UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Agente Recomendador en Jason

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Victor Gascó Ortiz

Tutor: Vicente Javier Julián Inglada

2013/2014

Resumen

Los sistemas de recomendación tradicionales basan sus recomendaciones en las medidas cuantitativas de la similitud entre las preferencias del usuario y los elementos actuales para recomendar (p.e., recomendadores basados en contenidos [1]), entre el perfil del usuario y el perfil de otros usuarios con preferencias similares (p.e., recomendadores de filtrado colaborativo [2]) y en combinaciones de ambos (p.e. recomendadores híbridos [3]).

Sin embargo, [4], se ha quedado patente la incapacidad de los sistemas de recomendación actuales para utilizar la gran cantidad de datos cualitativos disponibles en línea para mejorar las recomendaciones.

Por lo general, los sistemas de recomendación no proporcionan una explicación sobre el proceso de razonamiento que se ha seguido para llegar a recomendaciones específicas. Las recomendaciones tienden a venir directamente del algoritmo de recomendación que ejecuta el sitio web y no a partir de los conocidos que un usuario tiene en su red social.

Con el fin de mejorar los resultados de cualquier tipo de recomendación basado en nuestros gustos, en este proyecto se ha añadido el factor de recomendación según el entorno social.

Debido al auge en los últimos años de las redes sociales y de cualquier tipo de medio en el que se interactúe con personas, no es difícil medir el nivel de relación que se tiene con otras personas de esa red social, y por lo tanto suponer que cuanto más afín se es a una persona se debe la similitud de gustos. Partiendo de esa base se ha simulado un entorno social en el que un cliente que realiza una solicitud de recomendación, en este caso sobre recetas culinarias. El entorno posteriormente le devuelve una lista con los resultados que mejor se adapta a su perfil y a su relación con los amigos que le hacen estas recomendaciones.

Para ello, nos basamos en resultados objetivos, como es la similitud que las recetas tengan con el perfil del cliente, y en valores subjetivos, que serían la ponderación de la relación de amistad entre el cliente con los amigos que le recomienden, y los que les recomienden a estos.

Para la realización de este trabajo se ha creado un marco de desarrollo de recomendaciones sociales basadas en agentes en el dominio de las recetas de cocina. El marco permite evaluar distintas estrategias a aplicar en el proceso de recomendación. Su uso también podría extenderse al ámbito académico como simulador de distintas estrategias de recomendación social.

Este procedimiento de recomendación sería válido para cualquier temática (películas, viajes, deportes...), útil como aplicación a las redes sociales e incluso para crear portales web basados en una temática para recomendaciones con tus conocidos.

En el proyecto se ha realizado una evaluación de la propuesta. Los resultados han permitido demostrar la validez del marco desarrollado como banco de pruebas de diferentes estrategias.

Palabras clave: sistema multi-agente, sistema de recomendación, redes sociales.

Abstract

Traditional recommender systems base their recommendations on quantitative measures of similarity between the user's preferences and the current items to recommend (i.e, content-based recommenders [1]), between the user's profile and the profile of other users with similar preferences (i.e, collaborative filtering recommender [2]) and on combinations of both (i.e, hybrid recommender [3]).

However, [4] has stated the inability of current recommender systems to use the large amount of qualitative data available online to empower the recommendations.

Usually, recommender systems do not provide an explanation about the reasoning process that was followed to come up with specific recommendations. Recommendations tend to come directly from the recommendation algorithm that runs the website and not from the acquaintances that a user has in his social network.

In order to improve the results of any type of recommendations based on our tastes, on this project has been added the recommendation according to the social environment factor.

Due to the rise in recent years of social networks and any type of media in which to interact with people, it is not difficult to measure the level of relationship that you have with other people in the social network, and therefore assume that you are much more akin to a person is because the similarity of tastes. On this basis has been simulated a social environment in which a customer who makes a request for a recommendation, in this case about culinary recipes. The environment then returns a list with the results that best fits your profile and your relationship with friends that make these recommendations.

To do this, we are based in objective results, as it is the similarity recipes with the customer profile, and in subjective values, which would be the weighting of the relation of friendship between the client with their friends who recommend him, and those that they recommend to these.

To carry out this work has created a development framework of social recommendations based on agents in the domain of the recipes. The framework allows you to evaluate different strategies to implement the recommendation process. Their use could also be extended to academic scope as a simulator of different strategies of social recommendation.

This recommendation procedure would be valid for any topic (films, travel, sports...), useful as application to social networks and even to create web sites based on a theme for recommendations with your friends.

In the project carried out an evaluation of the proposal. The results have allowed to demonstrate the validity of the framework developed as test bench of different strategies.

Keywords: multi-agent system, recommendation system, social network.

Tabla de contenidos

1. Introducción

En los últimos años, la aparición de las redes sociales ha cambiado las principales actividades realizadas por los usuarios en Internet, pasando de una mera búsqueda y la navegación por la información almacenada a una interacción directa con otros usuarios. Los usuarios han pasado de ser consumidores de información a ser los productores reales (lo que se conoce como la transición de la Web 1.0 a la Web 2.0).

Debido al creciente número de usuarios y la información que se genera, su comportamiento impredecible y el alto dinamismo de la estructura de red heterogénea, los usuarios tienen que hacer frente a un alto grado de incertidumbre a la hora de elegir con quién interactuar o qué información utilizar [5]. Con el fin de hacer frente a esta incertidumbre, los usuarios requieren de herramientas que les ayuden a tomar decisiones con respecto a sus actividades dentro de la red. Los sistemas de recomendación [6] [7], son sistemas que proporcionan recomendaciones efectivas sobre las acciones que los usuarios deben realizar o la información que pueden utilizar. En definitiva, son instrumentos eficaces para la realización de tareas de soporte de decisiones.

Por otra parte, la gente confía en las recomendaciones cuando el motor de recomendación puede proporcionar razones para ello [8]. Esto es por lo que se entiende que una buena evaluación es aquella que, en lugar de minimizar algún error de evaluación, hace a la gente más feliz.

Además, los sistemas de recomendación en línea sufren de problemas inherentes a su uso en redes sociales complejas, donde el número de usuarios y/o artículos para recomendar puede ser muy alto. En el caso del filtrado colaborativo, por ejemplo, el proceso para la comparación de dos usuarios con el fin de extraer su similitud requiere tener clasificados los mismos objetos, situación que puede ser irrealista en grandes redes sociales.

Otra de las principales debilidades de los sistemas de recomendación en línea es su confiabilidad. En una red abierta con un gran número de usuarios, es imposible asegurar que todos los puntos de vista expresados son verdaderas opiniones de los usuarios y no hay alteración de las recomendaciones resultantes. Para superar estos problemas, es necesario integrar una capa social en los enfoques actuales de recomendación, teniendo en cuenta aspectos como la generación de argumentos que apoyan las recomendaciones, la reputación y la confianza.

El presente proyecto pretende desarrollar un sistema de recomendación para una red social basado en agentes inteligentes, apoyándose en las relaciones de los usuarios y cómo estos interactúan entre ellos.

Se pretende observar y mostrar el impacto que estas relaciones pueden tener en las recomendaciones que a un cliente se le pueden dar, es decir, partiremos del hecho de que la gente interactúa más con personas de gustos similares a los suyos. Basándonos en esto se puede añadir una valoración extra al proceso de recomendación, más allá de los utilizados normalmente basándose exclusivamente en el perfil del cliente.

Para mostrar esto y poder aplicarlo, se ha montado una red social de recetas, utilizando agentes BDI, agentes racionales con un conjunto de actitudes mentales (Creencias, Deseos e Intenciones) y que toman sus acciones en función de sus estados, aproximación muy apropiada para agentes en entornos reales complejos y dinámicos. Con esto, se simula una solicitud de un cliente para obtener una recomendación en base a su perfil y a sus contactos, pudiendo variar la importancia que le damos a la información extraída del perfil, y a la que aporta su relación con otros usuarios. Con todo ello es posible mostrar como varía una recomendación.

En el proyecto hemos establecido unas formas de valoración básicas, pero añadiendo más información, como podría ser: ingredientes favoritos, problemas con ciertos alimentos, e incluso añadiendo un sistema de retroalimentación, mediante el cual el usuario valora las recomendaciones realizadas, se puede dar pié a que una red social con muchísima información de sus usuarios y de las relaciones entre estos, como puede ser facebook, añada un sistema de recomendación para sus usuarios.

La presente memoria se estructura en el siguiente conjunto de capítulos:

- En el capítulo 2 se tratan los objetivos, los cuales contienen apartados a desarrollar en el proyecto para conseguir una versión operativa, introduciendo un objetivo principal y desglosándolo en subobjetivos.
- En el capítulo 3 se habla de las tecnologías utilizadas para el desarrollo del proyecto (agentes BDI, Jason…).
- En el capítulo 4 se muestra el diseño de la herramienta, se comentan los diferentes agentes BDI implementados, métodos de comunicación, acciones internas…
- En el capítulo 5 se realizan y comentan, pruebas llevadas a cabo en el marco desarrollado.
- Finalmente, en el capítulo 6 se muestran las conclusiones, comentando además la posibilidad de trabajos futuros basándose en este proyecto y en sus resultados.

2. Objetivos

El objetivo de este Proyecto de Fin de Grado es el de realizar un marco de recomendación social, basado en agentes inteligentes, y su posterior evaluación y validación mediante diferentes experimentos.

Este sistema de recomendación social requiere de una base de conocimiento que contenga las recetas almacenadas con los gustos de los clientes junto con un sistema multiagente que refleje el comportamiento de una red social en el dominio de las recetas de cocina.

El objetivo principal podemos dividirlo en subobjetivos, facilitando el proceso de desarrollo y la organización del proyecto, además de permitir controlar mejor su temporalización. Estos subobjetivos son:

- Estudio de la estructura de las recetas en la base de datos y determinar qué información de ésta es relevante para el proceso de recomendación y clasificación.

- Implantación de la base de datos con las recetas y determinar el acceso más adecuado a la información para su posterior evaluación.

