66	48,89	45,56	9,98	13,70	0,78
8	3	8321,33	42,86	86,67	57,35	2,07	5,44	0,78
9	3	4047,00	60,77	63,89	62,29	2,71	3,93	1,00
10	3	7364,67	57,04	91,67	70,32	2,88	0,00	1,00
11	3	4656,00	35,63	61,11	45,02	6,90	10,39	1,00
12	3	8302,00	33,43	86,11	48,17	1,82	3,93	1,00
13	3	2276,00	0,00	0,00	0,00	0,00	0,00	0,00
14	3	2896,00	0,00	0,00	0,00	0,00	0,00	0,00
15	3	3032,00	53,33	56,41	54,83	4,71	9,59	1,00
16	3	4151,00	50,00	74,36	59,79	0,00	7,25	1,00
17	3	2786,33	0,00	0,00	0,00	0,00	0,00	0,00
18	3	4052,00	0,00	0,00	0,00	0,00	0,00	0,00
19	3	3625,00	58,79	71,43	64,50	7,94	10,10	1,00
20	3	5213,67	55,69	92,86	69,62	0,84	5,83	1,00
21	3	3770,67	100,00	73,81	84,93	0,00	8,91	1,00
22	3	5453,33	100,00	92,86	96,30	0,00	5,83	1,00
23	3	2131,33	0,00	0,00	0,00	0,00	0,00	0,00
24	3	2474,00	0,00	0,00	0,00	0,00	0,00	0,00
25	3	2915,33	39,26	55,56	46,01	0,52	3,93	1,00
26	3	3500,00	39,26	55,56	46,01	0,52	3,93	1,00
27	3	3192,67	0,00	0,00	0,00	0,00	0,00	0,00
28	3	4193,00	0,00	0,00	0,00	0,00	0,00	0,00
29	3	4479,67	59,49	74,51	66,16	3,15	11,09	1,00
30	3	6010,00	54,47	84,31	66,18	1,09	2,77	1,00
31	3	1323,00	100,00	86,67	92,86	0,00	18,86	1,00
32	3	2573,00	100,00	100,00	100,00	0,00	0,00	1,00
33	4	10839,75	100,00	95,83	97,87	0,00	4,17	1,00
34	4	14707,25	100,00	97,92	98,95	0,00	3,61	1,00
35	4	12414,00	100,00	90,13	94,81	0,00	5,99	1,00
36	4	17238,25	100,00	96,05	97,99	0,00	2,94	1,00
37	4	45042,00	100,00	92,34	96,02	0,00	4,75	1,00
38	4	53587,25	100,00	96,77	98,36	0,00	1,98	1,00
39	1	6109,00	56,00	100,00	71,79	0,00	0,00	1,00
40	1	5953,00	52,00	100,00	68,42	0,00	0,00	1,00
41	1	5969,00	4,00	100,00	7,69	0,00	0,00	0,05
42	1	2609,00	90,91	50,00	64,52	0,00	0,00	1,00
43	1	2323,00	81,82	47,37	60,00	0,00	0,00	1,00
44	1	2397,00	54,55	46,15	50,00	0,00	0,00	1,00
45	1	6500,00	56,00	100,00	71,79	0,00	0,00	1,00
46	1	6134,00	52,00	100,00	68,42	0,00	0,00	1,00
47	1	6191,00	4,00	100,00	7,69	0,00	0,00	0,05
48	1	6945,00	85,71	90,00	87,80	0,00	0,00	1,00
49	1	6602,00	80,95	89,47	85,00	0,00	0,00	1,00
50	1	6313,00	61,90	100,00	76,47	0,00	0,00	1,00

Figura 35 - Resultados Promedios obtenidos para cada Caso de Prueba

Para los IDs de casos de prueba 13, 14, 17, 18, 23, 24, 26 y 27, no fue posible hallar Precision, Recall, RMSE ni MRR dado que para dichos casos de prueba no habían recursos relevantes.
Para los IDs de casos de prueba 39 a 50, no fue posible hallar MRR dado que esos casos de prueba se ejecutaron una sola vez.

Ahora mostramos los resultados generales obtenidos del Sistema Recomendador:

Promedios generales obtenidos en las pruebas	
Promedio Tiempo de Ejecución (ms)	6998,86
Promedio de Precision (%)	65,88
Promedio de Recall (%)	78,38
Media Armónica (F-Measure) (%)	71,59
RMSE de Precision (%)	2,76
RMSE de Recall (%)	7,32
Promedio MRR	0,93

Figura 36 - Resultados Generales obtenidos del Sistema Recomendador

Ahora mostramos los resultados generales obtenidos del Sistema Recomendador cuando las recomendaciones se realizan solamente sobre usuarios directamente conectados(distancia 1), y sobre usuarios conectados directamente e indirectamente (distancia 2):

Promedios generales obtenidos en las pruebas para recomendaciones con:	Usuarios relacionados directamente (distancia 1)	Usuarios relacionados directa e indirectamente (distancia 2)
Promedio Tiempo de Ejecución (ms)	5779,80	8217,92
Promedio de Precision (%)	66,29	65,46
Promedio de Recall (%)	68,69	88,07
Media Armónica (F-Measure) (%)	67,47	75,10
RMSE de Precision (%)	4,20	1,31
RMSE de Recall (%)	9,33	5,31
Promedio MRR	0,93	0,93

Figura 37 - Resultados Generales obtenidos del Sistema Recomendador discriminados por Distancia entre Usuarios (1 o 2)

4.5 Análisis e interpretación de los resultados de las pruebas

En base a los resultados obtenidos en las pruebas realizadas sobre el Sistema Recomendador, se observa que la similaridad entre usuarios emparejada con los tipos de relaciones que los usuarios tienen al momento de solicitar las recomendaciones, es un factor importantísimo para obtener las mismas, dado que tengo mas libertad al momento de solicitar recomendaciones de recursos al Sistema Recomendador, pudiendo asociar determinadas áreas temáticas, a determinadas relaciones en la red social, como por ejemplo: películas con mis amistades, o libros de aprendizaje con mi ámbito académico.

Otro de los factores importantes es, si queremos obtener recomendaciones de recursos de usuarios que estén directamente relacionados con el usuario que solicita la recomendación (distancia 1), o de usuarios que estén directa o indirectamente relacionados con el usuario que solicita la recomendación (distancia 2). En este sentido observamos que cuando la preferencia es de distancia 1, Precision es mayor a la Precision que observamos cuando la preferencia es de distancia 2, y Recall es el caso contrario. Esto debido a que, cuando la preferencia es de distancia 2, cabe la posibilidad de que el sistema retorne mas recursos en la recomendación, lo que hace que la Precision baje, pero pueden haber mas recursos relevantes en la misma, por lo que el Recall aumenta.

A nivel general, hemos obtenido una Precision del 65,88% y un Recall del 78,38%, ahora si el dato de entrada de distancia es 1, la Precision obtenida es del 66,29% y el Recall obtenido es del 68,69%, y si el dato de entrada de distancia es 2, la Precision obtenida es del 65,46% y el Recall obtenido es del 88,07%, observando el comportamiento descripto anteriormente de Precision vs. Recall. Aunque la Precision no varía mucho variando el dato de entrada de distancia, el Recall en cambio aumenta considerablemente.

Pero para obtener un valor único de la Precision y del Recall, utilizamos la media armónica, denominada Medida-F (F-Measure), la cuál muestra que a nivel general el sistema tiene una media del 71,59%, si el dato de entrada de distancia es 1, la media es del 67,47%, y si el dato de entrada de distancia es 2, la media es del 75,10%.

También se constató, que cuando un usuario elige un tipo de relación, que le gustaría o prefiriese que tuviera como destino un recurso evaluado por otro usuario, y variando este dato (cambiando el tipo de relación que tiene el recurso como objetivo), y manteniendo fijos los datos de la relación en la red social, la distancia y el área temática, no cambian las cantidades de recomendaciones devueltas, sino que varía el orden en que el Sistema Recomendador las muestra (varían su puntuación), influyendo este dato de entrada directamente en el cálculo de puntuación de las mismas, en beneficio del usuario que las solicita.

