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

SISTEMA DE RECOMENDACIÓN DE EXPERTOS PARA ROS ANSWERS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

BRAULIO ENRIQUE LÓPEZ PINO

PROFESOR GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
PABLO GUERRERO PEREZ
SERGIO OCHOA DELORENZI

SANTIAGO DE CHILE
2019

SISTEMA DE RECOMENDACIÓN DE EXPERTOS PARA ROS ANSWERS

ROS (Robot Operating System), es un framework para escribir software para robots. Con una colección de herramientas, librerías y convenciones que tienen como objetivo simplificar la tarea de crear comportamiento complejo y robusto para robots. ROS promueve la colaboración entre grupos en el desarrollo de software. Por esto, provee herramientas y librerías open source creadas por su comunidad que son alojadas en su mayoría en Github.

Desarrollar software para robots involucra profesionales de distintas disciplinas que se enfocan en temas específicos. La ventaja de usar ROS, es que facilita herramientas que usuarios pueden usar sin la necesidad de especializarse en ellas para saber cómo funcionan. Es decir, por simplicidad, es una caja negra que recibe entradas y produce salidas. El problema ocurre cuando una herramienta no funciona como los usuarios creen, dado que fue desarrollada por especialistas en otros temas, entonces no tienen conocimiento suficiente para solucionar su problema. Es entonces cuando los usuarios recurren a ROS Answers, una plataforma de preguntas y respuestas.

Dado que las preguntas publicadas en ROS Answers tratan temas específicos a las herramientas en sí, es importante encontrar a una persona que pueda guiar a quienes no tienen conocimiento sobre esos temas. Actualmente en ROS Answers, al igual que en StackOverflow [29], las preguntas son organizadas según tags o tópicos, y cada usuario puede dar un voto positivo o negativo a una pregunta o respuesta, dependiendo de si cree que un aporte es útil o no. Estos votos en las preguntas y respuestas de un usuario, más su historial de participación dentro de esa comunidad, es la única información disponible para identificar si un usuario domina un tema, y por ende confiar en su respuesta.

ROS y ROS Answers forman un ecosistema de software. Un ecosistema, es una colección de sistemas de software, que se desarrollan y evolucionan en un mismo ambiente, en este caso el ambiente es la comunidad de robótica open source. Sin embargo, falta considerar una parte importante de este ecosistema, Github. Este posee información relevante sobre la experiencia y conocimiento de usuarios que han colaborado a la comunidad, no sólo respondiendo preguntas, sino también participando activamente en el desarrollo de las herramientas que componen cada una de las distribuciones de ROS.

En esta memoria se presenta una herramienta desarrollada para obtener información de usuarios de Github y ROS Answers (plataforma de preguntas y respuestas), se implementa un algoritmo de recomendación de usuarios para responder una pregunta en ROS Answers, y por último se crea un verificador para evaluar localmente el algoritmo, utilizando preguntas ya respondidas en ROS Answers. También se logra verificar la correctitud del algoritmo implementado realizando una encuesta web a usuarios de ROS Anwers, donde se consulta sobre su conocimiento para responder preguntas sobre temas que el algoritmo sugiere que ellos manejan. Este trabajo está abierto a futuras extensiones, como por ejemplo, incluir más algoritmos.

Gracias papá, mamá y hermanos. Tenían razón, esto me iba a gustar.

Agradecimientos

Gracias a todos los que me apoyaron en este largo proceso. Comenzando por quienes me guiaron en esta memoria, Pablo Estefó y Jocelyn Simmonds. Luego, a todos las personas que fueron parte importante de mi carrera, Pablo Bustamente y Jaime Capponi. Finalmente a Dora, Lucky, mi familia y en especial a mi alegría, Camila.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

ROS (Robot Operating System) [22], es un framework para escribir software para robots. Con una colección de herramientas, librerías y convenciones que tienen como objetivo simplificar la tarea de crear comportamiento complejo y robusto para robots. ROS promueve la colaboración entre grupos en el desarrollo de software. Por esto, provee herramientas y librerías open source creadas por su comunidad que son alojadas en su mayoría en Github.

Un alumno de doctorado en el DCC, estudia dinámicas de colaboración en ecosistemas de software, en particular el de ROS. Es decir, estudia cómo la comunidad de ROS (Robot Operating System) consigue trabajar de manera conjunta en busca de un fin común, aprenden a trabajar juntos y realizan actividades en las que participan todos. Para ello, quiere realizar un experimento para saber si recomendando usuarios calificados a responder preguntas no respondidas, en la plataforma de preguntas y respuestas ROS Answers, [25] aumenta el nivel de participación de la comunidad.

Generalmente, para que un usuario pueda responder una pregunta, se debe tener conocimiento técnico y teórico sobre el tema abordado. En el caso de ROS, se espera que un usuario que haya participado en el desarrollo de una herramienta para esta plataforma, tenga alto conocimiento teórico sobre esa herramienta sobre la que trabajó. Por otro lado, de un usuario de ROS Answers, podemos suponer que si tiene preferencia para responder dudas sobre alguna herramienta en particular, y además sus respuestas son valoradas positivamente por los demás miembros de la comunidad, tenga alto conocimiento técnico sobre esa herramienta.

Se desea averiguar si recomendando al usuario mejor calificado según ambos tipos de conocimiento, de ahora en adelante experto, para responder una pregunta que aún no ha sido respondida, mejoraría la calidad de las respuestas en ROS Answers. Para ello, supone que: si el usuario que responde, es el que más sabe del tema que trata la pregunta, entonces esa respuesta puede concluir la pregunta, es decir, que será capaz de entregar los recursos necesarios para que otra persona pueda comprender y resolver el problema planteado.

1.2. Descripción

Actualmente, existen sistemas de recomendación de usuarios para ciertas tareas, que funcionan de forma similar a lo que se busca en esta memoria, pero con objetivos distintos. Por ejemplo, Zhang et al. [39] publicó un sistema de recomendación de desarrolladores, DevRec. Este recomienda desarrolladores, registrados en Github [14], para que participen en proyectos open source alojados en la misma plataforma. Su recomendación para un proyecto se basa en medir y comparar el conocimiento teórico y técnico de los candidatos sobre este proyecto al cual serán recomendados. Para un proyecto en particular, la medida de el conocimiento técnico lo obtiene de la participación del usuario respondiendo preguntas relacionadas al proyecto en StackOverflow [29]. Por otro lado, la medida de el conocimiento teórico lo obtiene de la participación del usuario en proyectos similares a este en Github. De esta forma, Zhang et al. propone encontrar al desarrollador mejor calificado para un proyecto en particular. Si bien, DevRec considera ambos tipos de conocimiento, la diferencia entre lo que se busca en esta memoria y Zhang et al., es que en este tranbajo se buscan expertos para responder preguntas mientras que el otro los busca para desarrollar en un proyecto.

Al igual que el ejemplo anterior, hay otros sistemas que comparten características como el propuesto por Tian et al. [31], este sólo se basa en el conocimiento técnico. Para recomendar un experto, mide el interés de todos los usuarios de StackOverflow en distintos temas. Si bien, los tipos de conocimientos involucrados son los mismos, pero ninguno ha sido implementado sobre la plataforma ROS y ROS Answers. Por otro lado, actualmente es difícil replicar un experimento. Según un artículo publicado en Nature [2], que muestra resultados de una encuesta a investigadores sobre replicar experimentos, más de un 70 % de los investigadores reportaron haber fallado al intentar replicar un experimento de otro investigador y peor aún, más de la mitad de ellos falló alguna vez en reproducir su propio experimento. De igual forma, en este caso replicar los experimentos propuestos en los papers no es sencillo ya que generalmente no se provee el código fuente o los datos que fueron utilizados en cada experimento. Sólo se cuenta con la descripción del funcionamiento del sistema y sus resultados.