- Estudio del estado del arte sobre los agentes BDI, aprender a usar correctamente esta potente herramienta. En concreto se hará especial hincapié en la comunicación entre agentes y envío de mensajes.

- Modelado del sistema multiagente, diseño de las creencias internas de cada agente simulando ser un amigo de nuestra red social así como de los protocolos de comunicación necesarios.

- Diseño de la red social basada en un modelo de confianza entre los agentes, favoreciendo así las recomendaciones recibidas desde amigos más fiables.

- Implementación final de la herramienta, implantación de la red de amigos en el entorno final y un agente cliente que simule la solicitud de un cliente real, mostrando y valorando así los resultados obtenidos.

- Evaluación de las estrategias de recomendación utilizadas y conclusiones respecto a la importancia de haber añadido una valoración social al proceso de recomendación.

3. Estado del Arte

En el contexto de este proyecto es imposible no hablar de los agentes BDI, estos agentes reciben el nombre (BDI) por su estructura interna compuesta por creencias, deseos e intenciones (Beliefs, Desires, Intentions). Es una metodología para el diseño de Sistemas Multi-Agente, propuesta inicialmente por Michael E. Bratman en 1987, perfecta para recrear la red social de amigos.

Los agentes BDI están implementados utilizando una arquitectura deliberativa que utiliza modelos de representación simbólica del conocimiento, es decir, los agentes parten de un estado inicial y son capaces de generar planes para alcanzar sus objetivos. Por tanto, podemos decir que un agente deliberativo es aquel que contiene un modelo simbólico del mundo, en donde las decisiones se toman utilizando mecanismos de razonamiento lógico con el propósito de alcanzar los objetivos del agente [9].

A la hora de implementar una arquitectura deliberativa hay que realizar e integrar en el agente una descripción simbólica adecuada del problema, para el correcto funcionamiento de este. Aunque pueda parecer una cuestión trivial, es un aspecto al que hay que prestar mucha atención, especialmente si se tiene en cuenta que los agentes se desenvuelven en dominios reales, en los que frecuentemente tienen que responder a los estímulos en tiempo real.

Los agentes intencionales se puede implementar utilizando arquitecturas deliberativas. Estos agentes están dotados de modelos de planificación capaces de generar planes a partir de las creencias e intenciones. La arquitectura deliberativa BDI es probablemente la más estudiada y posiblemente la más extendida.

3.1. Arquitectura BDI

Esta arquitectura está caracterizada por el hecho de que los agentes que la componen tienen conocimiento en forma de Creencias, Deseos e Intenciones. Tiene un modelo filosófico del razonamiento humano fácil de comprender y una semántica lógica abstracta, aceptada por gran parte de la comunidad científica.

Los agentes BDI incorporan componentes que permiten el desarrollo de sistemas que se integren adecuadamente en el mundo real. Así como la mayoría de aplicaciones trabajan con información exacta, los agentes BDI son capaces de relacionarse con un sistema más complejo, como sería un sistema con un entorno cambiante y con cierto grado de incertidumbre.

Este modelo de agentes BDI proporciona soluciones en entornos en los que el agente sólo tiene una visión parcial del problema. Las creencias, las intenciones y los planes son una parte fundamental del estado de ese tipo de sistemas.

El funcionamiento de este tipo de agentes es el siguiente:

Perciben, o bien del entorno mediante sensores, mediante información que le llega de este, o bien a través de mensajes de otros agentes, modificando así los conocimientos de éste, es decir, sus Creencias.

El agente tiene que tomar decisiones para interactuar con este entorno, las cuales están basadas en sus Creencias y gobernadas por sus Deseos, a fin de cumplirlos, como objetivos que ha de alcanzar.

Por último, con el fin de conseguir esos objetivos el agente se vale de una seria de Intenciones, acciones que el agente va realizando u objetivos elegidos, pudiendo ser abortadas en cualquier momento dependiendo de las creencias del agente.

La arquitectura BDI tiene sus raíces en el denominado Razonamiento Práctico [10]. Este tipo de razonamiento puede describirse como el proceso de decidir qué acción realizar para alcanzar las metas, e involucra dos importantes procesos: Deliberación, decidir qué metas se quieren alcanzar, y Razonamiento de medios-fines, decidir cómo se alcanzarán las metas.

Una vez el agente adopta una intención, ésta afectará al razonamiento práctico que considere en el futuro. El agente no deberá abandonar sus intenciones hasta que las cumpla, o bien sea evidente que no podrá cumplirlas, o bien las razones que tuvo para adoptarla dejaron de ser válidas.

El modelo básico de la arquitectura BDI propone el siguiente esquema de comportamiento para los agentes (ver Figura 1):

- Percibir los cambios del entorno en el que se desenvuelve.
- Revisar sus creencias acerca del mundo en base a su percepción.
- Razonar acerca de sus intenciones a fin de reconsiderarlas en caso de que sea necesario.
- Seleccionar una acción a seguir, razonando sobre sus creencias y sus intenciones.
- Ejecutar la acción seleccionada, la cual producirá cambios en el entorno, y volver al paso 1 a fin de percibir los cambios que se han producido.

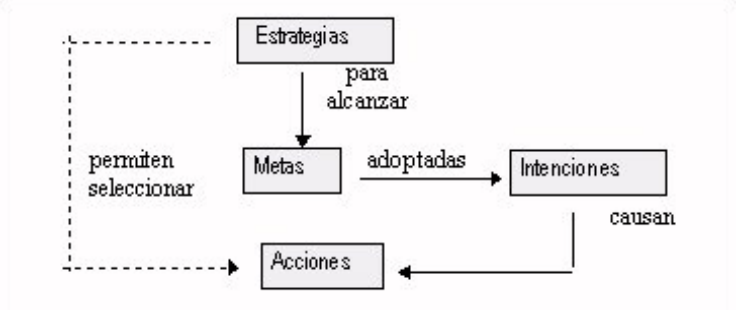

Figura 1: Esquema de la arquitectura BDI básica

Las creencias representan el conocimiento que el agente tiene del entorno. Son la forma en la que representamos el estado del entorno. En medios dinámicos es necesario mantener la información sobre eventos pasados, pero al mismo tiempo se debe permitir su adaptación y evolución.

Los deseos u objetivos, en términos informáticos, pueden simplemente ser el valor de una variable, un registro, o una expresión simbólica en alguna lógica. Representa un estado final deseado. Normalmente el software está "orientado a la tarea" en lugar de "al objetivo", de forma que cada tarea se ejecuta sin ningún recuerdo de por qué ha comenzado, esto significa que el sistema no puede recuperarse de fallos automáticamente y no puede aprovechar oportunidades que surjan inesperadamente. Por ejemplo, la razón de que un humano sea capaz de superar un pinchazo inesperado de la rueda de su coche es porque conoce dónde está (a través de sus creencias) y recuerda qué quiere conseguir (a través de sus objetivos).

Para alcanzar estos objetivos es necesario definir un mecanismo que nos permita identificar las intenciones. Tenemos que tener claro que los agentes están en sistemas dinámicos en los que ocasionalmente tendrán que decidir durante la ejecución del plan inicial, ante cambios del entorno, si se replanifica o no. El sistema necesita comprometerse con los planes y subobjetivos, pero también ser capaz de reconsiderar éstos en los momentos clave. Estos planes vinculados a la consecución de un objetivo constituyen las intenciones del agente, que son un conjunto de caminos de ejecución que pueden ser interrumpidos de una forma apropiada al recibir información acerca de cambios en el entorno.

3.2. Formalismo teórico del modelo BDI

Las actitudes se representan en un formalismo basado en la lógica modal y el concepto de "mundos posibles". La lógica modal se puede definir como aquella que permite razonar sobre lo que podría ser, en lugar de lo que es realmente.

Para conseguir que la lógica incluya estos significados se maneja una semántica de mundos posibles con una relación de accesibilidad entre ellos. La relación de accesibilidad enlaza la situación actual con todas las que son posibles a partir de ella. De este modo los conceptos de verdad necesaria y verdad posible se definen a partir de esta relación de accesibilidad. Una proposición es necesariamente verdadera si lo es en todos los mundos que son accesibles desde él, y una proposición es posiblemente verdadera en un mundo si lo es en alguno de los mundos accesibles desde él.

Los agentes BDI se pueden modelar utilizando una estructura basada en la lógica de mundos posibles, denominada árbol temporal con múltiples futuros y un solo pasado (ver Figura 2).

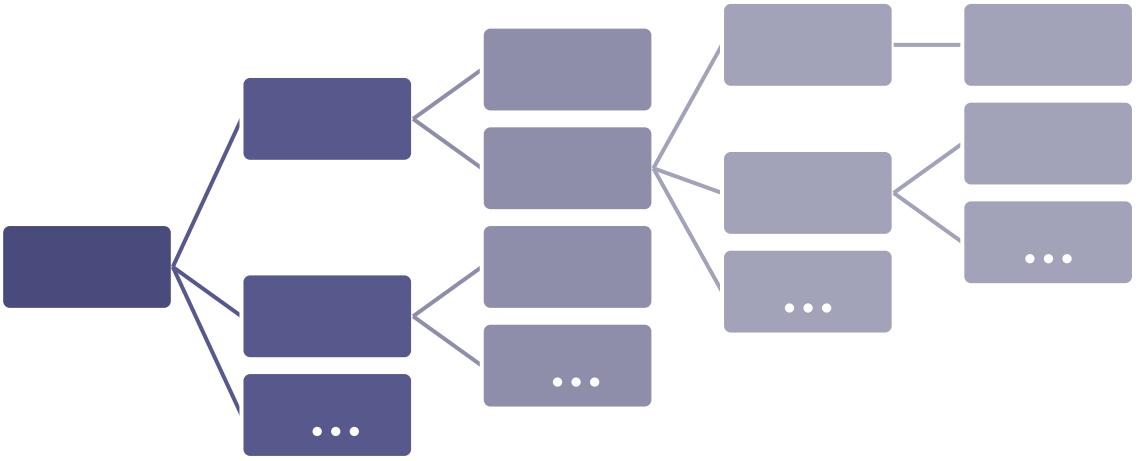

Figura 2: Árbol de mundos posibles

Cada nodo del árbol es una situación y las ramas del árbol pueden verse como las opciones disponibles para el agente en cada momento del tiempo. Para cada situación, se definen una serie de mundos accesibles desde el punto de vista de las creencias (mundos posibles), de los deseos (mundos que el agente desea alcanzar) y de las intenciones (mundos que el agente ha decidido que intentará alcanzar). El agente debe creer que esos deseos sean alcanzables.