En cuanto a las métricas que miden la tasas de error promedio, como lo es el RMSE tanto para la Precision como para el Recall, vemos que a nivel general, son bajas, situadas en un 2,76% y 7,32% respectivamente. Ahora si diferenciamos por el dato de entrada de las distancias entre usuarios, a distancia 1, las tasas de error son 4,20% y 9,33% respectivamente, y a distancia es 2, las tasas de error disminuyen a 1,31% y a 5,31% respectivamente, lo que indica que al obtener un mayor número de recursos, vamos a encontrar mas cantidad de recursos relevantes.

En cuanto a la métrica MRR a nivel general del sistema, el valor 0,93 (muy cercano al valor 1) nos indica que es altamente probable que el primer recurso recomendado esté incluido en nuestros recursos preferentes, por lo cual, nuestro algoritmo de recomendación presenta al usuario recomendaciones de recursos, en donde en la primera posición del ranking, el recurso es lo que está buscando el usuario. Esta métrica solo considera la posición del primer recuso preferente en el ranking de recursos recomendados, pero es bueno aplicarla, dado que por lo general, todo usuario elige el primer recurso de los recomendados, y si además, ese recurso es beneficioso para el mismo, mucho mas confortante.

Por todo lo descripto anteriormente, y los resultados que se desprenden de la utilización de las técnicas y de las métricas propuestas, el algoritmo empleado, funciona de manera correcta y brinda recomendaciones personalizadas de gran relevancia.

En cuanto a la rapidez del algoritmo, observamos que el tiempo promedio obtenido al solicitar una recomendación de recursos fue de casi 7 segundos, si lo comparamos con el tiempo obtenido para solicitudes de recomendaciones a usuarios que están directamente conectados en la red social con el usuario que solicita la recomendación, el mismo fue en promedio de casi 6 segundos, mientras que si se solicitaban recomendaciones a usuarios conectados directa o indirectamente en la red social con el usuario que solicita la recomendación, la media fue de 8 segundos.

Estos tiempos son influenciados por diversos factores, como ser:

- Las características del sistema computador donde se realiza la prueba.
- Los accesos a la información contenida en el modelo de la red social, en el modelo de las relaciones y en el modelo de las áreas temáticas (base de datos orientada a grafos).
- Los accesos a la información y operaciones sobre la base de datos relacional (recursos, recursos evaluados, atributos, consultas de usuarios).
- Las cantidades de recursos, de recursos evaluados, de atributos.

- El tamaño de la red social (usuarios y relaciones entre ellos).
- El tamaño de las ontologías de vocabulario de las relaciones y de las áreas temáticas.
- La distancia entre usuarios cuando se solicitan las recomendaciones.
- La implementación del algoritmo de recomendación, y por lo tanto el orden que tiene dicho algoritmo.

Si bien como se pueden apreciar en la hoja “Cálculos Extraídos de las Ejecuciones” del archivo Datos.ods, hubieron recomendaciones devueltas por debajo del segundo, sin embargo, al solicitar recomendaciones que incluían muchos cálculos (búsqueda en todas las relaciones, en todas las áreas temáticas, usuarios directa o indirectamente conectados (distancia 2)), los tiempos llegaban a los 55 segundos en el peor caso.

Es así que el tiempo en que se retornan las recomendaciones puede variar mucho dependiendo de la cantidad de cálculo requerido. No obstante, estos tiempos se pueden llegar a corregir, detectando los cuellos de botellas en el algoritmo (directamente relacionado con el orden del mismo), así como emplear por ejemplo otras técnicas de programación (programación concurrente o en paralelo por ejemplo), y/o dotar al sistema computador donde corre el Sistema Recomendador de mayores prestaciones. También puede ser de gran ayuda el obtener los tiempos que se consumen al realizar operaciones sobre la base de datos relacional y sobre las base de datos orientada a grafos, de forma de hacer mas eficiente el sistema.

5 – Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones del proyecto y finalmente se mencionan posibles mejoras al mismo, y se incentivan trabajo a futuro a partir de lo realizado.

En este trabajo se construyó un Sistema Recomendador con Friendsourcing Semántico, basándose en la extensión y aprovechamiento de la similaridad entre usuarios, emparejada con los tipos de relaciones que los usuarios tienen dentro de una red social, para brindar recomendaciones de mayor calidad, apropiadas para usuarios que se encuentren conectados en la red social mediante algún tipo de relación.

Para esto, fue necesario realizar un estudio en profundidad de los siguientes temas: Sistemas Recomendadores, Web Semántica, Friendsourcing y Redes Sociales, así como técnicas y herramientas incluidas en los mismos, de forma de que nos ayudaran en la construcción de nuestro Sistema Recomendador, y finalmente poder ponerlo a prueba mediante métricas y técnicas conocidas, empleadas para medir la calidad y eficiencia del mismo, al momento de brindar las recomendaciones.

Para evaluar el Sistema Recomendador, se simuló una red social donde se incluyen usuarios y relaciones vinculares entre los mismos, siendo factible de ser probada en la realidad.

El Sistema Recomendador desarrollado, está provisto de una interfaz gráfica, con la cuál los usuarios pueden acceder a ella, y realizar las operaciones requeridas sobre el Sistema Recomendador, haciendo mas amigable la interacción con el mismo.

Consideramos que los objetivos planteados al comienzo del proyecto fueron alcanzados de forma exitosa, ya que propusimos la construcción de un Sistema Recomendador con Friendsourcing Semántico, y demostrando a través de pruebas realizadas sobre el mismo, que el sistema brinda recomendaciones de calidad para los usuarios.

Además, creemos que nuestro proyecto, es un aporte muy bueno para las áreas de Sistemas Recomendadores, de Web Semántica y de Redes Sociales, dado que se utilizan herramientas y técnicas provistas en dichas áreas, creando una sinergia importante entre ellas, y por sobre todo, es que son temas de gran actualidad.

Como posibles mejoras, consideramos que se deberían revisar el tema de los tiempos de ejecución del algoritmo al brindar las recomendaciones, encontrando el cuello de botella en caso de existir, ya sea: modificando el algoritmo, empleando programación concurrente y/o dotando de mayores prestaciones al sistema computador donde corre el Sistema Recomendador.

Por otra parte, hemos descubierto que no hay trabajos sobre Sistemas Recomendadores que utilicen Friendsourcing Semántico, por lo cual nuestro proyecto, sienta las bases para futuros trabajos que extiendan o mejoren ciertas características del mismo como ser atributos de calidad, vocabularios de relaciones y de áreas temáticas, algoritmo de Friendsourcing Semántico, interfaz gráfica, entre otros, o que se complemente con otros proyectos como ser: extractores de perfiles de usuarios y de recursos de una red social para proveer las fuentes de información, o añadirlo como un servicio para los usuarios de una red social, para testear su funcionamiento en ella, y poder sacar mayores conclusiones y agregarle posibles mejoras al mismo.

También sería interesante avanzar hacia un Sistema Recomendador Semántico que utilice Friendsourcing Semántico, que no solo emplee semántica sobre las relaciones que se pueden dar sobre una Red Social, sino que también emplee semántica sobre las áreas temáticas a las cuales pueda pertenecer un recurso o un ítem de información, haciendo que el Sistema Recomendador busque recomendaciones sobre ítems de información, no por el contenido textual sino por el significado del mismo. En este sentido, el poder crear mejores y mas completas ontologías de vocabulario, para las relaciones en una Red Social, y para las áreas temáticas a las que pueda pertenecer un recurso, lo harían una herramienta mas potente, mas provechosa y mas flexible para los usuarios que quieren solicitar recomendaciones al Sistema Recomendador, dado que al incorporar mayor semántica al sistema, el mismo tendrá una abanico mas amplio de posibilidades al momento de buscar en las relaciones y en los recursos, haciendo selecciones por su significado, lo que devendrá en mejores recomendaciones para los usuarios, mejorando notablemente la calidad de las mismas.

Bibliografía

[1] – Konstan, J. A. (2008). *Introduction to recommender systems*. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (p. 1373 – 1374). ACM.

[2] – Yager, R. R. (2003). *Fuzzy logic methods in recommender systems*. Fuzzy Sets and Systems. (vol. 136, no 2, p. 133 – 149).

[3] – Burke, R. (2007). *Hybrid web recommender systems*. The Adaptive Web (p. 377 – 408). Springer Berlin Heidelberg.

[4] – Bernstein, M.; Tan, D.; Smith, G.; Czerwinski, M.; Tunkelang, D.; Horvitz, E. (2010). *Personalization via Friendsourcing*. ACM Transactions on Computer-Human Interaction (TOCHI).

[5] – He, J.; Chu, W. W. (2010). *A social network-based recommender system (SNRS)*. Springer US.