1.3. Solución propuesta

Se propone contar con la implementación de un sistema de recomendación de expertos para responder preguntas en ROS Answers, que considere habilidades técnicas y teóricas de los usuarios. El sistema debe ser diseñado de tal manera que permita incorporar distintos algoritmos de recomendación. Esto es para que sea posible estudiar distintas formas de hacer recomendaciones sin necesitar hacer grandes modificaciones al sistema. Además, para que los algoritmos implementados puedan ser comparados, es necesario obtener retroalimentación de los usuarios recomendados por el sistema, para ello se propone la creación de un sitio web que despliegue las preguntas recomendadas para un usuario. Así, el usuario podrá dar recomendaciones y sugerencias acerca de sus habilidades y disposición para responder las preguntas que le son indicadas.

1.4. Objetivo

1.4.1. Objetivo general

Implementar y evaluar un sistema que recomiende expertos para responder preguntas en ROS Answers. El sistema debe permitir incorporar distintos algoritmos de recomendación. Además, crear un sitio web donde cada usuario recomendado por el sistema pueda entregar una evaluación de las recomendaciones entregadas por el sistema.

1.4.2. Objetivos específicos

1. Identificar para al menos un 30 % de los aproximadamente 2.000 usuarios que han contribuido en paquetes ROS en Github, su cuenta en ROS Answers y extraer datos de sus cuentas en ambas plataformas.

2. Analizar dos algoritmos de recomendación: el utilizado en DevRec y el propuesto por Tian et al. Para determinar qué características comparten y cómo pueden ser implementados en ROS Answers.

3. Diseñar e implementar el sistema recomendador de expertos para responder preguntas y que incluya los algoritmos de recomendación de DevRec y de Tian et al.

4. Crear un sitio web que despliegue las preguntas de ROS Answers recomendadas para un usuario, para que éste pueda responder sobre su cualificación, disposición y evaluación de la dificultad de responder cada pregunta recomendada.

1.5. Metodología

Dado que el origen de esta memoria se da en la necesidad de un investigador en tener una herramienta para validar algoritmos de recomendación, es que para definir los objetivos y alcance de esta memoria se realizaron reuniones con el alumno de doctorado interesado en el estudio de estos algoritmos y se utilizó la plataforma Trello [34], que es un organizador de tareas, para llevar registro de las tareas para definir la memoria. La definición de objetivos se dividió en dos etapas: breve recopilación de información sobre algoritmos de recomendación existentes y definición de objetivos.

En la primera etapa se analizó distintos algoritmos de recomendación como DevRec [39], Tian et al. que serán explicados en el desarrollo de esta memoria. Luego, con esta información se definió los datos que se deben conseguir como mínimo para replicar uno de ellos, DevRec.

En la segunda etapa, se definió el alcance de este proyecto que contempla la recolección de datos desde las plataformas ROS Anwers y Github, enlace de cuentas de usuario entre ambas plataformas, implementación de algoritmo de recomendación existente a modo de ejemplo,

chequeo de los resultados de un algoritmo y creación de una encuesta web para validar los resultados con los mismos usuarios de ROS Answers.

Luego se procedió a desarrollar esta memoria en 3 etapas. La primera etapa, consistió en la recolección de datos. Para el investigador esta información es útil en su trabajo, por lo tanto con la ayuda de Trello se organizó tareas donde se indicaba qué información era relevante para él y además se registraban las entregas de estos datos. A medida que se avanzaba en la recolección de estos datos, la entrega de estos se realizaba mediante links externos a Trello puesto que estos recursos superaban el máximo establecido por Trello para compartir en su plataforma.

La segunda etapa corresponde a la recolección de datos, se estudió en detalle DevRec y Tian et al. En esta etapa se diseñó las herramientas contenidas en esta memoria tomando en cuenta ambos algoritmos.

Finalmente, en la tercera etapa, se realizó una encuesta web con los resultados de DevRec. Esta permite encuestar usuarios de ROS Answers enviando un correo solicitando su participación. Este correo se obtiene al enlazar sus identidades con cuentas en Github.

1.6. Desarrollo de la memoria

La memoria se divide en 4 capítulos. Primero el capítulo (2), donde se detalla todo el contexto en el que se desarrolla esta memoria incluyendo herramientas que fueron utilizadas y herramientas que fueron creadas para cumplir los objetivos de esta memoria. Segundo capítulo (3), en el cual se evidencia los requisitos de usuario y la forma en que fueron utilizadas las herramientas descritas en el Marco teórico para cumplir con los requisitos. Tercero capítulo (4), aquí se explica cómo se aplica el diseño explicado en Análisis y diseño incluyendo modelos de datos, diagramas de clases y ejemplos de uso de las herramientas. Finalmente el cuarto, capítulo (5), donde se diseña el cómo será validada esta memoria y los datos obtenidos de las validaciones.

Capítulo 2

Marco teórico

En este capítulo se describen herramientas analizadas para resolver el desafío propuesto en esta memoria. Se comienza en la Sección 2.1, explicando qué es ROS y ROS Anwers, además se indica qué datos son importantes para esta memoria. Después en la Sección 2.2, se enseñan distintas herramientas y maneras de extraer datos de Github y ROS Answers. En la Sección 2.3 se explica el funcionamiento de FRIL, que es una herramienta que enlaza información por similitud. Luego en la Sección 2.4 se presentan alternativas para almacenar toda la información recolectada. Finalmente, en la Sección 2.5 se describen dos algoritmos de recomendación de usuarios.

2.1. Robot Operating System

ROS [22] (de su nombre en inglés Robot Operating System) es una colección de frameworks para el desarrollo de software para robots. Este provee servicios como abstracción de hardware, control de dispositivos de bajo nivel y administración de paquetes. El software en ROS se distribuye en librerías o paquetes, la mayoría de estos paquetes están licenciados bajo distintas licencias de código abierto e implementan funcionalidades y aplicaciones de uso común, como controladores de hardware, modelos de robots, tipos de datos, planificación, percepción, localización y mapeo, herramientas de simulación y otros algoritmos. Muchos de los paquetes de ROS están orientados hacia un ambiente Unix, principalmente porque estos paquetes dependen de colecciones de software open source disponibles para sistemas basados en Unix.

Puesto que los paquetes son open source, estos están disponibles en plataformas de control de versiones de código cómo Github o Bitbucket. En particular, existe un repositorio [21] en Github que contiene la información de los paquetes incluidos en cada versión de ROS. Las versiones disponibles son: groovy, hydro, indigo, jade, kinetic, lunar y melodic.

En la Figura 2.1 se observa un ejemplo de la información contenida por cada paquete en cada versión de ROS. La información de un paquete contiene 3 secciones: `doc`, `release` y `source`. Estas secciones contienen información sobre documentación, publicación y código

fuente respectivamente. Generalmente la información de `doc` coincide con `source`.

```yaml
abseil_cpp:
  doc:
    type: git
    url: https://github.com/Eurecat/abseil-cpp.git
    version: master
  release:
    tags:
      release: release/kinetic/{package}/{version}
    url: https://github.com/Eurecat/abseil_cpp-release.git
    version: 0.2.3-0
  source:
    test_pull_requests: true
    type: git
    url: https://github.com/Eurecat/abseil-cpp.git
    version: master
  status: maintained
```

Figura 2.1: Datos de un paquete incluido en una distribución de ROS.