Podría decirse que el agente, situado en un mundo accesible por sus creencias, cambia a un mundo accesible por objetivos. De ese estado evoluciona a otro accesible por medio de las intenciones al comprometerse a realizar las acciones deseadas.

Basándose en esto, se definen las relaciones que deben existir entre las creencias, los deseos y las intenciones del agente:

- Compatibilidad entre creencias y objetivos. Si el agente quiere alcanzar un objetivo, debe creer que en algún mundo accesible con esta creencia, dicho objetivo es cierto.
- Compatibilidad entre objetivos e intenciones. Antes de adoptar una intención debe de haber un deseo.
- Las intenciones conducen a acciones.
- Relación entre creencias e intenciones. El agente cree en sus propias intenciones.
- Relación entre creencias y objetivos. El agente conoce sus objetivos.
- Cuando un agente adopta una intención, sigue con ella hasta algún momento del futuro, no puede haber una parada infinita en el proceso de alcance de un determinado objetivo.

El agente no tiene control directo sobre sus creencias y deseos, por lo tanto se restringe la condición de compromiso sobre las intenciones. Un agente puede comprometerse basándose en que el objetivo de una intención es satisfecho en un camino futuro. De esta forma, las intenciones actuales del agente guían o influyen en sus decisiones sobre futuras intenciones. Dependiendo de cómo afectan las intenciones, se identifican varios tipos de agentes:

- Ciego. Mantiene sus intenciones hasta alcanzarlas. Si es necesario rechazará las creencias o deseos que contradigan esos compromisos.
- Firme. Mantiene sus intenciones mientras crea que tiene opciones de alcanzarlas.
- Imparcial. Mantiene sus intenciones mientras que el deseo o deseos que dieron lugar a esa intención no cambien.

Todos estos aspectos descritos anteriormente han sido formalizados empleando elementos de la lógica modal. El propósito de esta formalización es construir sistemas prácticos y verificables. Si para un dominio de aplicación se conocen los cambios del entorno y los comportamientos esperados del sistema, se puede usar esta formalización para especificar, diseñar y verificar agentes, que en su entorno exhibirán los comportamientos deseados.

De este modo se define un mundo de planes, aunque no es necesaria una relación de accesibilidad entre mundos, puesto que los planes son estáticos. Por otro lado, debe existir una interrelación entre las intenciones de un agente y los planes de que dispone. Las relaciones formalmente definidas más importantes son las citadas a continuación:

- Las intenciones de un agente están restringidas por sus planes, es decir, las acciones se deben alcanzar exclusivamente utilizando los planes de la librería de planes del agente.
- Si un agente tiene la intención de alcanzar un objetivo, debe adoptar la intención de ejecutar el cuerpo del plan que permita alcanzarlo.
- Se espera que si un agente tiene un plan para alcanzar un objetivo, dentro de sus creencias esté el hecho de que el plan logrará el objetivo.

3.3. Arquitectura Abstracta

La arquitectura propuesta se basa en la construcción de sistemas de agentes cuyas estructuras de datos se corresponden con cada uno de los componentes de los agentes BDI: creencias, deseos e intenciones (ver Figura 3). En esta estructura, cada uno de los componentes se guarda por separado. Además, se trabaja con una secuencia de eventos ocurridos en el entorno, implementado como una cola con comportamiento FIFO (primero en entrar primero en salir).

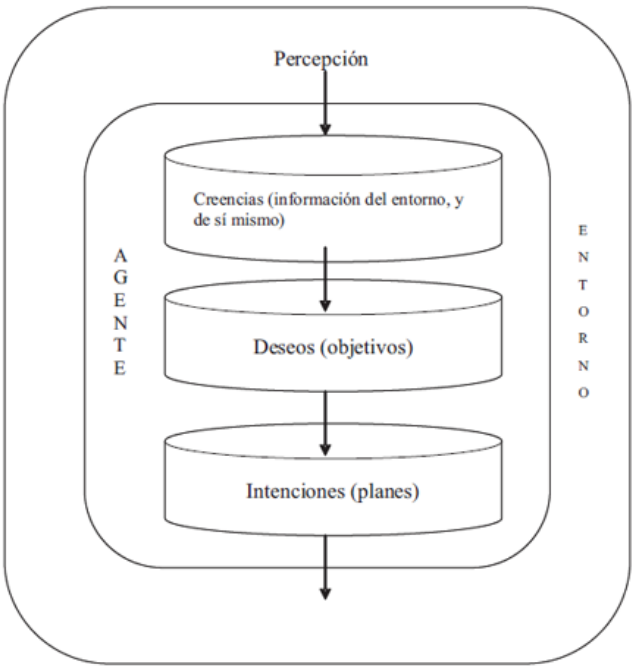

Figura 3: Arquitectura BDI

Para la utilización de estos datos la implementación no puede estar basada en métodos tradicionales como la demostración de teoremas. Aunque el método pudiese ser ampliado para dar cobertura a todos los aspectos lógico-formales antes descritos, no podría determinarse el límite máximo para el tiempo de computación, lo que comprometería la supervivencia del agente.

Para solventar esto, la arquitectura realiza una serie de pasos, cada uno de ellos con una duración limitada y por tanto, con posibilidad de reacción ante el entorno.

Al comienzo del ciclo, se lee la cola de eventos y se devuelve la lista de opciones. Se seleccionan aquellas que se deben adoptar y se añaden a la cola de intenciones. A continuación, se ejecutan todas las intenciones que impliquen la realización de una acción simple. En este momento se comprueba si existen nuevos eventos en el entorno y se incorporan a la cola de eventos. Por último el agente modifica las estructuras de deseo e intención, eliminando los ya satisfechos y los que son imposibles de alcanzar. Un esquema de este proceso, sería:

```
Interprete-BDI
Inicializar_estado();
Repetir
      Opciones:= generador_opciones (cola_eventos);
      Opciones_seleccionadas := decidir (opciones);
      Actualizar_intenciones (opciones_seleccionadas);
      Ejecutar();
      Leer_nuevos_eventos_externos();
      Eliminar_intenciones_alcanzadas();
      Eliminar_intenciones_imposibles();
Fin repetir
```

Esta arquitectura es una idealización que representa los conceptos teóricos, aunque a niveles prácticos, por ejemplo, no se hace referencia a cómo desarrollar los procesos de generación o de selección de opciones de forma suficientemente eficiente.

Por ello, a la hora de la implementación se han introducido algunas limitaciones para mejorar el modo de razonamiento que restringe el poder expresivo de la arquitectura. Por ejemplo, al definir las creencias hay que evitar el uso de disyunciones o implicaciones. Además los agentes suelen incorporar una librería de planes ya definidos.

Finalmente hacer hincapié en la introducido realizada en este capítulo sobre el concepto de agente BDI, el cual será la base para el desarrollo de nuestra propuesta, la cual se detalla en el siguiente capítulo.

4. Diseño de la herramienta

En este proyecto se va a desarrollar una herramienta cuya principal funcionalidad es la utilidad de agregar a un sistema de recomendación un marco social, propiciando de esta forma una mejor valoración por parte de la herramienta sobre las recomendaciones que se propondrán al usuario. Según requisitos del propio proyecto, como soporte para la recreación del marco social se va a utilizar el intérprete Jason [11] para agentes con arquitectura BDI, lo que por otra parte es muy adecuado en este tipo de sistemas, tal y como ha comentado en el capítulo anterior.

El intérprete Jason implementa la semántica operacional de los lenguajes *AgentSpeak,* y facilita una plataforma para el desarrollo de los sistemas multi-agentes perfecta y requerida para este proyecto.

Por ello además del propio lenguaje del intérprete Jason, también se ha desarrollado parte de la herramienta en Java, fundamentalmente a lo que se refiere a la introducción con el entorno, ya que el propio sistema está implementado en Java.

4.1. Tipos de agentes BDI

Una vez comentados los dos lenguajes de programación a utilizar en el desarrollo de la herramienta vamos a pasar a comentar los agentes implicados en ella.

Encontramos 3 tipos de agentes BDI:

- Cliente, este agente representa al usuario que en el entorno social haría la consulta para recibir recomendaciones de sus amigos.

- Amigos, como su nombre indica son la representación de los amigos que el Cliente tendría en su red social, y a su vez, los amigos de amigos, creando así una red social bastante amplia y con varias capas de profundidad. Estos Amigos tienen una base de conocimiento con sus recetas favoritas, entre las cuales buscan las que recomendar según la solicitud que se les envíe.

- System, este agente es el encargado de inicializar todo, el encargado de simular el entorno en el que se encontrarían el Cliente y Amigos en la realidad. Inicia las bases de datos de cada Amigo y una vez hecho esto, le indica al Cliente que puede hacer su solicitud, como ya se ha dicho, una vez ha simulado el entorno social.

Para crear estos agentes BDI, es necesario crear una base de datos con las recetas almacenadas y poder brindar a estos agentes la base de conocimientos. Por otro lado se ha de ser capaz de crear las comunicaciones y acciones internas necesarias en el interprete Jason, tal y como vamos a contar a continuación.

4.2. Conocimiento necesario.

Como se ha comentado, fue necesario ampliar los conocimientos sobre la programación de los agentes BDI a fin de programarlos con el intérprete Jason.

Además de lo ya conocido sobre las Creencias, Deseos e Intenciones y como crearlas, parte fundamental era el paso de mensajes entre agentes y que se enviasen creencias que desencadenasen en intenciones, como ejemplos de este paso de mensajes están la solicitud del cliente a sus amigos, y la de cada amigo a sus respectivos amigos. Esto en cuanto a la adquisición de conocimiento necesaria en el lenguaje Jason.