[6] – Zuccarelli, G. (2015). *La Web Semántica como plataforma para sistemas de recomendación.* Tesis de Grado. Supervisores: Dr. Alejando Fernández, Dra. Alicia Díaz. Universidad de La Plata, Facultad de Informática, Argentina. 94 p.
URL: http://hdl.handle.net/10915/48096 [F. consulta: 16/11/2015].

[7] – Ricci, F.; Rokach, L.; Shapira, B.; Kantor, P. B. (2011). *Recommender Systems Handbook*.

[8] – Peis, E.; Morales del Castillo, J. M.; Delgado López., J. A. (2008). Universitat Pompeu Fabra. *Sistemas de Recomendación Semánticos: Un análisis del estado de la cuestión*. URL: http://www.upf.edu/hipertextnet/numero-6/recomendacion.html
[F. consulta: 8/9/2015].

[9] – Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. (2001). *Item-Based Collaborative Filtering Recommendation Algorithms*. In Proceedings of the 10th international conference on World Wide Web.

[10] – Schafer, J. B.; Konstan, J.; Rield, J. (1999). *Recommender Systems in E-Commerce*. In Proceedings of the 1st ACM conference on Electronic commerce.

[11] – Tunkelang, D. (2011). *Recommendations as a Conversation with the User.* In Proceedings of the 5th ACM Conference on Recommender Systems.

[12] – *Tools – Semantic Web Standards*.
URL: https://www.w3.org/2001/sw/wiki/Tools [F. consulta: 10/11/2016].

[13] – Watts, D. J. (2004). *Six Degrees: The Science of a Connected Age*. WW Norton & Company.

[14] – Scott, J. P. (2000). *Social Network Analysis: A Handbook*. SAGE.

[15] – *The FOAF Project*. URL: http://www.foaf-project.org/
[F. consulta: 15/10/2015].

[16] – Díaz, A.; Motz, R.; Fernández, A.; Valdeni de Lima, J.; López, D. (2011). *Quality Web Information Retrieval: Towards Improving Semantic Recommender Systems with Friendsourcing*. Cuadernos de Informática - Volumen 6 - Número 1.

[17] – González Bernal, D; Tansini, L.; Motz, R. (2013). *Optimización de sistemas de recomendaciones en un entorno médico mediante la utilización de contexto social obtenido de las redes sociales*.

[18] – *Latin American and Caribbean Collaborative ICT Research - Quality Health Information Retrieval*.
URL: http://www.laccir.org/web/#/Projects/HealthCare/InformationRetrieval
[F. consulta: 15/9/2015].

[19] – *Universidad del Cauca - Red Social Unisalud.* URL:
http://esalud.unicauca.edu.co/redunisalud [F. consulta: 15/9/2015].

[20] – *RELATIONSHIP: A vocabulary for describing relationships between people*.
URL: http://vocab.org/relationship/ [F. consulta: 20/9/2015].

[21] – *Linked Open Vocabularies (LOV)*. URL: http://lov.okfn.org/dataset/lov/
[F. consulta: 20/9/2015].

[22] – Burke, R. (2002). *Hybrid Recommender Systems: Survey and Experiments*. User Modeling and User-Adapted Interaction. (vol. 12, no 4, p. 331 – 370).

[23] – Sinha, R. R.; Swearingen, K. (2001). *Comparing Recommendations Made by Online Systems and Friends*. DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries.

[24] – Golbeck, J. (2006). *Generating predictive movie recommendations from trust in social networks*. International Conference on Trust Management (p. 93 – 104). Springer Berlin Heidelberg.

[25] – Berners-Lee, T.; Fischetti, M. (1999). *Weaving the Web.* Orion Business. URL:
http://www.w3.org/People/Berners-Lee/Weaving/Overview.html
[F. consulta: 10/9/2015].

[26] – O'Reilly, T. (2005). *What Is Web 2.0*.
URL: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html
[F. consulta: 10/9/2015]

[27] – *Guía Breve de Web Semántica (W3C)*. URL:
http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica [F. consulta: 12/9/2015].

[28] – *List Of Metatdata Standards*. URL: http://www.dcc.ac.uk/resources/metadata-standards/list [F. consulta: 14/9/2015].

[29] – *DC: Dublin Core Metadata Initiative.* URL: http://dublincore.org/
[F. consulta: 14/9/2015].

[30] – Berners-Lee, T. (2000). *Semantic Web on XML – Architecture. URL:* http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html
[F. consulta: 12/9/2015].

[31] – Manola, F.; Miller, E.; McBride, B. (2004). *RDF primer. W3C recommendation*. (vol. 19, no 1-107, p. 6).
URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
[F. consulta: 13/9/2015].

[32] – Brickley, D.; Guha, R. V.; McBride, B. (2014). *RDF Schema 1.1*. W3C Recommendation. World Wide Web Consortium.
URL: https://www.w3.org/TR/rdf-schema/ [F. consulta: 13/9/2015]

[33] – Studer, R.; Benjamins, V. R.; Fensel, D. (1998). *Knowledge engineering: Principles and methods. Data & Knowledge Engineering*.
(vol. 25, no 1, p. 161 – 197).

[34] – *Vista General del Lenguaje de Ontologías Web (OWL).*
URL: http://www.w3.org/2007/09/OWL-Overview-es.html [F. consulta: 13/9/2015].

[35] – *OWL 2 web ontology language document overview (Second Edition)*. (2012). World Wide Web Consortium. URL:https://www.w3.org/TR/owl2-overview/
[F. consulta: 10/11/2016]

[36] – Gruber, T. R. (1995). *Toward Principles for the Design of Ontologies Used for Knowledge Sharing?*. International journal of human-computer studies. (vol. 43, no 5, p. 907 – 928). URL: http://ksl-web.stanford.edu/knowledge-sharing/papers/onto-design.rtf [F. consulta: 14/9/2015].

[37] – *Protégé*. URL: http://protege.stanford.edu/ [F. consulta: 20/9/2015].

[38] – *Apache Jena*. URL: https://jena.apache.org/ [F. consulta: 20/9/2015].

[39] – *Model (Apache Jena)*. URL: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Model.html [F. consulta: 20/9/2015].

[40] – *SPARQL Query Language for RDF.*
URL: http://www.w3.org/TR/rdf-sparql-query/ [F. consulta: 20/9/2015].

[41] – *Web Of Trust RDF Ontology*. URL: http://xmlns.com/wot/0.1/
[F. consulta: 14/9/2015].

[42] – *FOAF Vocabulary Specification 0.99*. URL: http://xmlns.com/foaf/0.1/ [F. consulta: 14/9/2015].

[43] – *XML Signature WG*. URL: http://www.w3.org/Signature/
[F. consulta: 14/9/2015].

[44] – *WorldCat*. Catálogo en línea gestionado por el OCLC (Online Computer Library Center). URL: http://www.worldcat.org/ [F. consulta: 12/7/2016].

[45] – *Internet Movie Database (IMDb)*. Base de datos cinematográfica en línea. URL: http://www.imdb.com/ [F. consulta: 20/6/2016].

[46] – Schütze, H. (2008). *Introduction to Information Retrieval*. Proceedings of the international communication of association for computing machinery conference. Cap 8. URL: http://nlp.stanford.edu/IR-book/pdf/08eval.pdf [F. consulta: 20/7/2016].

[47] – Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; Riedl, J. T. (2004). *Evaluating collaborative filtering recommender systems*. ACM Transactions on Information Systems (TOIS). (vol. 22, no 1, p. 5 – 53).

Apéndice

Anexo 1 – Extracción de lista de relaciones de usuarios(perfiles) para un perfil de usuario de Facebook.

Se muestra lo devuelto al consultar la listas de usuarios(y sus perfiles) relacionados con un usuario en particular. Para ello se accede a: https://developers.facebook.com/tools/explorer/

Por un tema de confidencialidad de los datos se manipularon los nombre de los usuarios e identificadores, pero la estructura mostrada es la devuelta en la consulta.


```
{
  "id": "XXXX ZZZZ XXXX",
  "name": "Diego Bastiani",
  "family": {
    "data": [
      {
        "name": "XXX YYY",
        "relationship": "cousin",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "relationship": "cousin",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "relationship": "aunt",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "relationship": "cousin",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY"
        "relationship": "cousin",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "relationship": "cousin",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "relationship": "cousin",
        "id": "ZZZZ"
      }
```

```
    ],
    "paging": {
      "cursors": {
        "after": "XXX YYY",
        "before": "XXX YYY"
      }
    }
  },
  "friends": {
    "data": [
      {
        "name": "XXX YYY",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "id": "ZZZZ"
      },
      {
        "name": "XXX YYY",
        "id": "ZZZZ"
      }
    ],
    "paging": {
      "next": "https://graph.facebook.com/v2.4/XXXX ZZZZ XXXX/friends?
access_token="XXX YYY"
    },
    "summary": {
      "total_count": XXX
    }
  },
}
```

Anexo 2 – Extracción de lista de relaciones de usuarios(perfiles) para un perfil de usuario de Google+.