Puesto que desarrollar software para robots no es una tarea sencilla y ROS está orientado a la colaboración open source, es que existe ROS Answers, un sitio donde todos pueden realizar preguntas u orientar a otros desarrolladores en sus inquietudes. ROS Answers, es una plataforma de preguntas y respuestas (de su nombre en inglés Q&A) destinada a resolver dudas sobre ROS. Tanto las preguntas como las respuestas pueden recibir votos positivos o negativos, de esta forma si a un usuario le parece útil un aporte puede sumar un voto o en caso contrario restar un voto si cree que el aporte perjudica. Entonces, cualquier persona puede aportar preguntando o respondiendo sin importar su nivel de conocimiento y es la misma comunidad quien decide si una pregunta o respuesta es aceptable o no.

Las preguntas son organizadas según tags (o etiquetas) que representan los temas sobre los que se discute en una pregunta. Estos tags se organizan de forma plana. Luego, un usuario puede buscar preguntas relacionadas a tags de su interés para responder o puede definir qué tags ignorar. Por lo tanto, los tags de intereses, tags ignorados, preguntas respondidas, preguntas creadas y votos recibidos en respuestas es la información que se puede encontrar en ROS Answers para medir el conocimiento de un usuario de ROS Answers.

2.2. Extracción de datos

Existen herramientas que permiten recopilar información de sitios de tipo (Q&A). A continuación se describen alternativas para extraer datos desde Github y ROS Answers que se estudió como parte de esta memoria.

2.2.1. API askbot

ROS Answers está basada en la plataforma askbot [1], que provee una API para consultar información de usuarios y preguntas realizadas. La información disponible es la siguiente:

Usuario: Puede ser consultado una lista de los usuarios existentes o por identificador. En la Figura 2.2 se muestran los datos asociados al usuario `tfoote`. Esta es toda la información que dispone la API. De estos datos, son relevantes username y reputation.


```
{
  username: "tfoote",
  gold: 95,
  joined_at: "1297685622",
  answers: 1678,
  questions: 10,
  id: 3,
  bronze: 422,
  comments: 1705,
  silver: 308,
  reputation: 46787,
  avatar: "//www.gravatar.com/avatar/3bc6528f8fe1fef6b6ecc7312ee0cf04?s=48&amp;amp;d=identicon&amp;amp;r=PG",
  last_seen_at: "1535251601"
}
```


Figura 2.2: Resultado de https://answers.ros.org/api/v1/users/3/

Preguntas: Puede ser consultado una lista de las preguntas existentes o por identificador. En la Figura 2.3 se muestra la información que entrega la API para una pregunta en ROS Answers. Los datos relevantes son tags, title (título de la pregunta), author, summary (contenido de la pregunta en formato html) y score (suma de votos positivos y negativos recibidos en la pregunta).


```
{
  tags: [
      "kinetic",
      "move_base",
      "source_code"
  ],
  answer_count: 1,
  accepted_answer_id: null,
  answer_ids: [
      301634
  ],
  id: 301632,
  last_activity_by: {
      username: "lucasw",
      id: 120
  },
  view_count: 6,
  last_activity_at: "1535297320",
  title: "How to edit package source code?",
  url: "http://answers.ros.org/question/301632/how-to-edit-package-source-code/",
  author: {
      username: "topkek",
      id: 34642
  },
  added_at: "1535295408",
  summary: "&lt;p&gt;I get the source code files for move base with apt-get source, but how do i
  saved and compiled??&lt;/p&gt;\n",
  score: 0
}
```


Figura 2.3: Resultado de https://answers.ros.org/api/v1/questions/301632/

La información provista por la API es útil, pero esta no entrega datos como los tags de interés y correo de cada usuario y la cantidad de votos que tiene cada respuesta. Además, los identificadores contenidos en `answer_ids` (ver Figura 2.3) no pertenecen a los usuarios que respondieron las preguntas sino que a las respuestas, y no hay un endpoint documentado donde rescatar esa respuesta según su identificador.

2.2.2. Gitana

Gitana [10] es una herramienta construida en `python` que permite importar y procesar datos de repositorios en Git y se sitios Q&A para luego almacenarlos en una base de datos para facilitar la búsqueda y consulta de toda esta información. Esto es algo similar a lo que se requiere para el objetivo específico 1 de esta memoria: extraer datos de repositorios en Github (Git) y ROS Answers (Q&A).

Visitando su repositorio oficial, se inspeccionó el código para verificar la factibilidad de usar Gitana con ROS Answers. En esta inspección se encontraron dos formas de extraer datos de sitios Q&A, una para StackOverflow y otra para foros de Eclipse. Estas dos formas de extraer eran específicas a las plataformas de las que extraen datos. Para StackOverflow, existe la implementación de un `querier`, que es la clase que extrae los datos, que usa la API de StackOverflow para acceder a ellos. Por otro lado, para Eclipse existe la implementación de otro `querier` que extrae datos usando un scraper. Al ver que no hay una forma de reutilizar alguna herramienta de Gitana para extraer datos de ROS Answers se decidió abrir un issue [9] en el repositorio de Gitana para consultar si existe otra posibilidad de integración con ROS Answers. La respuesta, por parte de un colaborador de Gitana, sugiere implementar nuestro `querier` que integre ROS Answers. Este además comenta que no es simple por lo que estaba dispuesto a servir como guía. Por último, el colaborador de Gitana recomienda otra herramienta, Perceval [19], que extrae datos de plataformas en askbot.

2.2.3. Perceval

Perceval es una herramienta que extrae contenido de diferentes sitios Q&A, incluyendo los que usan la plataforma askbot. Según su documentación [18], la información que entrega Perceval incluye las preguntas y respuestas. Esto lo hace usando la API de askbot y un scraper creado por ellos mismos.

2.2.4. Scraper

Puesto que la API de askbot no cubre todos los datos disponibles en ROS Answers, otra opción es hacer un scraper que recorra ROS Answers y extraer directamente de allí información que no entrega la API, suma de los votos de cada respuesta, mail de contacto, tags de interés, tags ignorados, etc. Para esto existen herramientas como Jsoup [16] que permiten navegar y obtener datos de una web. Se realizaron algunas pruebas logrando extraer datos exitosamente, que se describen a continuación. Para ello, se creó una herramienta en Java que utiliza Jsoup para recolectar información como tags, usuarios, preguntas, respuestas, etc. Con esta herramienta los resultados de la extracción fueron 21.388 usuarios, 40.995 preguntas, 128.433 respuestas o comentarios y 14.253 tags. Esta extracción fue realizada el 15 de julio y se logró extraer el 100 % de los usuarios, 99 % de preguntas y el 99 % de los comentarios a esa fecha. Estos dos últimos porcentajes son estimados puesto que existen preguntas [23] que son registrados como "wiki" y no pertenecen a un usuario en especifico.

2.2.5. API Github

La forma común de extraer datos de Github es utilizando su API [11] que provee toda la información necesaria para este proyecto: commits, autores e información de repositorios. Esta tiene un rate limit de 60 requests por minuto. Se realizaron pruebas de obtención de datos de repositorios y usuarios con éxito. Estas pruebas incluyeron la creación de una herramienta en Java donde se extrajo 2.858 usuarios de 1.679 repositorios asociados a ROS.