Para montar la base de conocimiento de los agentes se ha utilizado MongoDB [12], base de datos *open-source*, rápida, con accesos sencillos, y que forma parte de las bases de datos no SQL.

En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Las principales características de MongoDB son:

- Consultas Ad hoc. Búsqueda por campos, consultas de rangos y expresiones regulares.
- Indexación. Cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios.
- Replicación. MongoDB soporta el tipo de replicación maestro-esclavo. El maestro puede ejecutar comandos de lectura y escritura. El esclavo puede copiar los datos del maestro y sólo se puede usar para lectura o para copia de seguridad.
- Balanceo de carga. El desarrollador elige una llave shard [13], la cual determina cómo serán distribuidos los datos en una colección. Los datos son divididos en rangos y distribuidos a través de múltiples shard.
- Almacenamiento de archivos. Tomando la ventaja de la capacidad que tiene MongoDB para el balanceo de carga y la replicación de datos utilizando múltiples servidores para el almacenamiento de archivos.
- Agregación. La función MapReduce permite que los usuarios puedan obtener el tipo de resultado que se obtiene cuando se utiliza el comando SQL “group-by”.

- Ejecución de JavaScript del lado del servidor. MondoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

A la hora de acceder a la información de cada una de las recetas hemos de tener en cuenta la estructura que éstas tienen (para más información sobre las estructura general consultar el Anexo 1 al final del documento).

A continuación se muestra una receta perteneciente a la base de datos:

{ "_id" : { "$oid" : "51c1d81f427647ec305b10eb" },
"origin" : "quecocinohoy",
"nutrition" : {
"cautions" : [],
"healthLabels" : ["VEGETARIAN", "GLUTEN_FREE",
"WHEAT_FREE", "SUGAR_CONSCIOUS"],
"calories" : 1176,
"uri" :
"http://www.edamam.com/ontologies/edamam.owl#crujiente_de_queso_
878b648ef6f9aaabd2424db8f1014e23",
"yield" : 4,
"dietLabels" : ["LOW_CARB"] },
"ingredients" : [{
"name_en" : "grated Parmesan cheese",
"amt" : { "unit_en" : "Gr.", "unit" : "gr", "qty" : "300" },
"name" : "queso parmesano rallado " }],
"url" : "http://www.quecocinohoy.com/receta/crujiente-de-queso",
"title" : "Crujiente de queso",
"yield" : "4",
"recipe_id" : 2,
"observations" : {},
"images" : [],
"tips" : {
"Tiempo de preparación" : "Menos de 15 minutos",
"Dificultad" : "Intermedia",
"Tipo de cocina" : "Mediterránea",
"Temporada" : "-" },
"instructions" : {} }

La receta que tenemos arriba es una de las más pequeñas de la Base de Datos, como se puede observar tiene varios campos vacios, sin embargo podemos comprobar que campos contiene la receta y cómo se almacenan estos datos.

4.3. Descripción de agentes del sistema

Una vez comentado el conocimiento necesario para entender y montar el proyecto, vamos a comentar las acciones que realizan los agentes involucrados y explicar cómo interactúan entre ellos.

En primer lugar hay que tener en mente una imagen de la estructura del sistema, agente BDI '*System*', el cual inicia todo y en qué orden se produce la comunicación (Figura 4).

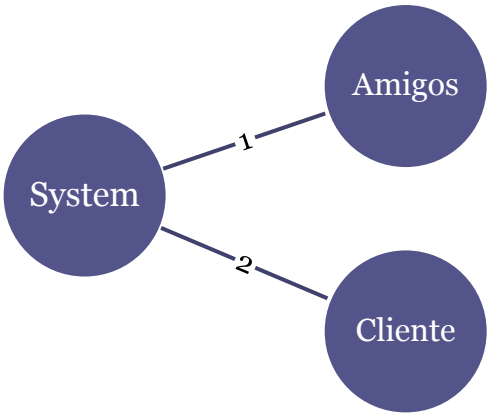

Figura 4: Accionamiento del sistema.

Como se observa en la Figura 1, el agente System es el eje central que controla la ejecución de los demás agentes. Su función es la de crear e iniciar ese entorno social simulado en el que vamos a trabajar, hace las llamadas al entorno y para que este acceda a la Base de Datos (BD), y que a cada agente Amigo se le indexen su lista de Amigos y su BD de recetas, esto es: sus recetas favoritas. Al mismo tiempo el agente Cliente, el agente que va a realizar la consulta, se le indexa su lista de amigos.

```
.println("INDEXANDO RECETAS DE AMIGOS. . .");
iniciarBDamigos(slot);
.println("INDEXANDO RELACIONES DE AMIGOS. . .");
obtenerAmigos(slot);

.wait(5000);
.println("");
.println("ENVIANDO EL REQUEST DEL CLIENTE...");
.send(cliente,achieve,request);
```

Figura 5: Fragmento de código agente System.asl

Como se observa en el código del agente System (Figura 5), primero se inicia la BD de los agentes Amigo con *'iniciarBDamigos(slot)'*, luego se indexan las listas de amigos con '*obtenerAmigos(slot)'*, y tras 5 segundos, espera necesaria para que se inicialice todo, se envía un mensaje al cliente agregándole el objetivo de hacer una solicitud de recomendación, '*send(cliente,achieve,request)'*. Esto inicia el algoritmo de recomendación en nuestro marco social.

A continuación vamos a hablar del entramado que forman las relaciones sociales del sistema. Este está formado simulando la red de amistades que podemos encontrar en cualquier red social. Como podemos ver en la Figura 6, el entramado social es similar al de un árbol de jerarquía, en el que el cliente es el solicitante principal y cada amigo tiene, a su vez, sus propios amigos.

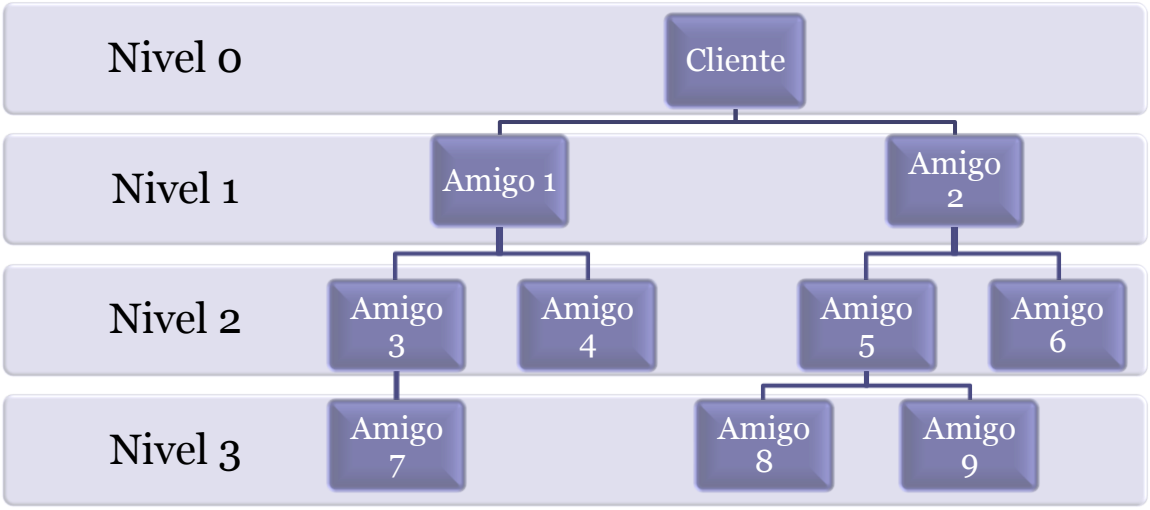

Figura 6: Estructura básica de las relaciones

Se observan los niveles de profundidad según el amigo que estemos viendo, es necesario tenerlo en cuenta en el algoritmo de búsqueda, una búsqueda con mayor profundidad puede ser más precisa pero más costosa.

Sin embargo, enseguida podemos darnos cuenta de que ésta no es una representación social aceptable, en una red social no es difícil observar que la gente tiene amigos comunes, por ejemplo el Amigo 4, perfectamente podría ser amigo de Amigo 2, y esto también habría que tenerlo en cuenta. Estas relaciones se observan en el siguiente esquema (Figura 7), partiendo del anterior se han añadidos algunas relaciones más.

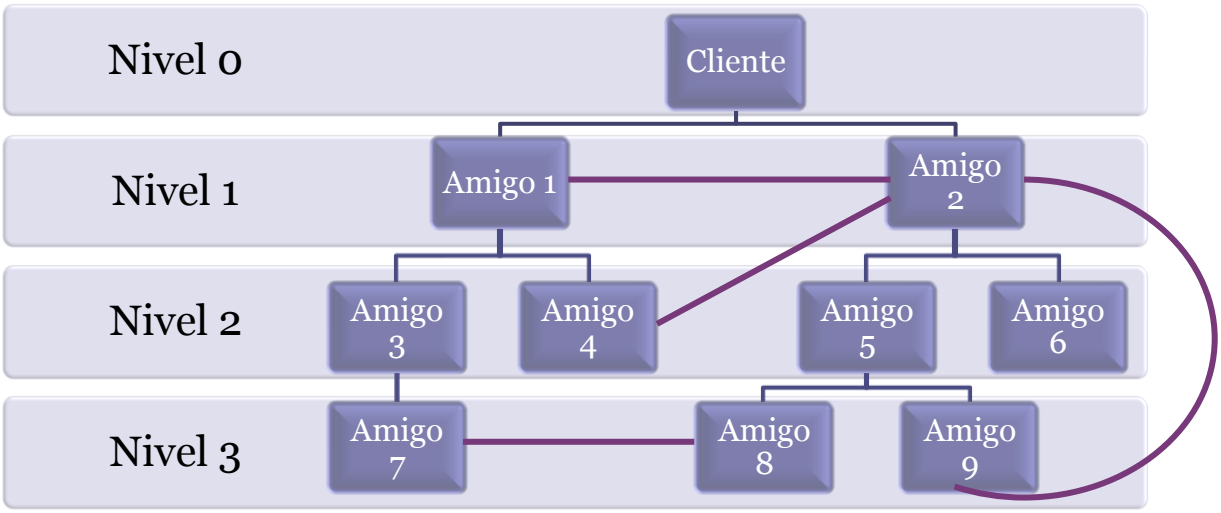

Figura 7: Estructura ampliada de las relaciones sociales

Se observa algunas relaciones sociales extra entre amigos, que han deformado el árbol, en estas relaciones (color morado), no se ha de tener en cuenta el nivel al que llegan, si no desde el que proceden.