Se muestra lo devuelto al consultar la listas de usuarios(y sus perfiles) relacionados con un usuario en particular. Para ello se accede a: https://developers.google.com/+/web/api/rest/latest/people/list

Por un tema de confidencialidad de los datos se manipularon los nombre de los usuarios, pero la estructura mostrada es la devuelta en la consulta.


```
{
 "kind": "plus#peopleFeed",
 "etag": "\"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\"",
 "title": "Google+ List of Visible People",
 "totalItems": XXX,
 "items": [
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY\"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "Marcos Suiffet",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY\"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY\"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
```

```
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },  {
```

```
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY\"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  },
  {
   "kind": "plus#person",
   "etag": "\"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY\"",
   "objectType": "person",
   "id": "XXXXXXXXXXXXXXXXXXXXXXXXX",
   "displayName": "ZZZZZZZZZZZZZZZZZZZZZ",
   "url": "https://plus.google.com/ZZZZZZZZZZZZZZZZZZZZ",
   "image": {
    "url": "https://lhX.googleusercontent.com/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/photo.jpg?sz=50"
   }
  }
 ]
}
```

Anexo 3 – Técnicas de Filtrado de información utilizadas por los Sistemas Recomendadores.

Existen diversas clasificaciones de las técnicas de recomendación que son utilizadas por los sistemas de recomendaciones. La clasificación indicada es la provista por Burke[3], y comprende seis diferentes clases de tratamiento que se presentan a continuación:

- Filtrado colaborativo
- Basadas en contenido
- Basadas en conocimiento
- Basadas en la comunidad
- Demografico
- Hibridos

A continuación se brinda una descripción de las diferentes técnicas de recomendación.

Filtrado Colaborativo

En los sistemas de filtrado colaborativo se reconocen similitudes entre los usuarios basados en las puntuaciones de ítems y se generan recomendaciones en base a comparaciones entre usuarios similares. A su vez estos sistemas se pueden dividir en dos categorías: *basados en memoria* (memory-based) y *basados en modelo* (model-based).
En los sistemas basados en memoria se comparan los usuarios directamente entre sí utilizando correlación u otras medidas. En los sistemas basados en modelo se construye un modelo a partir de los datos históricos de puntuaciones y éste es utilizado para la generación de recomendaciones [22].
En los sistemas de filtrado colaborativo los datos base son las puntuaciones de ítems realizadas por los usuarios, los datos de entrada son las puntuaciones de ítems realizadas por el usuario al que se le quiere brindar las recomendaciones, y el algoritmo de recomendación consiste en identificar usuarios similares y extrapolar las preferencias de los ítems a partir de las puntuaciones hechas por dichos usuarios [22].
La gran fortaleza de esta técnica es que es completamente independiente de la representación de los ítems que son recomendados [22].

Algunas desventajas que presenta la tecnica:

- Problema de Cold-Start: Algunas caracteristicas de este problema son: Usuario nuevo (o early rater), Sitio nuevo (item nuevo), Comunidad nueva.
- Problema de Dispersión (o Sparsity): si el número de usuarios es pequeño en relación al volumen de información en el sistema, se corre el riesgo de que el cubrimiento de ratings se vuelva muy disperso. Achicando la colección de ítems recomendables.
- Problema de Escalabilidad: a medida que la cantidad de usuarios y de ítems crece, también crece la cantidad de cómputos de vecinos mas cercanos para la determinación de usuarios similares, y como los cálculos se hacen en tiempo real, el sistema puede colapsar.
- Confianza entre usuarios
- Problema de la “Oveja Gris”: existen usuarios donde sus perfiles caen entre clases existentes de usuarios, haciendo difícil determinar para ellos una recomendación adecuada.
- Problema de la Sinonimia: se produce por la escasez de cualquier forma de interpretación semántica. Ítems similares no se trataran de tal manera cuando se hagan las recomendaciones.
- Problema de Subjetividad: respecto a la naturaleza de los ratings.

Basado en el Contenido

En los sistemas de recomendaciones basados en contenido, los ítems de interés son definidos por sus características, como por ejemplo las palabras en un documento de texto. Estos sistemas aprenden el perfil de los intereses de un usuario basándose en las características de los ítems que el usuario ha puntuado, a esto se le llama *correlación ítem a ítem* (ítem-to-item correlation). Para determinar los perfiles se utiliza algún método de aprendizaje, como por ejemplo árboles de decisión o redes neuronales. Los perfiles son modelos a largo plazo y se actualizan a medida que se observan más preferencias del usuario [22].
En los sistemas basados en contenido, las caracteristicas de los items son los datos base, las puntuaciones de ítems hechas por el usuario al que se le quiere brindar las recomendaciones los datos de entrada, y el algoritmo de recomendación consiste en generar el perfil del usuario que se ajuste al comportamiento en las puntuaciones de ítems que luego es usado para determinar las preferencias [22].

Algunas desventajas de estos sistemas:

- se necesitan muchas puntuaciones de ítems para poder generar perfiles confiables.
- están limitados por las características de los ítems que recomiendan.

Basado en el Conocimiento

En los sistemas de recomendaciones basados en conocimiento se realizan recomendaciones de ítems basadas en inferencias acerca de las necesidades y preferencias del usuario. En estos sistemas hay tres tipos de conocimiento:

- *conocimiento del catálogo*, que es el conocimiento de los ítems que son recomendados y sus características
- *conocimiento funcional* que es el conocimiento para mapear las necesidades del usuario y el ítem que podría satisfacer estas necesidades
- *conocimiento del usuario*, que es el conocimiento que el sistema debe tener sobre el usuario para poder brindar buenas recomendaciones

En estos sistemas las características de los ítems y el conocimiento de cómo estos ítems satisfacen las necesidades de los usuarios son los datos base, la descripción de las necesidades o intereses del usuario al que se le quiere brindar las recomendaciones es la entrada, y el algoritmo de recomendación consiste en inferir aquellos ítems que satisfagan las necesidades o sean de interés para el usuario activo [22].

Ventaja: apropiados para exploración casual.

Desventaja: necesitan adquirir el conocimiento.

Basado en la Comunidad

Este tipo de sistema recomienda los tópicos en base a las preferencias de los amigos del usuario. Las personas tienden a confiar más en las recomendaciones de sus amigos que en las de desconocidos [23]. Esto unido a la creciente popularidad de las redes sociales, está incrementando el interés en este tipo de sistemas, también conocidos como *sistemas de recomendación social* [24]. Estos sistemas recomendadores adquieren información acerca de los contactos o relaciones sociales del usuario y de las preferencias de sus amigos. Las recomendaciones se basan en calificaciones dadas por los amigos del usuario. El crecimiento continuo de las redes sociales posibilitan la adquisición de estos datos.

Demográfico

En los sistemas de recomendaciones demográficos se categorizan los usuarios a partir de sus atributos personales y se realizan recomendaciones de ítems basándose en clases demográficas.
En estos sistemas los datos base son las puntuaciones de ítems hechas por los usuarios e información demográfica de los mismos, los datos de entrada son los atributos demográficos del usuario al que se le quiere brindar las recomendaciones, y el algoritmo de recomendación consiste en identificar aquellos usuarios que son demográficamente similares al usuario activo y se extrapolan las preferencias a partir de las puntuaciones de ítems.
A modo de ejemplo, si una mujer busca comprar productos de belleza, el sistema de recomendación tendrá en cuenta su información demográfica (sexo, edad, país de residencia, etc.) para brindar recomendaciones de productos que sean acordes a dicha mujer y evitará recomendar productos que potencialmente no sean de su interés como por ejemplo perfumes para hombre.
Una ventaja de esta técnica es que no se requiere de un historial de puntuaciones del usuario como en las técnicas de filtrado colaborativo y en las basadas en contenido.
Un problema que presenta esta técnica es que debe reunir información demográfica del usuario [22].