2.3. Enlace de cuentas de usuarios

Luego de la recolección de datos, es importante relacionar a los usuarios de Github y ROS Answers para analizar el comportamiento de cada usuario en ambas plataformas. FRIL [8] permite relacionar información utilizando algoritmos como por ejemplo el de distancia de edición, que según cuantos cambios necesite una cadena de texto para convertirse a otra (distancia) entrega un puntaje y según esta discrimina si dos datos están relacionados o no. Esta herramienta posee una interfaz gráfica, como se aprecia en la Figura 2.4.

FRIL recibe 2 entradas (marcadas con A en la figura 2.4) que son archivos *.csv que contienen la información que se quiere relacionar. En esta sección se define qué columnas de cada entrada se van a utilizar para enlazar cuentas. Incluso se puede procesar una columna antes de ser usada, por ejemplo, se puede filtrar los datos de la columna usando expresiones regulares o eliminar datos duplicados.

Luego, se define cómo se va a relacionar esta información y qué algoritmos se van a utilizar (marcado con B en la figura 2.4). Se escoge una columna de cada entrada definida en Data Source y luego se escoge un algoritmo que va a definir su similitud. Entre los algoritmos disponibles están:

- **Equal fields boolean distance:** Compara las entradas indicando si son exactamente iguales o no.
- **Edit distance:** Cuenta cuántas operaciones se debe aplicar para convertir la primera entrada a la segunda. Las operaciones incluyen eliminación, inserción o intercambio de caracteres.
- **Numeric distance:** Orientada a datos numéricos, chequea que la diferencia entre las dos entradas esté dentro de un rango que debe ser especificado.

Finalmente se selecciona qué campos de los seleccionados en Data Source se quiere en el *.csv que arroja como salida (marcado con C en la figura 2.4). Adicionalmente, es posible definir qué hacer en los casos en que FRIL debe escoger entre varios datos relacionados con el mismo puntaje. Se puede elegir manualmente, al azar o definir algún algoritmo.

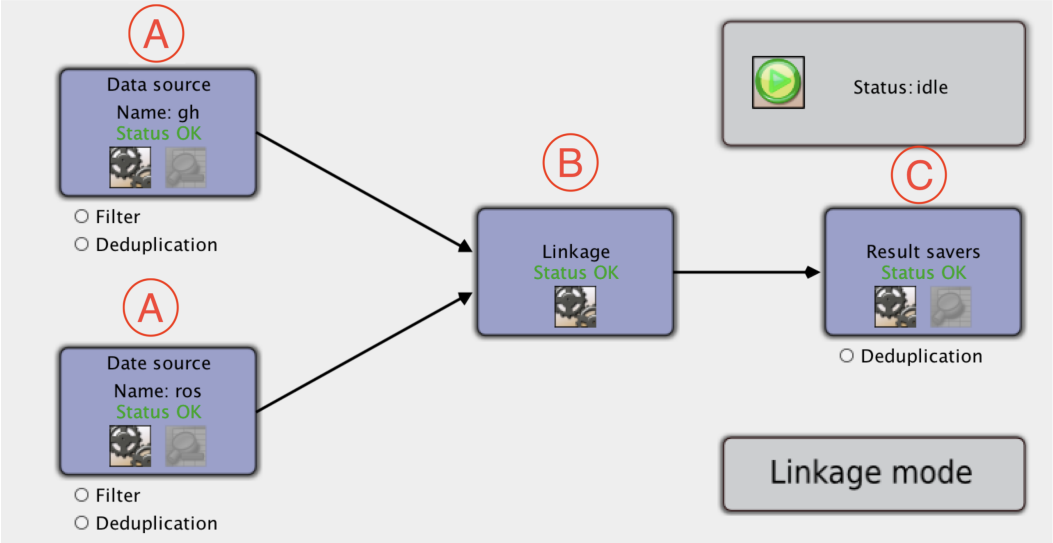

Figura 2.4: Interfaz gráfica de FRIL, configurada para enlazar similitudes en cuentas de Github y ROS Answers a partir de archivos csv.

2.4. Almacenamiento de la información

Según cifras publicadas por ROS [20], a la fecha en ROS Anwers se han realizado más de 42.360 preguntas y 23.500 usuarios registrados. Los tipos de datos a almacenar son principalmente texto y números. Existen distintas formas de almacenar esta información, por ejemplo texto plano o bases de datos relacionales y no relacionales. A continuación se describen dos alternativas para almacenar datos. Un aspecto importante a considerar es la facilidad para compartir los datos.

2.4.1. SQLite

SQLite [27] es una base de datos relacional que a diferencia de la mayoría de otros sistemas de administración de bases de datos, no es un motor de base de datos basado en cliente-servidor. En vez de eso, SQLite incluye todo en un sólo programa para ser utilizado. Por esto mismo SQLite es una opción popular como software de base de datos integrado para almacenamiento en software de aplicación, como los navegadores web. Podría decirse que es el motor de base de datos más ampliamente implementado, ya que es utilizado en la actualidad por varios navegadores, sistemas operativos y sistemas integrados (como los teléfonos móviles), entre otros. De esta manera, para compartir información, basta con compartir el mismo archivo de base de datos.

2.4.2. MySQL

Es un administrador de bases de datos que a diferencia de SQLite necesita estar alojado en un servidor. Luego para compartir información se puede compartir un dump de los datos o un acceso (usuario y contraseña) para acceder a la base de datos. MySQL [17] soporta muchos más tipos de datos que SQLite.

2.5. Algoritmos de recomendación

Como se mencionó en la sección 1.2, existen algoritmos de recomendación de usuarios que se basan en uno o dos tipos de conocimiento, teórico y técnico. Como primer ejemplo, se describe el sistema DevRec, cuyo algoritmo de recomendación considera el conocimiento teórico y técnico. Por otro lado, se resume un estudio realizado por Tian et al. donde buscan expertos para responder preguntas considerando solamente el conocimiento técnico.

2.5.1. DevRec

Para un proyecto open source es importante atraer desarrolladores. Esta tarea es difícil para un proyecto debido a la gran cantidad de posibilidades que tiene un desarrollador para elegir a qué repositorio contribuir. DevRec, tiene como objetivo mejorar esta interacción recomendando desarrolladores preparados para participar en proyectos open source. Este sistema combina la actividad del usuario participando en proyectos open source en Github y que a la misma vez hacen preguntas y las responden en StackOverflow, para luego recomendar desarrolladores como candidatos a participar en proyectos open source.

Se basa en dos supuestos. Primero, que los desarrolladores con intereses teóricos similares, tienden a participar en proyectos similares. Segundo, que los desarrolladores con intereses técnicos similares, tienden a enfocarse en las mismas preguntas, que son marcadas con los mismos tags. Por ejemplo, desarrolladores de Ruby tienden a participar en StackOverflow preguntando y respondiendo preguntas sobre un interés en común, que en este caso es Ruby.

En la Figura 2.5, se describe la arquitectura de este sistema donde se puede identificar las etapas: Data Extraction, Separate Recomendation y Recomendation Integration. Cada etapa será explicada a continuación.

Data Extraction: Esta etapa involucra la extracción de datos. A modo de experimentación, se usaron datos de Github alojados en un dump de SQL en GHTorrent [12], por otro lado para StackOverflow se usaron datos de un dump de StackExchange [28] que es la plataforma en la que está basada StackOverflow. Luego emparejaron identidades de distintas cuentas de usuario comparando los correos con la función de hash MD5 [36]. Luego crean una matriz que relaciona usuario y proyecto en Github R_{u-p}, donde 1 significa que participó y 0 que no. De la misma manera crean una matriz que relaciona usuarios con tags de StackOverflow R_{u-t}. Finalmente a partir de cada matriz se crean dos matrices que relaciona todos

los usuarios R_{u-u}.