Por ejemplo: Amigo 2 – Amigo 9, en esta relación el Amigo 9 sería consultado cuando el Amigo 2 le hiciera una consulta a sus amigos, por lo tanto el nivel de profundidad del Amigo 9, no sería 4, si no que sería 3 (nivel de Amigo 2 + 1), y estaría al mismo nivel que Amigo 5 y Amigo 6.

Una vez mostrado la estructura social que simula el sistema, hemos de ver cómo se propaga la información y se hace la valoración de cada receta. Para ello, primero se ha de observar como el agente Cliente realiza la solicitud a toda su lista de amigos, y posteriormente, como estos hacen lo propio con sus amigos.

4.4. Cálculo de recetas.

Tal y como se muestra en el apartado anterior, el agente System inicia el sistema y lanza el objetivo "*request*" al agente Cliente. Una vez hecho esto, el Cliente, ya inicializado, envía consultas a todos sus amigos.

En estas consultas se han de indicar dos valores: El nivel de profundidad actual, y el nivel de profundidad máximo (ver Figura 8). Gracias a esto, los amigos saben si han de contestar en base a sus conocimientos o si han de consultar también a sus amigos.

```
+!request: true
    <-  ?amigos(ListaA);
        .length(ListaA, Length);
        +bucle(0);
        while (bucle(I) & (I < Length)) {
            .nth(I, ListaA, A);
            ?prof(P);
            if(myLib.comprobar_amigo(A)){
                .send(A, tell, consulta(0,P));
            }
            else{/*...*/}
             -+bucle(I+1);
        }
        -bucle(_).
```

Figura 8: Fragmento de código agente Cliente.asl

Como se observa en la imagen, el bucle "*while*" recorre toda la lista de amigos del agente Cliente, comprueba si existen con "*myLib.comprobar_amigo(A)*", y les envía la creencia consulta, para que sepan que se les solicita una recomendación.

Este mensaje es el mismo que se envían los amigos entre sí, por ello el agente Cliente también envía el nivel de profundidad actual, aunque por supuesto siempre enviará nivel 0.

Una vez le llega el mensaje a los amigos, hacen lo propio y, si aun no están al máximo de profundidad especificado, le envían el mensaje consulta a sus amigos, pero esta vez ellos si envían la profundidad actual (ver Figura 9).

En este caso sí que variamos el nivel de profundidad que enviamos a la lista de amigos. Al agente le llega en el mensaje el nivel al que pertenece el agente que le envía el mensaje, aumenta en uno el nivel de profundidad y se lo envía a sus amigos.

Si por otro lado, estuviéramos al máximo nivel de profundidad, el agente tendría que enviar el listado de recetas que el mismo recomendase. El cálculo que hacen los agentes para realizar la recomendación se realiza mediante una acción interna, en código Java, la función que lo ordena es "*myLib.sortRecetas(...)*".

```
@p2[atomic]+consulta(X,MaxProf)[source(A)]
    <-
        /* ... */
        if (X+1 < MaxProf) {
            ?amigos(ListaA);
            .length(ListaA, Length);

            +bucle(0);
            while (bucle(I) & (I < Length)) {
                .nth(I, ListaA, F);
                if(myLib.comprobar_amigo(F)){
                    .send(F, tell, consulta(X+1,MaxProf));
                } else {/*...*/}
                    -+bucle(I+1);
                }
                -bucle(_);
        }
        if (X+1 == MaxProf){
            myLib.sortRecetas(RecetasIni,NomAmFinal," ",ListaA,ValorAmistad);
            ?listaRec(MDos);
            .send(A, tell, requestRecetas(MDos));
            }.
```

Figura 9: Fragmento de código agente Amigo.asl

Una vez se ordena la lista de recetas "*listaRec(..)*", se consulta la creencia y ésta es la que se devuelve al Amigo, o al propio cliente, que la solicitó. Siempre que un agente envía un mensaje de consulta le llega un mensaje de "*requestRecetas*(...)" por cada mensaje de consulta que haya enviado. Por ello vamos a comentar qué acciones realiza un agente en cuanto le llega "*requestRecetas*".

Como es de suponer, las acciones que realizarán el agente Cliente o cualquier agente Amigo variarán, puesto que el Cliente es el que en un principio hace la solicitud, y por lo tanto el no ha de enviar más mensajes para seguir escalando ese árbol social.

Primero vamos a comentar las acciones que realizará el agente Cliente. Cada vez que le llegue un "*requestRecetas*(...)", es decir, cada vez que le contesten a una solicitud, aumenta en uno un contador interno que tienen los agentes para saber cuando les han contestado a todas sus consultas. Una vez le llega todo el número de consultas que esperaba, mediante la acción interna "*myLib.sortRecetas(...)*" ordena todas las recetas que los agentes del nivel inferior, en este caso siempre serán desde el nivel 1, y muestra las *n* mejores. Por defecto el sistema muestra las 10 mejores.

A continuación se muestra el código de los agentes Amigo que trata estas respuestas a las consultas que les hayan llegado, en este caso no solo las ordenan, sino que además han de enviarlas a los que se las hayan solicitado (ver Figura 10).

```
@p1[atomic]+requestRecetas(RNuevas): true
    <-  /*...*/
        ?amigos(ListaA);
        .length(ListaA, I);
        ?numResp(M);
        -+numResp(M+1);
        if (I == M+1) {
            ?valorAmigos(ValorAmistad);
            myLib.sortRecetas(RTotales,NomAmFinal,NumRecFinal,ListaA,ValorAmistad);

            ?listaRec(RFinales);
            ?clientRequest(A);
            .send(A,tell,requestRecetas(RFinales));
            }.
```

Figura 10: Fragmento de código Amigo.asl

Como se observa, en cuanto el número de respuestas es igual al número de consultas, es decir, el número de amigos, se ordena la lista de recetas y posteriormente se envía. En el código se puede ver que el objetivo de ese mensaje se extrae de los conocimientos del agente, mediante "*clientRequest(A)*", esta creencia se añade en la primera consulta que le llega al agente, y por lo tanto, sólo se le contesta al primer agente que le consulta. Al resto de agentes que le hacen una consulta, no se les devuelve nada, puesto que la información que este agente podía contener ya se ha introducido en el sistema con la primera consulta que le hacen.

Una vez se ha mostrado cuando y como contestan los agentes a las consultas que les hacen, vamos a comentar cómo se valoran y ordenan las recetas que cada agente envía como respuesta.

Antes de valorar las recetas, hay que hacer una preselección de las que se corresponden a la solicitud del cliente, se hace un filtrado de sobre las recetas que el agente conoce, y se descartan las que carecen de total interés para la consulta. En nuestro marco social, estas son las que no contienen los ingredientes que el cliente requiere en su consulta.

La fórmula para valorar cada receta es la siguiente:

$$\alpha * valor\_social + \beta * valor\_propio$$

$$\alpha + \beta = 1$$

Los valores de α y β, son pesos que le damos a la parte social y a la parte objetiva de la valoración de la receta, pudiendo de esta forma probar a darle mayor o menor importancia al valor social. Por defecto esta puesto, α = 0.2 y β = 0.8.

En el capítulo 5, se comentará como repercute en la valoración final, el hecho de variar estos pesos.

Por otro lado, tenemos los que le da realmente la valoración a la receta, *valor_social* (u opinión) y *valor_propio*.

En primer lugar vamos a comentar *valor_social*, es decir, la opinión que tiene el amigo que va a valorar la receta, respecto a quién se la envió. Esta información no requiere ningún cálculo, simplemente se tiene almacenado todas las relaciones que los agentes de la aplicación tienen entre ellos, simplemente se extrae la que corresponde. Esta valoración está comprendida entre 0.1 y 1, dando por hecho que 0 sería una relación nula y, por tanto, no existente, y un valor de 1 representa una amistad ideal.

Sin embargo, esto es una representación para nuestro marco social, pero a la hora de aplicar esto a una red social real, se podría hacer de varias formas, o bien simplemente que el hecho de enviarse mensajes aumentase esa valoración de la relación entre amigos, o una solución más compleja, por ejemplo que se pudiesen analizar las comunicaciones que se realizarán entre ellos. Esto se comentará con mayor profundidad más adelante, en el capítulo 6: Conclusiones y trabajo futuro.

La otra parte de la fórmula para el valor de la receta es el valor objetivo de la propia receta, cómo es de buena según el perfil del solicitante, es decir, el agente Cliente. En nuestro marco social hacemos un cálculo sencillo, es una valoración simple de la similitud entre los ingredientes que el cliente solicita que tenga su receta y los ingredientes totales de esta, por tanto la fórmula es:

$$Ingredientes\_receta \,/\, ingredientes\_solicitados$$

Esto es una interpretación muy básica, como se muestra a continuación la estructura de una receta en esta base de datos tiene multitud de campos entre los que se podría hacer una búsqueda más precisa (ver Anexo 1), pero requeriría de un perfil del cliente mucho más completo, sobre esto se hablará en el capítulo 6: Conclusiones y trabajo futuro.

Esta valoración se realiza cada vez que un agente va a responder a una solicitud de recomendación, es decir, una vez que todos sus amigos le han contestado a él. Como se ha comentado anteriormente, el método encargado de realizar esto esta implementado en java, como una acción interna de los agentes Jason, "*myLib.sortRecetas(..)*". Esta función, además de valorar las recetas que cada agente tiene en sus conocimientos y que se corresponden con los valores que el cliente ha solicitado, las ordena y devuelve las 10 mejores.

Es decir, consulta las recetas que el agente ha adquirido, las propias y las que sus amigos le han recomendado, en base a los requisitos de la consulta.