Híbridos

Estos sistemas recomendadores se basan en la combinación de dos o más de las técnicas anteriores. Se pueden combinar dos técnicas para aprovechar las ventajas de una y solucionar las desventajas de la otra, en un modelo en el cual ambas técnicas se apoyan mutuamente. De esta manera se logra una mejor performance [22].
Por ejemplo los sistemas de filtrado colaborativo tienen el problema asociado con la aparición de un nuevo tópico, del cual no existe calificación. Por otro lado los sistemas basados en contenido no tienen esta limitación, dado que fácilmente pueden conocer las características de un nuevo tópico y verificar si éstas son del gusto del usuario.
A continuacion se nombran diversos metodos de estos sistemas:

- Ponderado (Weighted): Un sistema híbrido de recomendación ponderado es aquel sistema en que el puntaje para un ítem a recomendar es computado a partir de los resultados de todas las técnicas de recomendación disponibles en el sistema. Por ejemplo, se podría hacer una combinación lineal de los puntajes de recomendación. Una ventaja de estos sistemas es que es fácil ajustar el híbrido para que éste brinde mejores resultados [22].
- Conmutador (Switching): Un sistema híbrido de recomendación conmutador es

aquel sistema que conmuta entre las diferentes técnicas de recomendación dependiendo de la situación actual. Estos sistemas presentan una complexidad extra al proceso de recomendación, que es la determinación del criterio para conmutar entre las diferentes técnicas [22].

- En cascada (Cascade): Un sistema híbrido de recomendación en cascada es aquel sistema en el cual primero se aplica una técnica de recomendación para generar recomendaciones candidatas y luego se aplica una segunda técnica que refina dichas recomendaciones [22].
- Mixto (Mixed): Un sistema híbrido de recomendación mixto es aquel sistema en que se presentan al mismo tiempo las recomendaciones obtenidas de las diferentes técnicas [22].
- Combinacion de caracteristicas (Feature combination): Un sistema híbrido de recomendación con combinación de características es aquel sistema en que las características que se obtienen de las diferentes técnicas de recomendación se combinan en un único algoritmo de recomendación [22].
- Aumento de caracteristicas (Feature augmentation): Un sistema híbrido de recomendación con aumento de características es aquel sistema en el cual primero se aplica una técnica de recomendación para producir un puntaje o una clasificación de un ítem y luego esta información se incorpora al proceso de recomendación de la siguiente técnica. En otras palabras, la salida de una técnica de recomendación se utiliza como la entrada de otra técnica [22].
- Nivel sobre nivel (Meta-level): Un sistema híbrido de recomendación nivel sobre-nivel es aquel sistema en que el modelo generado por una técnica de recomendación es utilizado como la entrada de otra técnica de recomendación [22].

Anexo 4 – La Web Semántica

A continuación se realiza una breve pero concisa descripción de lo que es la Web Semántica.

El camino hacia la Web Semántica

En 1989 Tim Bemers-Lee[25] (creador de la World Wide Web, del lenguaje html y del protocolo HTTP) empieza a escribir el primer Navegador, implementando los conceptos de hipertexto. Buscando contar con gran cantidad de información, para establecer nuevas formas de trabajo en equipo y para analizar la estructura social a través de Internet.
En 1991 es lanzado el proyecto, evolucionando hacia las páginas Web o Web 1.0.
En 2004 Tim O'Reilly[26] se refirió a la Web 2.0 (también llamada Web Social) como una conjunto de aplicaciones Web, basadas en comunidades de usuarios y una gama especial de servicios, permitiendo pasar de consumidores de información a productores.
En esta se da un alto grado de colaboración y la voluntad de compartir recursos. El usuario es el centro de atención: decide qué y cómo usarlo. Tiene interfaces y usabilidad altas. La representación semántica es poco compleja. No hay autoridad central y se tiene un cierto grado de dificultad al crear aplicaciones para utilizarlas dado que no están normalizadas.
En 1999 Tim Bemers-Lee[25] con su libro “Tejiendo la Red”, habla del origen de la Web y el Objetivo de la Web Semántica o Web 3.0

Problemas de la web actual

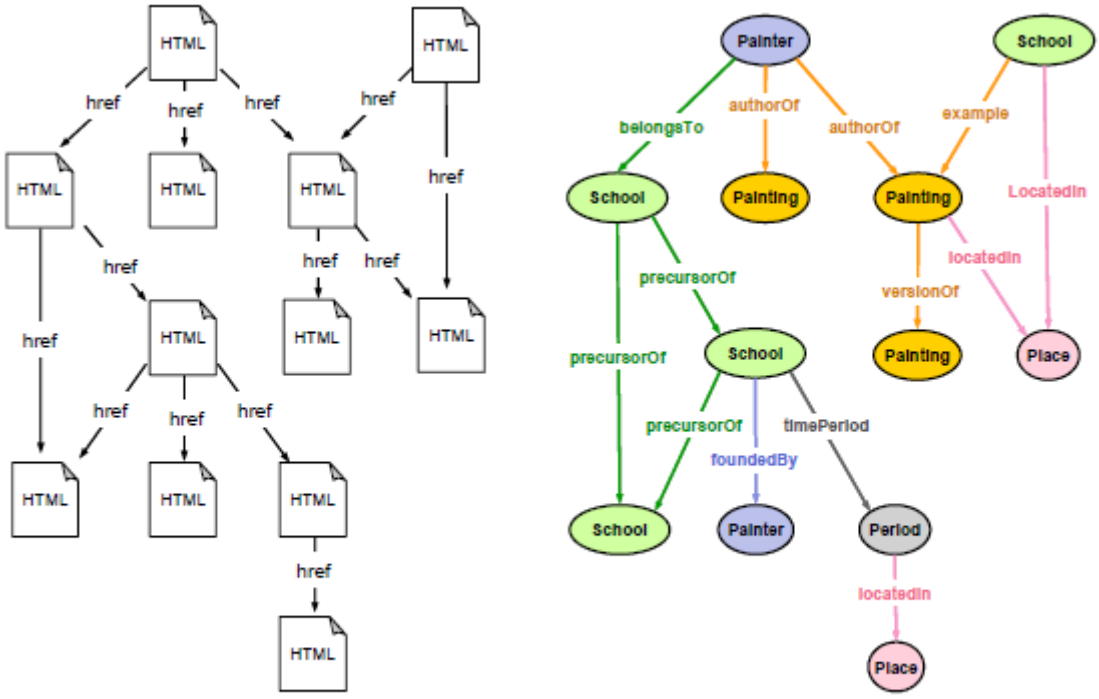
Actualmente podemos: almacenar, consultar, publicar y editar la información de la Web fácilmente, estableciendo su relación de manera propia y personalizada.
Lo anterior genera: gran cantidad de información suelta, redundante y de calidad dudosa. Problemas de interoperabilidad por formatos y sistemas heterogéneos. Gran costo de tiempo en búsqueda de información, dado que la semántica debe ser analizada por el usuario.

¿Qué es la Web Semántica?

La Web Semántica[27] es una **Web extendida**, dotada de **mayor significado** en la que cualquier usuario en Internet podrá encontrar **respuestas a sus preguntas** de forma más rápida y sencilla gracias a una **información mejor definida**. Al dotar a la Web de más significado y, por lo tanto, de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la

utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante.

Actualmente la web se asemeja a un grafo formado por nodos del mismo tipo, y arcos (hiperenlaces) igualmente indiferenciados. Por el contrario en la web semántica cada nodo (recurso) tiene un tipo y los arcos representan relaciones explícitamente diferenciadas.
La siguiente figura muestra la diferencia entre ellas:



a) Web actual b) Web semántica

Es así que la Web Semántica propone:

- Utilización de una sintaxis común (XML) y la expresión del conocimiento en estructuras simples predefinidas (ej.: RDF).
- Utilización de vocabularios de metadatos y ontologías.
- Referenciar los términos a recursos que expliquen su contenido mediante la URI.
- Sea utilizada por personas y máquinas.

¿Para que sirve la Web Semántica?

Permite realizar lo siguiente: organizar la gran cantidad de información y datos sueltos existentes en la Web, utilizar un método para integrar recursos con diferentes formatos, y la interoperabilidad entre diversos dispositivos y plataformas.

¿Como funciona la Web Semántica?