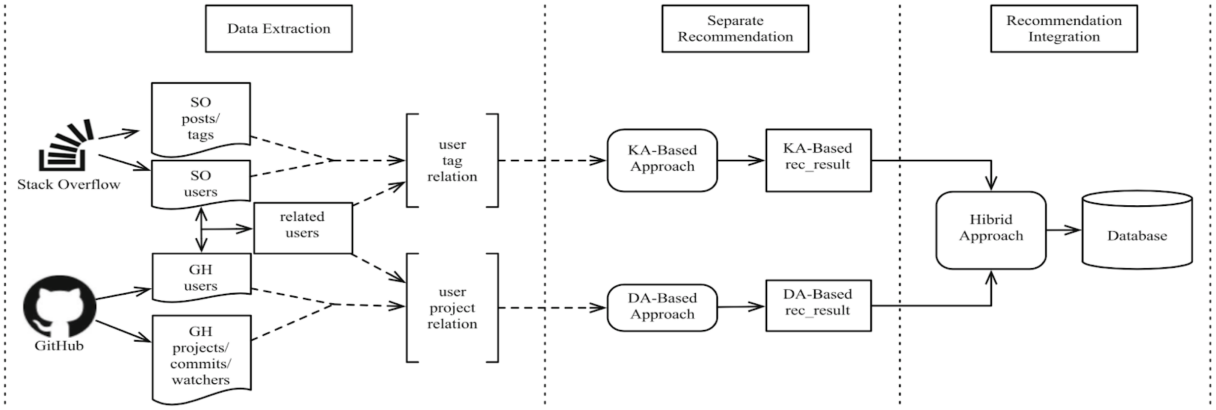

Figura 2.5: Arquitectura de DevRec (fuente: [39]).

Separate Recomendation: A partir las dos matrices R_{u-u} se calculan las métricas development activity (DA) y knowledge sharing activity (KA), que son basadas en los supuestos 1 y 2 respectivamente. Estas dos métricas sirven para recomendar usuarios a proyectos o a preguntas por separado. DA se define de la siguiente manera, sea $UA_p < u_1, u_2, ..., u_n >$ los usuarios que participan en el proyecto p. Se obtiene el puntaje para cada usuario respecto al proyecto p usando la Eq. (2.1) con R_{u-u} creada a partir de R_{u-p}. Por otro lado KA se calcula de la misma manera pero con información relacionada a tags.

$$DA = \sum_{\mathrm{i}=1}^{|UA_p|} R_{u-u}[UA_p[\mathrm{i}]] \tag{2.1}$$

Recomendation Integration: Finalmente realizan una aproximación híbrida que considera las dos métricas DA y KA ponderadas por un factor que regula cuánta importancia se da a cada una.

De esta forma la entrada y salida de DevRec es la siguiente:

Entrada/Salida

entrada	salida
1. Preguntas en StackOverflow y proyectos en Github en los que participa cada usuario. 2. Un proyecto para el cual se busca recomendar un desarrollador.	Lista de usuarios rankeados según su expertise para aportar al proyecto ingresado como entrada.

Tabla 2.1: Entrada y salida de DevRec

Experimento de prueba:

1. Incluyó 72,877 usuarios de StackOverflow.

2. Incluyó 165.741 proyectos de Github donde participaba al menos un usuario incluido en el paso anterior.

3. Concluye que la métrica DA es más precisa que KA en repositorios populares. Sin embargo, al combinarlas obtienen mejores resultados para recomendar repositorios no populares.

2.5.2. Predicción del usuario con la mejor respuesta

Tian et al. proponen un enfoque para predecir el mejor "respondedor" para una nueva pregunta en un sitio de una comunidad de preguntas y respuestas. Considera el interés y la expertise del usuario en tags relevantes sobre la pregunta dada.

Es importante destacar que este enfoque sólo considera el conocimiento técnico del usuario ya que utiliza solamente información obtenida de StackOverflow.

Se basa en tres supuestos. Primero, el candidato más calificado debe tener un alto interés y expertise en la pregunta dada. Segundo, las preguntas son generadas por tags. Por último, el expertise del usuario y su interés en una pregunta están afectados por su expertise e interés en sus tópicos relacionados. Su objetivo es comparar los resultados con los obtenidos usando tf-idf [35] el cual entrega una estadística que indica qué tan importante es una palabra dentro de un documento.

Descripción en alto nivel del algoritmo

1. Se crean perfiles de usuario a través de recolectar un histórico de preguntas con sus respuestas.

2. Usa Latent Dirichlet Allocation (LDA) [4], un modelo probabilístico que entrega descripciones de miembros en una colección de datos discretos tales como textos. Con esto, se crea un modelo de los perfiles de usuario para aprender sobre sus intereses en diferentes tópicos.

3. Los tópicos de conocimiento de los usuarios son modelados en base a la información histórica de las preguntas respondidas.

4. Los votos sobre una pregunta, en StackOverflow, implican que el usuario tiene expertise en sus tópicos relacionados.

Entrada/Salida

entrada	salida
1. Histórico de preguntas respondidas por cada usuario. 2. Una pregunta para la que se busca un respondedor.	Lista de usuarios rankeados por su probabilidad de ser el más calificado para responder.

Tabla 2.2: Entrada y salida de algoritmo de Tian et al.

Experimento de prueba:

1. Incluyó 99.000 preguntas de StackOverflow.

2. Su performance es mejor que enfoques basados en TF-IDF.

2.5.3. Algoritmo a implementar

La principal diferencia entre los dos algoritmos descritos en la sección 2.5 son los datos usados y el conocimiento en los que se basan para dar un ranking a los usuarios. Mientras que DevRec se basa en la participación de los usuarios medida a través de la cantidad de veces que participa en tags, Tian et al. se basan en la búsqueda en el contenido de las preguntas de los usuarios. Por otro lado, DevRec considera los tipos de conocimiento teórico y técnico al dar un ranking y Tian et al. sólo consideran el conocimiento técnico, es decir la participación en sitios Q&A.

Capítulo 3

Análisis y diseño

Una vez definido el marco teórico, en este capítulo se detalla los requisitos que fueron resultado de reuniones con el alumno de doctorado (ver Sección 3.1). Luego en la Sección 3.2 se enseña el diseño ejecutado para cumplir los requisitos y objetivos de esta memoria. Finalmente, en la Sección 3.3 se hace un resumen de los objetivos que son abarcados en esta memoria.

3.1. Análisis

En la primera etapa de esta memoria, se realizaron reuniones con el alumno de doctorado donde se definió este proyecto. Se requiere un sistema de recomendación de respondedores a preguntas en ROS Answers. A continuación se describen los requisitos que debe cumplir esta memoria para ayudar en el estudio de las dinámicas de colaboración en ecosistemas de software, en particular el de ROS.

Se define como Cliente al alumno de doctorado y como usuario a cualquier investigador que quiera utilizar o extender esta herramienta y a los usuarios de ROS Answers.