Posteriormente pasa a valorar cada receta y las introduce en un *ArrayList valorRecetas* como objetos *receta*, que contienen el identificador y la valoración.

Finalmente se ordenan y se devuelven las 10 mejores de la lista ordenada según su valoración, calculada tal y como se ha comentado anteriormente.

4.5. Base de datos, entorno y acciones internas.

En este apartado vamos a centrarnos en comentar la parte del desarrollo de la herramienta que no se realiza en lenguaje Jason para agentes BDI.

Primero se va a comentar brevemente como está montada la base de datos (BD). Como se ha comentado en el capítulo anterior, la BD está montada con mongoDB, a partir de un fichero Json, tal y como se muestra en su web [12]. El fichero ha sido proporcionado por el GTI-IA (Grupo de Tecnología Informática e Inteligencia Artificial) del DSIC (Departamento de Sistemas Informáticos y Computación) de la Universidad Politécnica de Valencia. En él hay un total de 9020 recetas, cuya estructura se puede ver en el Anexo 1, extraídas de diferentes páginas web.

En segundo lugar encontramos el entorno y las acciones internas, tal y como hemos comentado en capítulos y apartados anteriores, la herramienta Jason está desarrollada en Java, cuando hablamos del entorno de nuestro marco social y las acciones internas de nuestros agentes, todo ello está escrito en java.

El código java desarrollado en este proyecto para crear nuestro marco social, sirve sobre todo para su inicialización:

- Obtener las recetas que cada agente Amigo tendrá en su base de conocimientos.
- Crear las relaciones de amistad que los agentes tienen entre ellos y su correspondiente valoración de esa amistad.
- Adjuntar a cada agente su correspondiente lista de amigos y recetas propias.

Algunas de las funcionalidades expuestas arriba simplemente será necesario ejecutarlas una vez, para montar el marco social, como por ejemplo crear los índices de relación de amistad y las recetas que corresponden a cada agente Amigo. Se podría ejecutar cada vez, pero además de ralentizar la ejecución, carecería de sentido a la hora de realizar distintas pruebas para valorar la herramienta, puesto que aun sin cambiar las variables de la prueba los resultados serían diferentes.

Por el contrario, agregar las relaciones de amistad y las recetas correspondientes a cada agente a la base de conocimientos de éste, se ha de cada vez que se ponga en marcha la herramienta, hemos de tener en cuenta que esto es una simulación de un entorno social, en uno real no haría falta puesto que esto ya estaría en la base de datos correspondiente.

El proceso de inicialización le adjunta a cada agente Amigo 50 recetas al azar de la base de datos de recetas original. Sobre la relación con los otros agentes, a cada agente Amigo del sistema le asigna al azar, entre 5 y 10, amigos de los agentes Amigo y Cliente del sistema.

Una vez se han creado estas relaciones el proceso de inicialización las valora, tal y como se ha comentado anteriormente, con un valor al azar de 0.1 a 1, ambos inclusive.

5. Validación

En este capítulo se va a tratar de analizar los beneficios del marco social desarrollado, y las pruebas realizadas para ver la mejoría de una red social, frente a un sistema de recomendación tradicional.

Una de las mejores virtudes de esta herramienta es el hecho de que, al implementar un sistema de recomendación, en una red social, ésta no necesitaría de una base de datos con todos los elementos con una posible recomendación montada por los administradores, si no que con un simple formulario adaptado al elemento sobre el que quieras montar el marco de recomendaciones, bastaría para que los propios usuarios aportasen la información a la base de datos.

Como se ha comentado anteriormente, en el marco actual, tenemos 100 agentes, cada uno con entre 5 y 10 amigos, esto es así debido a que si pusiéramos más amigos para cada agente en apenas un par de niveles de profundidad habríamos visto todas las recetas del sistema.

Cada uno de estos agentes Amigo tiene 50 recetas al azar de las 9020 totales, siendo posibles que varios agentes tengan algunas recetas en común.

Otro aspecto a tener en cuenta es la forma de valorar las recetas, recordemos que se realiza dividiendo los ingredientes que contiene la receta por los que queremos que contenga, o solicitados por el cliente.

$$Ingredientes\_receta \;/\; ingredientes\_solicitados$$

Es muy difícil por lo tanto, obtener una valoración alta en este apartado, es por ello que en las ponderaciones para la valoración se decidió dar un 80% del peso a esta valoración, frente al 20% del apartado social, pues es bastante común tener una valoración por encima de 0.5, y se distorsionaban mucho las valoraciones, no hay que olvidar que estamos recomendando recetas y no valorando relaciones entre amigos. En un sistema real, se podrían igualar más los pesos o dependiendo de los valores de confianza, e incluso tenerlos más en cuenta que las propias valoraciones de las recetas.

5.1. Experimento 1: variación de la profundidad

En la siguiente prueba, se va a mostrar cómo a medida que aumenta la profundidad de la consulta, es decir, cuantos más amigos contactemos, mejor será la recomendación.

Los parámetros de la pruebas són:

- Profundidad = 2
- Ingredientes = cebolla y nata
- Amigos en el sistema = 100
- Valoración: por defecto -> 0.8*valor_propio + 0.2*valor_social

Ponemos como profundidad en la herramienta 2, para que haya relación social más allá de la del cliente con sus amigos, y estos son los resultados (ver Figura 11):

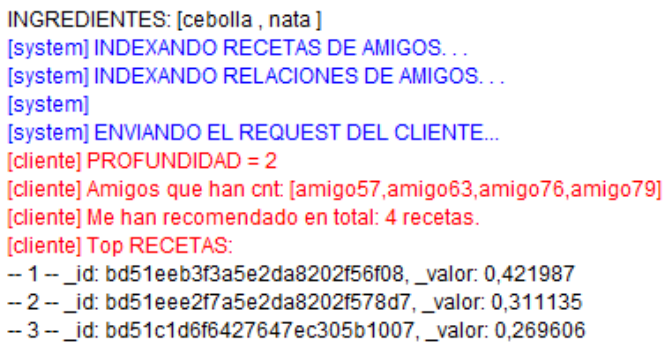

Figura 11: Captura ejecución con profundidad 2

Como se observa en la imagen, se encuentran 3 recetas, sin embargo han contestado 4 amigos, esto se debe a que algún amigo habrá recomendado la misma receta que otro, cuando esto ocurre se mantiene la receta con mayor valoración.

En la imagen vemos que la mayor valoración es de 0.42 y la menor de 0.26, pese a esto, lo que más llama la atención sigue siendo que solo se hayan encontrado 3 recetas, puesto que si hacemos los cálculos, con una media de 7 amigos y 50 recetas por amigo, hemos consultado:

7(nivel 1) x (50 x (7(nivel 2)x50)) = 2800 recetas.

Este dato no es exacto, de estas recetas habrá algunas que sean comunes a los amigos que han sido consultados. Además 7 es un valor aproximado, los agentes tienen entre 5 y 10 amigos, como se ha dicho anteriormente.

A continuación vamos a variar la profundidad (ver Figura 12), con los mismos parámetros de antes, pero esta vez con profundidad 6.

Por lo tanto, los parámetros son:

- Profundidad = 2
- Ingredientes = cebolla y nata
- Amigos en el sistema = 100
- Valoración: por defecto -> 0.8*valor_propio + 0.2*valor_social

```
INGREDIENTES: [cebolla , nata ]
[system] INDEXANDO RECETAS DE AMIGOS. . .
[system] INDEXANDO RELACIONES DE AMIGOS. . .
[system]
[system] ENVIANDO EL REQUEST DEL CLIENTE...
[cliente] PROFUNDIDAD = 6
[cliente] Amigos que han cnt: [amigo63,amigo57,amigo79,amigo76]
[cliente] Me han recomendado en total: 12 recetas.
[cliente] Top RECETAS:
-- 1 -- _id: bd51ef238ca5e2da8202f585aa, _valor: 0,485680
-- 2 -- _id: bd51eeb3f3a5e2da8202f56f08, _valor: 0,380727
-- 3 -- _id: bd51ef23a0a5e2da8202f585b1, _valor: 0,365680
-- 4 -- _id: bd51ef64aca5e2da8202f590a2, _valor: 0,365680
-- 5 -- _id: bd51efd4cba5e2da8202f59f6d, _valor: 0,343458
-- 6 -- _id: bd51efe378a5e2da8202f5a1c6, _valor: 0,325680
-- 7 -- _id: bd51eee2f7a5e2da8202f578d7, _valor: 0,311135
-- 8 -- _id: bd51eef641a5e2da8202f57d05, _valor: 0,299013
-- 9 -- _id: bd51ef5be9a5e2da8202f58ee2, _valor: 0,288757
-- 10 -- _id: bd51ef0160a5e2da8202f57f22, _valor: 0,279966
```

Figura 12: Captura ejecución con profundidad 6

Como se puede observar, lo más destacable respecto a la anterior consulta es el hecho de que imprima 10 recetas, esto es porque como mínimo le han sido devueltas 10, en concreto los agentes le han enviado 12 recetas.

La profundidad aumenta el número de agentes consultados y por lo tanto las recetas a las que accede, se muestra que la recomendación es mejor, más allá de que la mejor valoración sea más alta. Hay más recetas entre las que elegir.

Concluimos pues que la profundidad es un factor clave a la hora de hacer una recomendación, lo cual era de esperar, cuanta más información tengamos mejor será nuestra respuesta.

A continuación, se va a mostrar como el hecho de variar la profundidad, hace que la valoración media de la consulta aumente.

Este hecho es bastante sencillo de imaginar, cuanta más profundidad, más recetas consultamos y por lo tanto, al quedarnos con las mejores, mejor valoración final tendremos (ver Figura 13).

Los parámetros para el experimento son:

- Profundidad = [1...8]
- Ingredientes = dientes de ajo
- Amigos en el sistema = 100
- Valoración: por defecto -> 0.8*valor_propio + 0.2*valor_social

Antes de nada, se ha de tener en cuenta que la valoración utilizada radica en los ingredientes solicitados, por lo tanto en cuanto una receta tenga dos o más ingredientes la valoración bajara a 0.5 o menos. Generalmente en la Base de Datos que tenemos las recetas tienen como mínimo 4 ingredientes, por lo tantos las valoraciones serán por lo general bajas.