La Web construye una Base de Conocimiento sobre sus usuarios, además tiene relacionados los datos y la información con su significado, y es capaz de entender de manera exacta lo que se le pide que busque. No se trata de una inteligencia artificial mágica que permita a las máquinas entender las palabras de los usuarios, es sólo la habilidad de una máquina para **resolver problemas bien definidos**, a través de **operaciones bien definidas** que se llevarán a cabo sobre **datos existentes bien definidos**[27].

¿Cuáles son los pilares básicos de la Web Semántica?

Los Metadatos

Debido a la gran diversidad y volumen de las fuentes y recursos en internet, se hizo necesario establecer un mecanismo para etiquetar, catalogar, describir y calificar los recursos presentes en la web con el fin de facilitar la posterior búsqueda y recuperación de la información. Este mecanismo los constituyen los llamados metadatos que es la información acerca de un recurso (datos de datos). Tienen como requisito que la información debe estar estructurada y debe ser comprensible para las máquinas. En este sentido, la Web Semántica propone el uso de XML para anotar el contenido de los documentos.

Existen diversos formatos y estándares de metadatos[28], uno de los mas conocidos para describir recursos de información en la web es Dublin Core (DCMI)[29].

Recurso en Internet

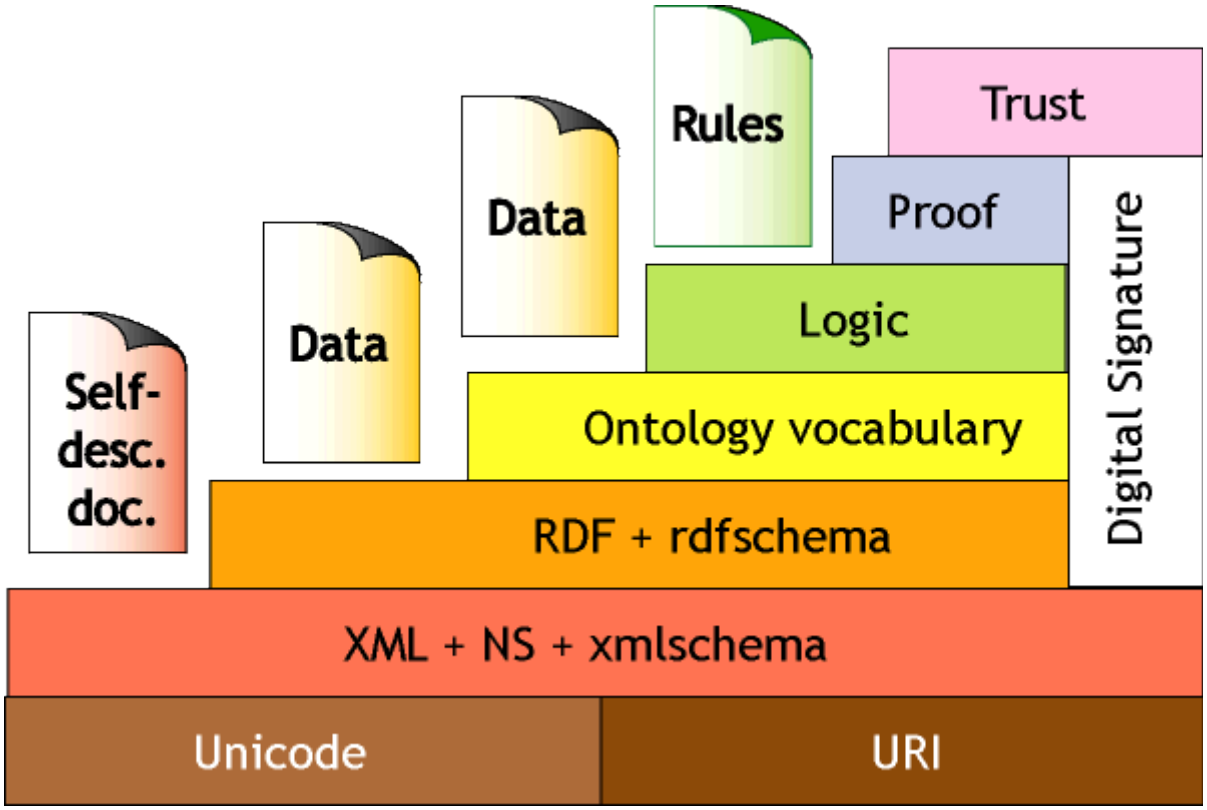
Páginas html, xhtml, imágenes, videos, documentos en diversos formatos, etc..

La Web Semántica se basa en dos conceptos fundamentales: descripción y manipulación del significado de la información.

- Descripción: semántica, metadatos, ontologías.
- Manipulación: lógica, motores de inferencia.

Anexo 5 – Arquitectura de la Web Semántica y sus componentes.

Según Berners-Lee [30], la arquitectura de la Web Semántica se podría representar de la siguiente forma:



A continuación detallamos los componentes que conforman la misma:

Juego de caracteres (Unicode)

Las máquinas deben ser capaces de reconocer los caracteres que intercambian. Se trata de una codificación del texto que permite utilizar los símbolos de diferentes idiomas sin que aparezcan caracteres extraños. De esta forma, se puede expresar información en la Web Semántica en cualquier idioma. Vendría a ser el alfabeto.

Identificador uniforme de recurso (URI)

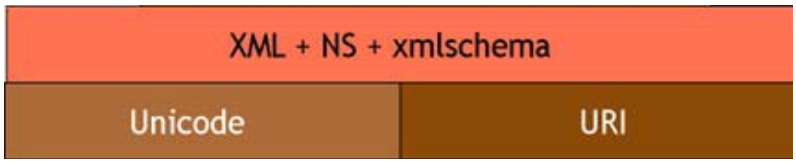
Cada recurso en la Web Actual tiene una URL(localizador uniforme de recurso).

Cada recurso en la Web Semántica tiene una URI (identificador uniforme de recurso).

Por lo tanto una URI es el identificador único que permite la localización de un recurso que puede ser accedido vía web.

Una URI es la combinación de URL más el URN (descripción del espacio de nombre).

XML, NS y XML-Shema



Se trata de la capa más técnica de la Web Semántica. En esta capa se agrupan las diferentes tecnologías que hacen posible que los agentes puedan entenderse entre ellos.

Extensible Markup Language (XML)

Permite describir datos mediante etiquetas, y define una estructura de árbol procesable por las máquinas.

XML ofrece un formato común para intercambio de documentos.

Ej.:

```
<Creator>
        <uri>http://www.proyectodegrado.com</uri>
        <name>Recomendador con Friendsourcing Semántico</name>
</Creator>
```

Espacio de Nombres (NS)

Un archivo XML puede contener nombres de elementos o atributos procedentes de más de un vocabulario XML. Si a cada uno de estos vocabularios se le da un espcaio de nombres, un ámbito semántico propio, referenciado a una URI donde se listen los términos que incluye, se resuelve la ambigüedad existente entre elementos o atributos que se llamen igual (homonimia). Los nombres de elementos dentro de cada espacio de nombres deben ser únicos.

XML-Schema

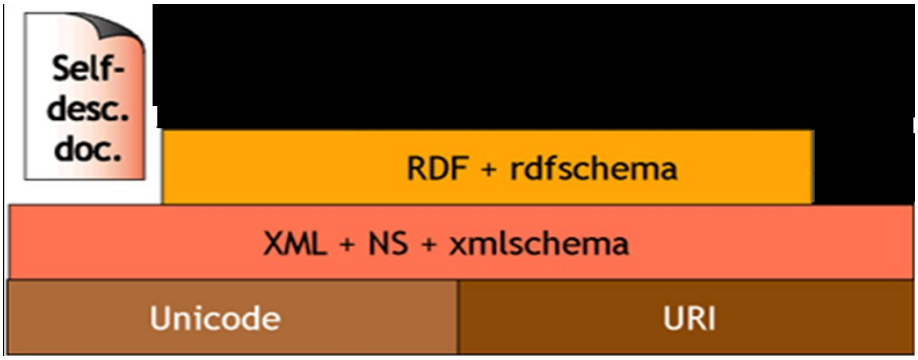
Permite definir tipos de documentos XML. Esto es, ofrecer una plantilla para elaborar documentos estándar. De esta forma, aunque se utilicen diferentes fuentes, se crean documentos uniformes en un formato común y no propietario.