RU.1 Perfil de usuarios en ROS Answers y Github

Descripción: Se debe crear un perfil de usuarios recolectando información de ROS Answers y Github. Extrayendo datos de todos los usuarios de ROS Answers y de los usuarios de Github que hayan participado en el desarrollo de algún proyecto de ROS.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Funcional

T. Usuario Asociado: Investigador

RU.2 Enlace de cuentas en ROS Answers y Github

Descripción: Identificar identidades entre las dos fuentes de datos. Es decir, ver las cuentas recolectada en Github y ROS Answers que pertenecen a la misma persona. Las cuentas que no puedan ser identificadas deben ser incluidas en el conjunto de datos de todas formas.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Funcional

T. Usuario Asociado: Investigador

RU.3 Recomendar usuarios según distintos algoritmos

Descripción: Este sistema debe ser capaz de ejecutar distintos algoritmos de recomendación de usuarios. Para una pregunta sin respuesta se recolectan los usuarios mejor calificados para responder según cada algoritmo.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Funcional

T. Usuario Asociado: Investigador

RU.4 Inclusión de distintos algoritmos de recomendación

Descripción: Debe existir la posibilidad de implementar distintos algoritmos de recomendación distintos al incluido en el sistema para ser ejecutados por el sistema.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Funcional

T. Usuario Asociado: Investigador

RU.5 Validación de un algoritmo

Descripción: Se debe validar si los resultados de la implementación de un algoritmo son correctos utilizando preguntas ya respondidas: el algoritmo debería dar un mejor ranking al usuario que respondió correctamente en los primeros lugares.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Calidad

T. Usuario Asociado: Investigador

RU.6 Pequeña encuesta web para los usuarios recomendados

Descripción: Se requiere de una pequeña encuesta web para que los usuarios de ROS Answers puedan responder sobre su cualificación, disposición y evaluación de la dificultad de responder una pregunta recomendada.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Funcional

T. Usuario Asociado: Investigador, Usuario ROS Answers

RU.7 Almacenamiento de los resultados de la encuesta

Descripción: La encuesta contiene 3 preguntas que quedarán registradas en una base de datos para un análisis posterior.

Fuente: Cliente

Estabilidad: Intransable

Tipo: Restricción

T. Usuario Asociado: Investigador

3.2. Diseño

En base a lo descrito en la sección 3.1, se propone la siguiente solución para cumplir el objetivo de esta memoria. Este proyecto es construido usando Java 8 (scraper y perfilamiento), también se usa SQLite para almacenar los datos debido a la facilidad de compartir datos y de poner en marcha. Por otro lado, la API Github provee toda la información de Github que necesita este proyecto. Similar a la API de Github, la API askbot contiene información parcial de ROS Answers que es util en este trabajo. Para enlazar cuentas se utiliza FRIL puesto que se obtuvo buenos resultados probando con los datos descritos en la sección 2.2.4 y por ultimo se usa servlet-jsp para la creación del sitio web aprovechando su integración con Java, el mismo lenguaje en el que el proyecto es construido. Ya se creó un repositorio para el proyecto en Github [26]. El proyecto, tal como se aprecia en la Figura 3.1, se divide en:

1. **Extracción:** Incluye la extracción de los datos y enlazar cuentas de usuario.

2. **Recomendación:** Involucra procesar datos de los usuarios, la aplicación del algoritmo

y la entrega de los resultados.

3. **Validación:** Se encarga de obtener el feedback de los usuarios recomendados por el sistema.

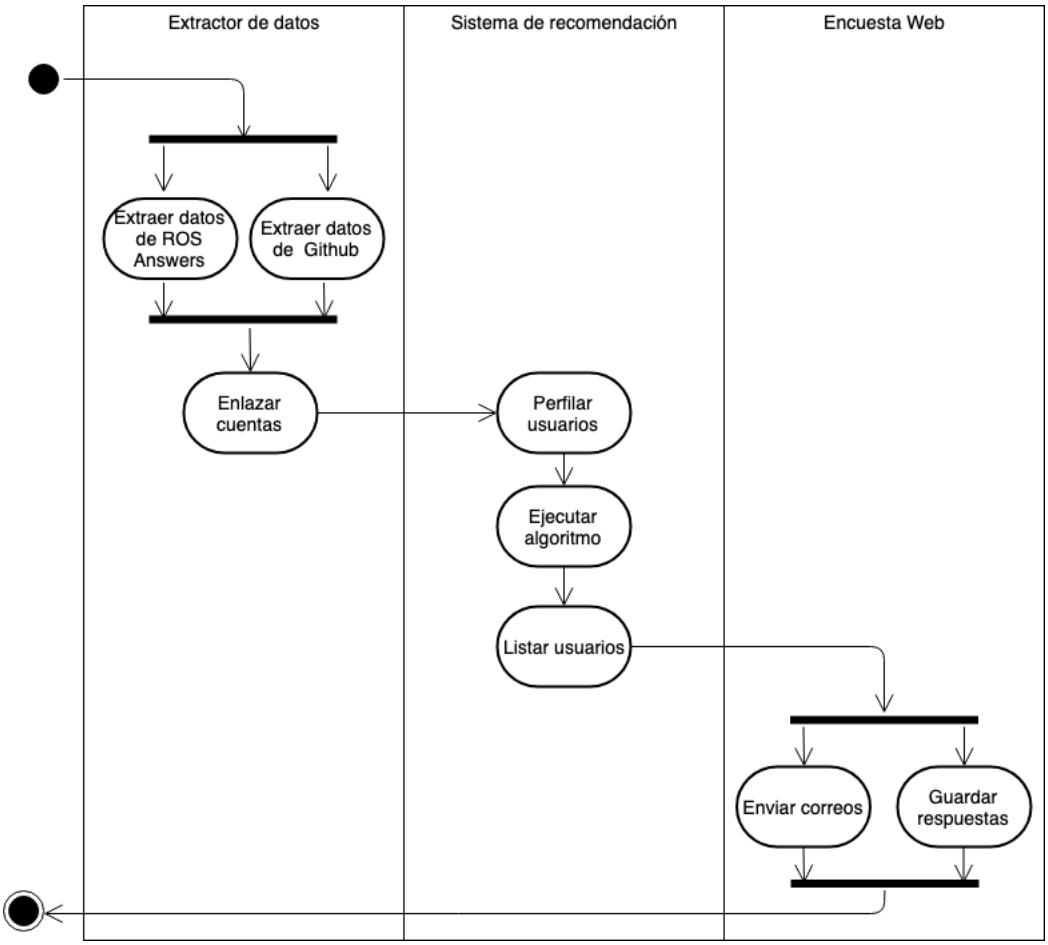

Figura 3.1: Etapas del proyecto Sistema de Recomendación de Expertos para ROS Answers.

Es importante aclarar que la extracción de datos y el sistema de recomendación son independientes. Este proyecto incluirá una forma de obtener datos, pero de todas formas el usuario puede usar otra alternativa si la tiene para completar ese paso. De esta forma, se busca facilitar la replicación de un experimento, por ejemplo, puede ser que quien use esta herramienta ya tenga los datos y sólo necesite aplicar el algoritmo de recomendación. Por lo tanto, las herramientas de obtención de datos incluidas en este proyecto son sólo una ayuda para el investigador, en este caso el alumno de doctorado, y él y otras personas que utilicen este sistema son libres de utilizar estas u otros recursos como Gitana o Perceval para la extracción de datos.

A continuación se detalla cómo se resolverá cada uno de los requisitos propuestos en la sección 3.1.