Se ha escogido un solo ingrediente, pero bastante común, para que se encuentre varias recetas con cualquier nivel de profundidad, a continuación los resultados:

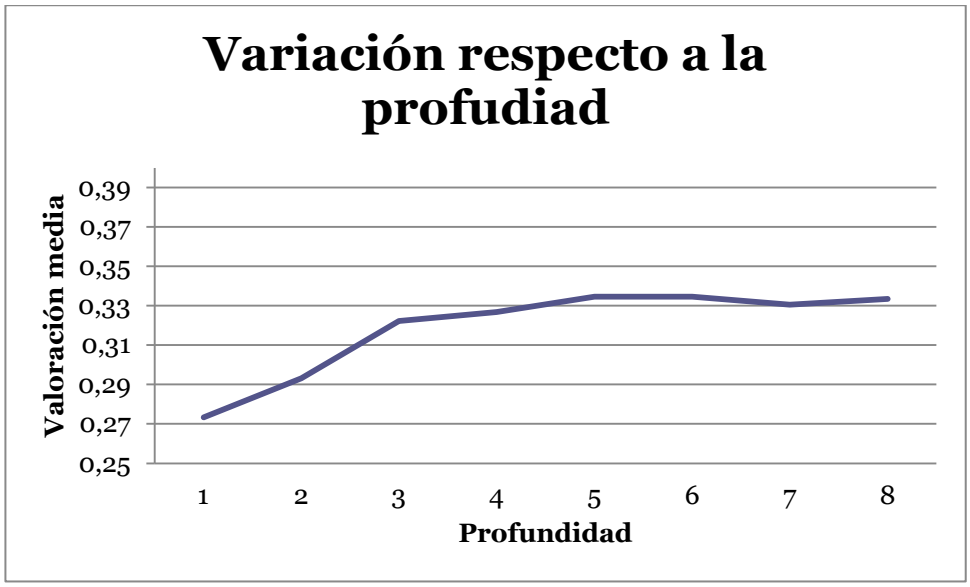

Figura 13: Gráfica de variación de la valoración dependiendo de la profundidad

Tal y como se observa en la gráfica, la valoración aumenta en los primeros niveles de profundidad de una forma acusada, y posteriormente, sobre los niveles de profundidad 4 y 5, se estabiliza. Como es lógico, cuanto mayor sea la base de datos más niveles de profundidad necesitará para estabilizarse.

Esta valoración media se ha realizado mostrando 10 recetas, esto implica que además de mostrar la mejor receta posible en cada consulta, la valoración de las restantes a cada nivel de profundidad será mejor.

Cabe añadir que en sistemas complejos con agentes BDI los resultados pueden ser indeterministas, en este caso en concreto, la mayoría de veces, dos ejecuciones de profundidad alta, como 4 o más, no saldrían valoradas con la misma media, variaría un poco y el orden de las recetas no sería exactamente el mismo.

En este caso, la variación se debe sobre todo a la parte social, es decir, si se consulta antes a un agente, desde un amigo con buena relación, que desde uno con mala, la información que introduzca este agente en el sistema tendrá una mayor valoración.

Con esto se muestra cómo afecta la profundidad, no solo al número de recomendaciones que puede realizar, si no al nivel medio de las valoraciones que haga el sistema, cuando más profunda sea la consulta mejor valoración media devolverá, es decir, más recetas útiles le recomendará al usuario.

5.2. Experimento 2: variación del sistema de valoración.

Finalmente, se ha de comprobar cómo afecta el sistema de valoración al sistema, más concretamente cómo balancear la ponderación entre el *valor_propio* de la receta y el *valor_social*.

Se ha de tener en cuenta que el sistema montado para este proyecto es un marco social recomendador de recetas culinarias, por lo tanto el valor social ha de tener un efecto sobre el sistema pero sin dejar que predomine sobre la receta. Por lo que se puede suponer que para este sistema el valor social ha de ser inferior al valor objetivo de cada receta.

Además, otro valor añadido a la valoración social es el poder evitar recomendaciones de gente que no tiene recetas valoradas tal y como se debería o simplemente han puesto lo primero que se les ha ocurrido en su lista de recetas favoritas, en consecuencia tendrían un valor social bajo, y por lo tanto, esto haría que su valoración fuese inferior.

Para mostrar esto vamos a suponer un perfil: Una persona que quiere encontrar recetas con cebolla y nata, por lo que se supone, han de tener pocos ingredientes además de estos dos.

Los parámetros de la prueba son:

- Profundidad = 100
- Ingredientes = cebolla, nata
- Amigos en el sistema: 100
- Valoración: variable…

En la siguiente figura (Figura 14), se muestran las 10 recetas en las que la nata y el sabor de la cebolla predominan.

-- 1 -- _id: bd51ef238ca5e2da8202f585aa, _valor: 0,400000
-- 2 -- _id: bd51eeb3f3a5e2da8202f56f08, _valor: 0,333333
-- 3 -- _id: bd51ef23a0a5e2da8202f585b1, _valor: 0,250000
-- 4 -- _id: bd51ef64aca5e2da8202f590a2, _valor: 0,250000
-- 5 -- _id: bd51efd4cba5e2da8202f59f6d, _valor: 0,222222
-- 6 -- _id: bd51efe378a5e2da8202f5a1c6, _valor: 0,200000
-- 7 -- _id: bd51eee2f7a5e2da8202f578d7, _valor: 0,181818
-- 8 -- _id: bd51eef641a5e2da8202f57d05, _valor: 0,166667
-- 9 -- _id: bd51ef5be9a5e2da8202f58ee2, _valor: 0,153846
-- 10 -- _id: bd51c1d6f6427647ec305b1007, _valor: 0,142857

Figura 14: Listado de recetas con nata y cebolla

Partiendo de esta lista de 10 recetas y haciendo variar la ponderación social vamos a mostrar los resultados.

Se ha procedido a aumentar en 0.1 la valoración social, reduciendo en la misma medida la valoración propia de la receta, y los resultados se muestran a continuación (Figura 15):

- Valoración: 0.9*valor_propio + 0.1*valor_social

-- 1 -- _id: bd51ef238ca5e2da8202f585aa, _valor: 0,442840
-- 2 -- _id: bd51eeb3f3a5e2da8202f56f08, _valor: 0,377660
-- 3 -- _id: bd51ef23a0a5e2da8202f585b1, _valor: 0,307840
-- 4 -- _id: bd51efd4cba5e2da8202f59f6d, _valor: 0,282840
-- 5 -- _id: bd51ef64aca5e2da8202f590a2, _valor: 0,282030
-- 6 -- _id: bd51efe378a5e2da8202f5a1c6, _valor: 0,262840
-- 7 -- _id: bd51eee2f7a5e2da8202f578d7, _valor: 0,246476
-- 8 -- _id: bd51eef641a5e2da8202f57d05, _valor: 0,232840
-- 9 -- _id: bd51ef5be9a5e2da8202f58ee2, _valor: 0,221302
-- 10 -- _id: bd51ef0160a5e2da8202f57f22, _valor: 0,211411

Figura 15: Recetas con nata y cebolla, valoración social de 0.1

Tal y como se muestra en la figura, el orden de las tres primera recetas permanece igual, además se observa un incremento significativo en ambas, debido a la valoración positiva de la relación del cliente con los amigos que las han proporcionado.

Por otro lado se observa como la 4 y la 5 receta han intercambiado posiciones respecto a la valoración original (ver Figura 14), y la 10ª receta ha sido sustituida por una nueva. Todos estos cambios se deben a las relaciones sociales y niveles de amistad de los agentes.

A continuación balanceamos los niveles de valoración otorgando casi el mismo peso a ambas (Figura 16):

- Valoración: 0.6*valor_propio + 0.4*valor_social

```
-- 1 -- _id: bd51ef238ca5e2da8202f585aa, _valor: 0,550640
-- 2 -- _id: bd51eeb3f3a5e2da8202f56f08, _valor: 0,510640
-- 3 -- _id: bd51ef64aca5e2da8202f590a2, _valor: 0,460640
-- 4 -- _id: bd51ef23a0a5e2da8202f585b1, _valor: 0,460640
-- 5 -- _id: bd51efe378a5e2da8202f5a1c6, _valor: 0,451360
-- 6 -- _id: bd51efd4cba5e2da8202f59f6d, _valor: 0,443973
-- 7 -- _id: bd51eef641a5e2da8202f57d05, _valor: 0,431360
-- 8 -- _id: bd51eee2f7a5e2da8202f578d7, _valor: 0,419731
-- 9 -- _id: bd51ef0160a5e2da8202f57f22, _valor: 0,417074
-- 10 -- _id: bd51ef5be9a5e2da8202f58ee2, _valor: 0,402948
```

Figura 16: Recetas con nata y cebolla, valoración social de 0.4

Se observan dos cambios importantes respecto a la lista anterior (ver Figura 15), la primera es el cambio de la receta "*bd51ef64aca5e2da8202f590a2*", que de la 5ª posición ha alcanzado la 3ª, en esta ocasión el agente que la recomendó tenía un buena valoración social y por lo tanto subió dos posiciones.

Pero lo que más contrasta respecto a la lista inicial (ver Figura 14), es el hecho de que la receta "*bd51ef0160a5e2da8202f57f22*", que no estaba en la lista, haya llegado incluso a ascender en el ranking de 10 recetas.

Como última muestra de la importancia de la valoración social, vamos dotar de la mayor parte de la valoración a la parte social y dejar con el mínimo a la valoración objetiva (Figura 17).