Ej.:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/08/XMLSchema">
        <xsd:element name="persona" type="tipoPersona"/>
        <xsd:element name="comentario" type="xsd:string"/>
        <xsd:complexType name="tipoPersona">
                <xsd:sequence>
                        <xsd:element name="nombre" type="xsd:string"/>
                        <xsd:element ref="comentario" minOccurs="0"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

A este nivel podemos decir que XML tiene a favor una sintáxis fácilmente procesable pero tiene como contras que no infiere, que pueden haber etiquetas iguales en documentos distintos y que el significado de las etiquetas no está definido.

Resource Description Framework (RDF) y RDF-Schema



Basada y apoyada en la capa anterior, esta capa define el *lenguaje universal* con el cual podemos expresar diferentes ideas en la Web Semántica. Tanto esta capa como la anterior corresponden a las anotaciones de la información (***metadatos***).

RDF

Un recurso en la web se identifica por su URI y sus relaciones con otros elementos de la web, entonces todo lo que hay en la web es un recurso y todo lo que hay en la web se puede representar en RDF[31].

Entonces:

- **R**esource: Recursos que pueden ser nombrados por URIs.
- **D**escription: Afirmaciones sobre las propiedades de los recursos.
- **F**ramework: Un modelo común.

Una descripción RDF es un conjunto de proposiciones simples (también llamadas sentencias o declaraciones). A una proposición se conoce también como una tripleta, porque está compuesta de tres elementos: un Sujeto, un Predicado y un Objeto. Estas sentencias se pueden representar formalmente usando la tripleta <sujeto, predicado, objeto>, pero existe otra forma de notación que es mostrar una sentencia mediante grafos dirigidos.

En este sentido vemos que:

- Sujeto: es el recurso sobre el cual queremos hablar. (URI)
- Predicado: relaciona el recurso sobre el que queremos hablar con otro recurso o con un valor (es una propiedad del recurso). (URI)
- Objeto: Es el valor de la propiedad para el recurso del que queremos hablar. Puede ser otro recurso también. (Literal (número, cadena de caracteres, etc.) o URI)

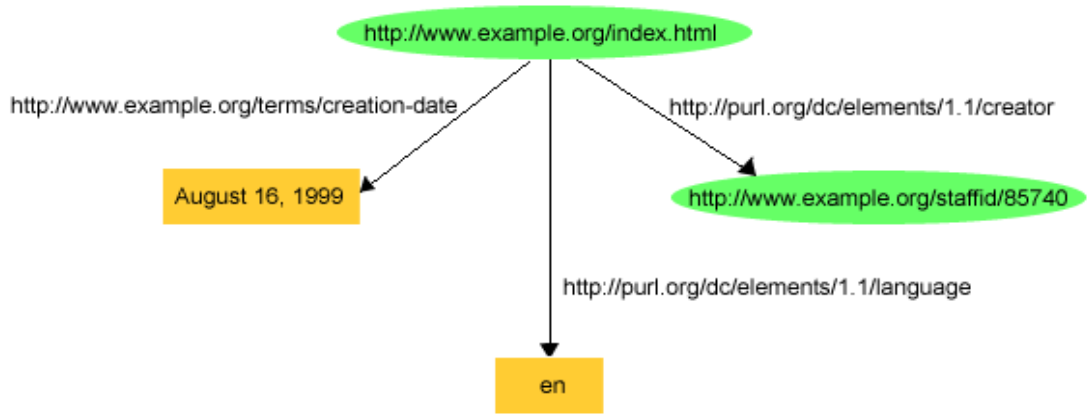
Ej. (en tripletas):

<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/creator> <http://www.example.org/staffid/85740>.

<http://www.example.org/index.html> <http://www.example.org/terms/creation-date> ”August 16, 1999”.

<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/language> ”en”.

Ej. (en formato de grafo):



Diferencia con XML:

Un mismo documento se puede modelar de distintas formas con XML:

```
<Libro idTitulo=“El_Quijote"xmlns="http://www.ejemplo.org/libro">
        <autor>Cervantes</autor>
</Libro>

<obra>
        <libro>El Quijote</libro>
        <autor>Cervantes</autor>
</obra>

<autor nombre=“Cervantes”>
        <Libro>El Quijote</Libro>
</autor>
```

Pero no en RDF:

```
<rdf:RDF rdf:ID=“El_Quijote"
        xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xml:base="http://www.ejemplo.org/libro#">
        <rdf:Description ID=“quijote”>
                <dc:title>El Quijote</dc:title>
                <dc:creator>Miguel de Cervantes</dc:creator>
        </rdf:Description>
</rdf:RDF>
```

Desventajas de RDF:

- No se permite el manejo/definición de restricciones de integridad.
- Vocabulario pobre:
  - Sólo permite:
    - Definir que algo es de determinada clase (type).
    - Definir que algo es una propiedad (rdf:property).
    - Definir listas y conjuntos.
    - Hacer afirmaciones sobre ternas (reificación)..
- El mecanismo para describir recursos no creen ninguna asunción sobre un dominio de aplicación particular, ni define a priori la semántica de algún dominio de aplicación.

RDF-Schema

RDFS es una extensión semántica de RDF [32], que provee elementos básicos para la descripción de ontologías (también llamadas vocabularios RDF) con el objetivo de estructurar los recursos RDF. Proporciona la manera de definir las clases y las propiedades específicas de la aplicacion en RDF. RDFS no proporciona las clases sino que proporciona el marco necesario para poder definirlas.
Clases en RDFS se parecen mucho a las clases en lenguajes orientados a objetos. Esto permite que los recursos se definan como instancias de clases y subclases de las clases.

Elementos básicos:

- *rdfs:Class*, permite declarar recursos como clases para otros recursos. A través de la propiedad rdf:type, se puede asignar un tipo al recurso, un

recurso de tipo rdfs:Class puede entonces ser tipo de otro recurso. Por ejemplo, podemos definir el recurso Auto como un rdfs:Class, y a su vez podemos definir el recurso Volkswagen Gol con la propiedad rdf:type y siendo el objeto de esa propiedad el recurso Auto, entonces el tipo de Volkswagen Gol sera Auto.
La definición de rdfs:Class es recursiva: rdfs:Class es la rdfs:Class de cualquier rdfs:Class.

- *rdfs:Resource*, es la clase a la que pertenecen todos los recursos.
- *rdfs:Literal*, es la clase de todos los valores literales, cadenas y enteros.
- *rdfs:Datatype*, es la clase que abarca los tipos de datos definidos en el modelo RDF.
- *rdfs:Property*, es la clase que abarca las propiedades.

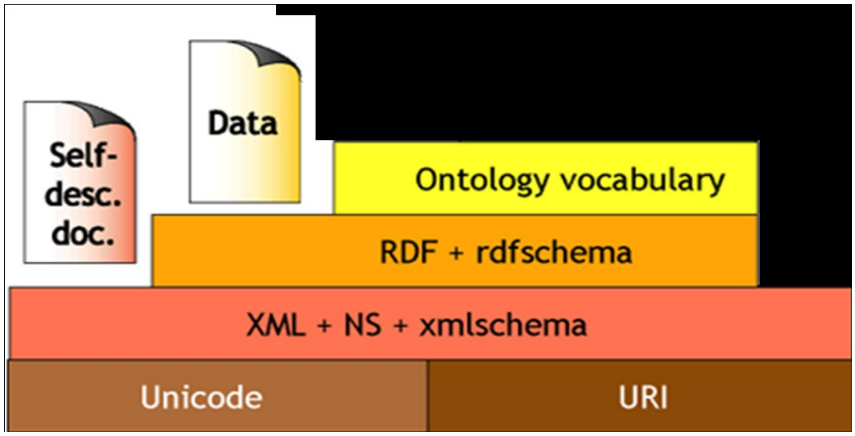
Elementos que definen relaciones:

- *rdfs:subClassOf*, es una instancia de rdf:Property (o sea una propiedad) que permite definir jerarquías. Relaciona una clase con sus superclases.
- *rdfs:subPropertyOf*, es una instancia de rdf:Property que permite definir jerarquías de propiedades.

Restricciones de propiedades:

- *rdfs:domain*, es una instancia de rdf:Property que especifica el dominio de una propiedad P. Esto es, la clase de los recursos que aparecen como sujetos en las tripletas donde P es predicado.
- *rdfs:range*, es una instancia de rdf:Property que especifica el rango de una propiedad P. Esto es, la clase de los recursos que aparecen como objetos en las tripletas donde P es predicado.