3.2.1. Extraer datos de ROS Answers

El objetivo de esta sección es resolver el requisito **RU.1** respecto a la información de usuarios en ROS Answers. De ROS Answers se extrae por cada usuario: tags de interés, participación preguntando y respondiendo, total de votos recibidos en cada uno de los aportes, nombre y correo si existe. Debido a que la API no provee toda la información básica, como tags e intereses, que utilizan la mayoría de los algoritmos, como los mencionados en la sección 2.5, la información restante se obtiene del scraper.

De los usuarios de ROS Answers se obtendrá información haciendo scraping sobre la lista de usuarios [24] que provee la plataforma de forma pública. En el caso de las preguntas, se utilizará la API para obtener datos sobre todas las preguntas hechas en ROS Answers y la información que no está incluida en la API como los comentarios, participantes o votaciones, se recolectan haciendo scrapping.

Entonces, el proceso de recolección de información es el siguiente:

1. Se extrae los usuarios y tags haciendo scraping.
2. Se extrae todas las preguntas creadas en ROS Answers:
    - Se consulta la API para obtener detalles como el autor y el cuerpo de la pregunta.
    - Se hace scraping para obtener datos no incluidos en la API.
    - Cada comentario, respuesta o pregunta, se relaciona con los usuarios.

Los datos obtenidos de ROS Answers son representados en el modelo de la Figura 3.2. En este modelo se destaca:

- ra_user: representa a un usuario de ROS Answers. Este contiene información que no puede ser obtenida de la API como los votos recibidos y el nombre real del usuario.
- ra_question: representa una pregunta en ROS Answers. La mayoría de la información está disponible en la API.
- ra_answer: representa una respuesta a una pregunta o a otra respuesta. Toda esta información es obtenida usando el scraper.

3.2.2. Extraer datos de Github

El siguiente paso para completar **RU.1** es obtener datos sobre los usuarios en GitHub que han participado escribiendo código en algún paquete de cualquier distribución disponible de ROS Answers listada en el repositorio oficial. Como se explicó en la sección 2.1, este repositorio organiza todos los paquetes incluidos en cada release de ROS en directorios. Cada paquete contiene distintas direcciones a repositorios de este paquete, y son estas las que se usa para averiguar qué usuarios han participado haciendo commits en alguno de ellos.

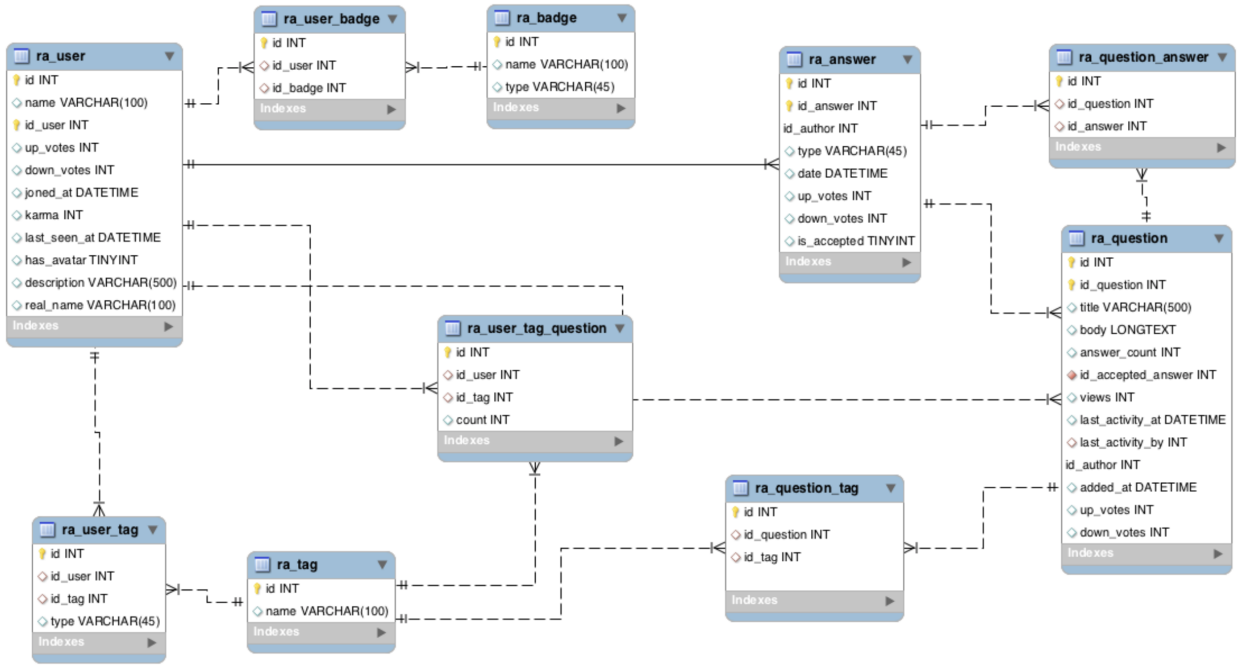

Figura 3.2: Modelo de los datos en ROS Answers.

Cada paquete puede contener la dirección del código fuente (source), documentación (doc) o su release. Generalmente se repiten las direcciones de doc y source. Por otro lado, release, sólo contiene información correspondiente a las distribuciones de ROS para la cual fue lanzado el paquete y no a su desarrollo. Debido a esto, se decidió explorar sólo un repositorio de cada paquete, y como no siempre están definidas las tres direcciones, se explora en el siguiente orden: source, doc, release.

En ocasiones, una parte del paquete está en otra plataforma como Bitbucket [3] o simplemente no está registrada, estos no serán incluidos ya que representan sólo un 4.7 % del total de repositorios de todas las distribuciones de ROS. Entonces, por cada paquete se explora su repositorio, obteniendo la información de los contribuidores y su cantidad de commits por contribuidor. Toda esta información se obtiene usando la API de Github.

Finalmente, toda esta información se almacena en una base de datos junto a la información obtenida de ROS Answers para ser explorada. El modelo de datos extraídos de Github se puede ver en la Figura 3.3, donde la única información que se extrae sobre la participación de un usuario en un repositorio es la cantidad de commits que ha hecho en este.

Para la API de Github simplemente se utilizará la herramienta jcabi-github [15], que sirve para consultar datos a la API de Github de una manera más simple usando Java. Las distribuciones de ROS analizadas son groovy, hydro, indigo, jade, kinetic, lunar y melodic.

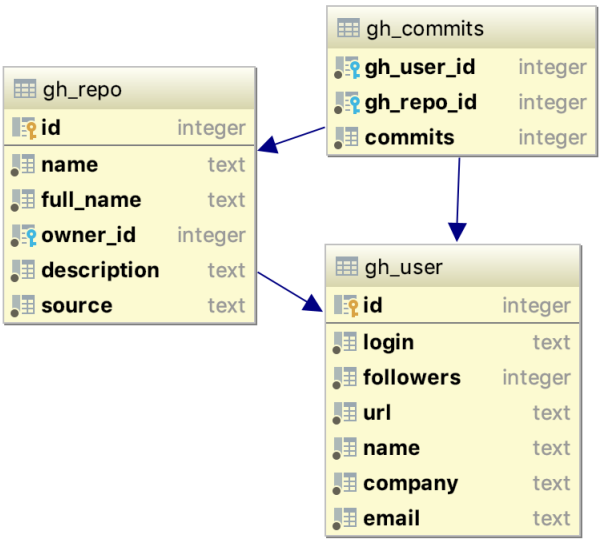

Figura 3.3: Modelo de datos en Github.