- Valoración: 0.1*valor_propio + 0.9*valor_social

```
-- 1 -- _id: bd51ef238ca5e2da8202f585aa, _valor: 0,785560
-- 2 -- _id: bd51ef64aca5e2da8202f590a2, _valor: 0,770560
-- 3 -- _id: bd51ef23a0a5e2da8202f585b1, _valor: 0,770560
-- 4 -- _id: bd51efd4cba5e2da8202f59f6d, _valor: 0,767782
-- 5 -- _id: bd51efe378a5e2da8202f5a1c6, _valor: 0,765560
-- 6 -- _id: bd51eee2f7a5e2da8202f578d7, _valor: 0,763742
-- 7 -- _id: bd51ef5be9a5e2da8202f58ee2, _valor: 0,760945
-- 8 -- _id: bd51ef0160a5e2da8202f57f22, _valor: 0,759846
-- 9 -- _id: bd51eeda61a5e2da8202f57644, _valor: 0,756671
-- 10 -- _id: bd523b40e0a5e2da768fb054e0, _valor: 0,753893
```

Figura 17: Recetas con nata y cebolla, valoración social de 0.9

Tal y como se iba observando en las anteriores listas de las anteriores valoraciones, la receta "*bd51ef64aca5e2da8202f590a2" ha subido posiciones, incluso la receta "bd51ef0160a5e2da8202f57f22*", que originalmente no aparecía ha subido a la 7 posición.

Sin embargo se observan cambios drásticos, como por ejemplo que la receta que estaba en 2º puesto (ver Figura 16) ha desaparecido de la lista, esto sucede debido a que ahora prácticamente no valoramos que tenga pocos ingredientes además de los requeridos ("cebolla y nata").

Adicionalmente, con cada incremento en la valoración social se observaba un incremente en el valor medio de la recomendación, hecho sencillo de explicar si tenemos en cuenta lo que se dijo al inicio de este capítulo, donde se mencionaba que la valoración objetiva utilizada para las recetas, obtenía valores bastante bajos, aunque no suponía un problema debido a que esto sucedía en todas las recetas por lo tanto no era relevante. Este hecho, sin embargo, nos ayuda a que en las valoraciones como la anterior en las que prácticamente solo se tiene en cuenta el valor social, no aparezcan recetas que no tengan que ver con la solicitud.

Destacar que las pruebas nos permiten validar el marco para hacer recomendaciones sociales y que es un punto de partida para poder probar algoritmos más sofisticados.

6. Conclusiones y trabajo futuro

Finalmente se va a comentar punto por punto, los objetivos que se expusieron en el capítulo 2, y comentar las posibilidades que presentan este tipo de aplicaciones de recomendación basadas en un marco social.

6.1. Resolución del objetivo y subobjetivos.

Primero comentar, que tal y como se ha expuesto en los capítulos 3, 4 y 5, el objetivo principal de este Proyecto de Fin de Grado ha sido alcanzado, tanto en el aspecto de desarrollo como en el de evaluación.

Además, se ha ido comentando el desarrollo seguido en el que se puede observar todos los subobjetivos propuestos, tanto la información relevante de una receta (aspecto que también se comentará a continuación, en el trabajo futuro), estudio de la BD, acceso a ella y estudio del estado arte sobre los agentes BDI. Así como la parte de desarrollo del sistema, diseño de la red social e implantación final de la herramienta.

Finalmente en el capítulo 5, se ha procedido a desarrollar diferentes métodos de evaluación de esta herramienta, tal y como se comentaba en el subobjetivo final.

Una vez obtenidos todos los subobjetivos, ha sido demostrada la importancia de aportar a un sistema simple de recomendación un marco social, aún más conociendo el peso de las redes sociales hoy día y cómo influyen en nuestra vida cotidiana, por ella se destaca la relevancia y el nivel de beneficio de este tipo de sistemas, aplicando un nivel extra de recomendación, basada en las relaciones entre usuarios.

6.2. Trabajo futuro.

Para terminar, se ha de comentar las posibilidades, que tiene este proyecto, de trabajo futuro, tanto en el plano docente, como en el empresarial.

Respecto a los beneficios en el plano docente, se puede utilizar para realizar un estudio y mostrar os diferentes algoritmos de recomendación, tanto probando distintos métodos de recorrido del ‘árbol’ de amigos, como probar con diferentes pesos de valoración de las recetas, algo similar al último experimento del capítulo anterior.

Por otro lado, se podría tener en cuenta para generar tanto páginas web de recomendaciones como para implantar este tipo de herramientas en redes sociales, en las que por supuesto tendría una gran utilidad dado que la base de datos a utilizar podría ser creada en base a los elementos que añadan los propios usuarios.

Posibles mejoras para este marco social de recomendaciones serían: podríamos incluir unas valoraciones mucho más exhaustivas, incluyendo más información del perfil del cliente, teniendo en cuenta campos de las recetas más allá de los ingredientes, podemos incluir búsquedas para gente con trastornos alimenticios, como por ejemplo celíacos, con el colesterol alto, etc; también se podría montar una búsqueda que incluyese los campos de kilocalorías y las cantidades diarias recomendadas, los cuales son ideales para gente que se regule la alimentación, por ejemplo deportistas, modelos o sencillamente para dietas.

Posteriormente tenemos suficiente información como para mostrar un resumen de estas recetas o, al contrario, seguir una descripción detallada de su elaboración, instrucciones e ingredientes.

Más allá de los datos de las recetas, dependiendo de los elementos a recomendar, el sistema de recomendación podría basarse exclusivamente en las relaciones de los usuarios, por ejemplo: En un sistema de recomendación de películas, podrías basarte en recomendar películas según la valoración que tus amigos les hayan dado a estas películas y tu relación con ellos.

Por todo lo comentado, podemos concluir destacando la utilidad de esta herramienta tanto para aplicaciones basadas exclusivamente en marcos de recomendación, como para su implementación en redes sociales pudiendo realizar pequeñas variaciones en su recorrido y valoración. Y por lo tanto, siendo útil en el ámbito académico, para mostrar distintas estrategias de valoración y las repercusiones que estos cambios conllevan.

7. Bibliografía

1. Pazzani, M., Billsus, D. In: Content-Based Recommendation Systems. Volume 4321 of Lecture Notes in Computer Science. Springer-Verlag (2007) 325_341

2. Schafer, J., Frankowski, D., Herlocker, J., Sen, S. In: Collaborative Filtering Recommender Systems. Volume 4321 of Lecture Notes in Computer Science. Springer-Verlag (2007) 291_324

3. Burke, R.: Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction 12(4) (2002) 331_370

4. Chesñevar, C., Maguitman, A., González, M. In: Empowering

5. van der Aalst, W.M.P., Song, M.: Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In: 2nd Intenational Conference on Business Process Management. Volume 3080 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (June 2004) 244_260

6. Adomavicius, G., Tuzhilin, A.: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering 17(6) (2005) 734_749

7. Zhou, X., Xu, Y., Li, Y., Josang, A., Cox, C.: The state-of-the-art in personalized recommender systems for social networking. Arti_cial Intelligence Review 37(2) (2012) 119_132

8. Linden, G., Hong, J., Stonebraker, M., Guzdial:, M.: Recommendation Algorithms, Online Privacy and More. Communications of the ACM 52(5) (2009)

9. Ana Mas. (2005). Agentes Software y Sistemas Multi-Agente: Conceptos, Arquitecturas y Aplicaciones. Madrid: Pearson Educación.

10. M. E. Bratman, D.J.Israel, M.E.Pollack. Plans and Resource-Bounded Practical Reasoning. Computational Intelligence, 4: pp. 349-355, 1988.

11. Jason, A Java-based interpreter for an extended versión of AgentSpeak. [acceso 26 de junio de 2014] http://jason.sourceforge.net/.

12. MongoDB, an open-source document database, and the leading NoSQL database. [acceso 26 junio de 2014] http://www.mongodb.org/.

13. MongoDB, Replication. [acceso 2 julio de 2014] http://docs.mongodb.org/manual/replication/.

Anexo

1. Estructura de las recetas

- _id: clave primaria
- origin: web de origen de la receta [quecocinohoy, elcorteingles, consumer]
- nutrition: datos nutricionales calculados por EDAMAM. compuesto de:
  -- cautions
  -- healthLabels (ej. ["LOW_FAT_ABS", "VEGAN", "VEGETARIAN", "PALEO", "DAIRY_FREE", "GLUTEN_FREE", "WHEAT_FREE", "EGG_FREE", "MILK_FREE", "PEANUT_FREE", "TREE_NUT_FREE", "SOY_FREE", "FISH_FREE", "SHELLFISH_FREE"])
  -- calories: calorias calculadas por EDAMAM
  -- uri: url aleatoria
  -- yield : número de comensales (según EDAMAM)
  -- dietLabels (ej. ["HIGH_FIBER", "LOW_FAT"])
  -- totalDaily (ej. { "PROCNT" : { "label" : "Protein", "unit" : "%", "quantity" : 21.8898 } })
  -- cautions (ej. ["GLUTEN", "WHEAT"])
  -- totalNutrients (ej. { "ENERC_KCAL" : { "label" : "Energy", "unit" : "kcal", "quantity" : 567.15 } })
- title: titulo de la receta
- url: url de origen de la receta
- ingredientes: listado de ingredientes. lista de items compuestos de:
  -- name_en: nombre del ingrediente en ingles
  -- name: nombre del ingrediente en castellano
  -- amt: cantidad. compuesto de:
    --- unit_en: unidades en ingles (ej. grams)
    --- unit: unidades en castellano (ej. gramos)
    --- qty: cantidad (ej. 100)
- yield: número de comensales (según la web original)
- recipe_id: identificador de receta. número secuencial
- observations: observaciones o notas a la receta
- images: lista de imágenes (sus nombres de fichero)
- tips: Notas. Compuesto de:
  -- Tiempo de preparación
  -- Dificultad
  -- Temporada
  -- Tipo de cocina
  -- Precio
- instructions: objeto (clave/valor) donde la clave es el orden y el valor un paso de las instrucciones. (ej. {1: "calentar agua", 2: "cocer la pasta"})
- keywords: etiquetas o tags (similares a tips)

- nutrition_extra: información adicional nutricional (proporcionada por nutricionistas). Compuesto de:
  -- fibra
  -- sal
  -- grasa_sat
  -- grasa
  -- calorias
  -- azucares
  -- enfermedades-desaconsejadas
  -- enfermedades-aconsejadas