Ontologías



Según Studer[33], ”una ontología es una **especificación formal** y **explícita** de una **conceptualización compartida**”.

Donde:

- **Conceptualización**: Modelo abstracto de algún fenómeno del mundo. Se identifican los conceptos que son relevantes.
- **Explícito**: Especificar los distintos conceptos que conforman una ontología.
- **Formal**. La especificación debe representarse por medio de un lenguaje de representación formalizado.
- **Compartida**. Una ontología debe ser aceptada, como mínimo, por el grupo de personas que deben usarla (consenso).

Las ontologías son una forma de representación del conocimiento y la información. Son la base para la arquitectura de la Web Semántica y contienen un vocabulario aceptado por una amplia comunidad. Además pueden inferir conocimiento implícito a partir del conocimiento representado.

Lenguajes para definir ontologías

Los lenguajes de ontologías deben permitir escribir conceptualizaciones explícitas y formales. Los principales requisitos son:

- Una sintáxis bien definida.
- Posibilidad de razonamiento eficiente.
- Suficiente riqueza semántica.

Sin embargo, cuanto más rico es el lenguaje, más ineficiente es su razonamiento,

llegando incluso hasta el punto de ser “incomputable”, es por ello que necesitamos un compromiso entre ambas cosas.

- *RDF Esquema -> Problema:*
  - RDF-Schema es un lenguaje de construcción de ontologías muy primitivo para modelar dado que no hay restricciones en el rango o dominio y no hay restricciones de existencia/cardinalidad.
  - Muchas primitivas que serían deseables no están como ser las propiedades transitivas, inversas o simétricas.
  - No hay disjunción (por ej. decir que “macho y hembra” son disjuntos).
  - El razonamiento que podemos hacer pues es muy pobre.
  - Necesidad de una capa más rica en semántica encima de RDF y RDF-Schema.

- *OWL*:
  - Se basa en lógica descriptiva.
  - Posee un nivel avanzado de razonamiento para la Web Semántica.

OWL (Web Ontology Language)

Web Ontology Languaje [34], es un vocabulario que funciona como lenguaje para crear ontologías en la web. Fue creado en el 2006 y aceptado por W3C como estándar en 2007 en su version OWL 1.1. Tiene por objetivo proveer reglas a los vocabularios para aumentar su expresividad. Estas reglas incluyen: jerarquías de clases, cardinalidad, relaciones de conjunto, tipologías de propiedades mas complejas, etc.. Lo que proporciona la posibilidad de crear ontologías que pueden representarse en un paradigma muy parecido a la orientacion a objetos. Cabe destacar que OWL no es el único vocabulario con este objetivo, ya que RDFS es también un conjunto de estándares mucho mas básico que permite estas características. En la actualidad existe la version 2 de OWL [35] que es mucho mas completa.

Diseño de Ontologías

Según Gruber[36] los principios de diseño de ontologías son los siguientes:

1. Claridad: definiciones objetivas, claras, completas con condiciones necesarias y suficientes. Con documentación en lenguaje natural (claridad implica comunicar el significado de los términos).

2. Coherencia: que las inferencias que se realicen sean consistentes con las definiciones (las inferencias a través de axiomas no deben contradecir las definiciones de los conceptos).
2. Extensibilidad: es anticipar el uso compartido del vocabulario.
3. Especificación es independiente de codificación por símbolos (una moneda no se definiria como tipo número sino como moneda)
4. Mínimo compromiso ontológico: cuantos menos compromisos mejor para llegar a un acuerdo, se debe procurar el uso consistente del vocabulario esencial para comunicar el conocimiento.

Además:

- Conceptos similares con definiciones similares.
- Estandarizar nombres.

Herramientas de desarrollo de Ontologías

Este grupo incluye herramientas y paquetes integrados que pueden ser utilizados para construir una nueva ontología desde cero.

Ej.: KAON, OilEd, Ontolingua Server, OntoSaurus, Protégé, SWOOP, Topbraid Composer, WebODE y WebOnto.

Protégé

Ideada por la Universidad de Stanford con la ayuda de la Universidad de Mánchester, Protégé[37] es un editor de código abíerto usado para construir Ontologías y un marco general para representar el conocimiento. Está escrito en Java que es un lenguaje de programación orientado a objetos. Se usa para construir aplicaciones para la Web Semántica, para hacer una descripción semántica de la información. Sus archivos se hacen en el lenguaje OWL.

Herramientas de consulta de ontologías y motores de inferencia

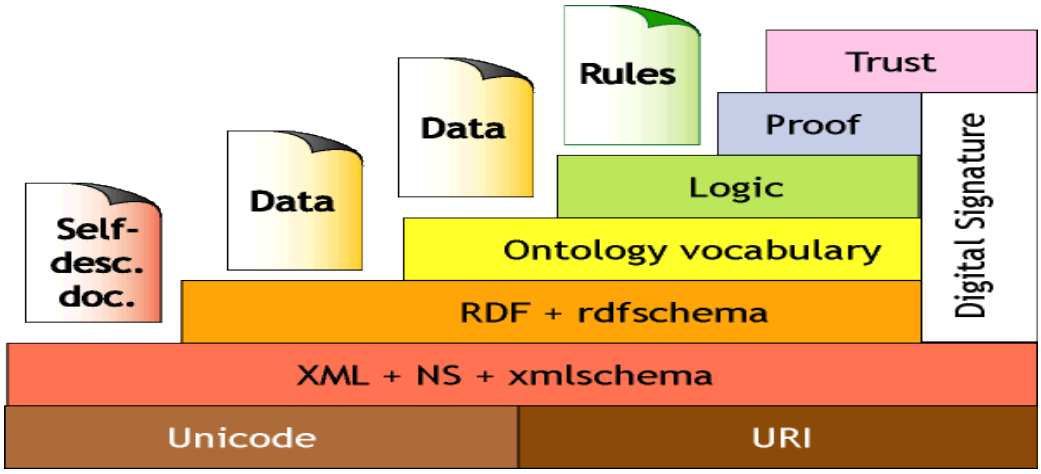
Éstas permiten la consulta de ontologías de manera sencilla y llevan a cabo inferencias con ellas. Normalmente están muy relacionadas con el lenguaje utilizado en la implementación de las ontologías.

Ej.: ICS-FORTH RDFSuite, Sesame, Jena, KAON API, TRIPLE, Cerebra y Ontopia Knowledge Suite.

Jena

Jena[38] es un framework de Java para crear aplicaciones en la Web Semántica. Proporciona unas extensas librerías de Java para desarrollar código que maneje RDF, RDF-Schema, OWL y SPARQL, siguiendo las recomendaciones publicadas por el W3C. Jena incluye un motor de inferencia basado en reglas que permita realizar razonamientos sobre ontologías OWL y RDF-Schema, y una variedad de estrategias de almacenamiento para guardar tripletas RDF en la memoria o en el disco, esto mediante el uso de modelos abstractos(Model[39]). Los modelos se consultan a través de SPARQL[40].

Lógica, Pruebas, Confianza y Firma Digital



Lógica

Además de ontologías se precisan también reglas de inferencia. Una ontología puede expresar la regla "Si un código de ciudad está asociado a un código de estado, y si una dirección es el código de ciudad, entonces esa dirección tiene el código de estado asociado". De esta forma, un programa podría deducir que una dirección de la Universidad Complutense, al estar en la ciudad de Madrid, debe estar situada en España, y debería por lo tanto estar formateado según los estándares españoles. El ordenador no "entiende" nada de lo que está procesando, pero puede manipular los términos de modo mucho mas eficiente beneficiando la inteligibilidad humana.

Pruebas (Proof)

Será necesario el intercambio de "pruebas" escritas en el lenguaje unificador (se trata del lenguaje que hace posible las inferencias lógicas hecha posibles a través del uso

de reglas de inferencia tal como es especificado por las ontologías) de la Web Semántica.

Confianza (Trust)

Los agentes deberían ser muy escépticos acerca de lo que leen en la Web Semántica hasta que hayan podido comprobar de forma exhaustiva las fuentes de información. (Web Of Trust RDF Ontology -WOT[41] y FOAF[42])

Firma Digital (Digital Signature)

Bloque encriptado de datos que serán utilizados por los ordenadores y los agentes para verificar que la información adjunta ha sido ofrecida por una fuente específica confiable. (XML Signature WG[43]).