3.2.3. Enlazar cuentas

El objetivo del requisito de usuario **RU.2**, es enlazar las cuentas de usuario entre Github y ROS Answers, para ello se utiliza FRIL, esta herramienta permite hacer este enlazar conjuntos de datos utilizando distintos algoritmos. Este punto es importante ya que se decidió utilizar una herramienta externa y con interfaz gráfica para hacer el enlace de cuentas. Esto impide la automatización del proceso ya que necesita esta interacción manual, pero no es problema ya que cuando el sistema esté en funcionamiento, esta tarea manual sólo debe ser ejecutada una vez al mes ya que no se espera que un nuevo usuario se vuelva experto en menos de un mes y por lo tanto se puede ignorar la identificación de usuarios nuevos.

3.2.4. Sistema de recomendación

El sistema de recomendación se compone de distintas herramientas, como el scraper y el chequeo, creadas en esta memoria. Debido a que este sistema está orientado a ayudar a la investigación, cada una se ejecuta de forma independiente, en particular el algoritmo debe permitir gatillar su ejecución de forma manual. Es decir, recibe los perfiles de usuarios y una pregunta para entregar una lista de los usuarios recomendados a responder la pregunta entregada. De esta forma, se puede experimentar con preguntas ya contestadas donde se tiene una idea de cual es la mejor respuesta y quien la contestó. Como se aprecia en la Figura 3.4, el algoritmo a implementar debe retornar una lista de `Aspirant`. De esta manera se puede implementar distintos algoritmos de recomendación que implementen esta interfaz de Java, cumpliendo el requisito **RU.3**.

3.2.5. Implementación de un algoritmo

Estas etapas están pensadas para que si alguien quiere extender este proyecto agregando más algoritmos de recomendación, pueda hacerlo siguiendo estos pasos. De esta manera se cumple el requisito **RU.4**. Para facilitar este trabajo, se debe incluir interfaces (Java) para implementar en cada paso de este proyecto.

Procesar datos

Si bien la mayoría de los algoritmos estudiados usan datos similares como tags, commits e intereses, estos son analizados de formas distintas. Por ejemplo, algunos crean matrices para comparar actividades entre usuarios y otros analizan directamente el texto de las preguntas que ha respondido un usuario para saber si este podría responder algo similar. Es por esto que es necesario comenzar por preparar los datos que serán utilizados en la siguiente etapa.

Aplicar Algoritmo

En esta etapa se recibe los datos procesados, se aplica el algoritmo y finalmente se entrega una lista de los usuarios recomendados.

Listar recomendados

En esta etapa se decide qué hacer con la lista de recomendados, por ejemplo podría escribir un `*.csv` o guardar en una base de datos.

El procesamiento de datos para ser usados en un algoritmo se suele hacer de forma independiente antes de la ejecución del algoritmo, por lo tanto esa etapa no se incluye en la ejecución del algoritmo. Luego, se implementó una interfaz para un algoritmo al que se le solicita aspirantes para una pregunta dada.

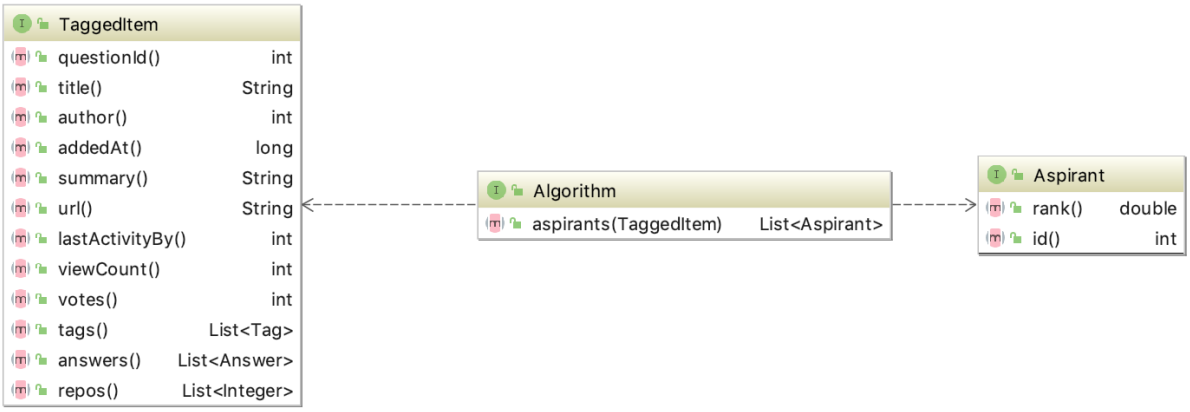

Figura 3.4: Diagrama de clases de la interfaz *Algorithm*

En la Firgura 3.4 se identifican tres componentes en la ejecución del algoritmo:

1. **TaggedItem:** Representa un elemento que contiene tags y será usado como consulta.

2. **Aspirant:** Aspirante a ser respondedor. `rank()` indica el puntaje obtenido.

3. **Algorithm:** Para un `TaggedItem` entrega una lista de `Aspirant`. Cada uno con su puntaje o ranking.

Luego de la ejecución del algoritmo, basta con ordenar los resultados según el ranking. Para ello se creó un objeto simple que ordena `Aspirant` según su ranking. Finalmente la ejecución de un algoritmo se resume en la Figura 3.5.

```
Algorithm devrec = new Devrec(args ...);
TaggedItem item = new TaggedQuestion(args ...);
System.out.println(
    new ByRankAsc().orderedList(
        devrec.aspirants(item)
    )
);
```

Figura 3.5: Ejecución de `Algorithm`

3.2.6. Chequeo del algoritmo de recomendación

El objetivo de esta sección es resolver el punto **RU.5** incluyendo una herramienta para validar el algoritmo con preguntas ya respondidas. Este módulo hace una validación del algoritmo previo a las encuestas a través de la pagina web. Esto se verifica con las métricas precisión, sensibilidad y MRR. Esta última métrica (por su nombre en inglés Mean Reciprocal Rank) es una medida estadística para evaluar un proceso que produzca una lista de posibles respuestas a una muestra de preguntas, ordenado por probabilidad de exactitud. Este se define en la Eq. (3.1).

$$MRR = \frac{1}{|Q|} \sum_{\mathrm{i}=1}^{|Q|} \frac{1}{rank_{\mathrm{i}}} \tag{3.1}$$

Esto mide, en promedio, qué tan cerca del primer lugar en el ranking está la respuesta correcta para cada $rank_{\mathrm{i}}$ de cada consulta Q.

Luego un Chequeo debe dejar logs con los resultados, por lo que se define la interfaz de la Figura 3.6.

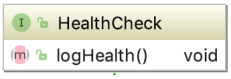

Figura 3.6: Interfaz de Health Check

3.2.7. Encuesta web

El objetivo de construir esta página web es completar los puntos **RU.6** y **RU.7**. Con esto se busca obtener feedback de los mismos usuarios acerca de la correctitud de la recomendación del algoritmo utilizado. Esta plataforma será simple, sólo enviará un correo a cada uno de los usuarios recomendados explicando de qué trata este proyecto e indicando un link donde responder una pequeña encuesta.

Una encuesta contiene 3 consultas relacionadas a una pregunta en ROS Answers. Se espera que para cada nueva encuesta, se envíe un correo a los usuarios mejor calificados con el siguiente contenido (en inglés):

- Breve explicación del proyecto.
- Link a la encuesta.

La página web a la que serán redirigidos mostrará lo siguiente (en inglés):

- Una pregunta de ROS Answers, incluyendo su título, texto y enlace a la pregunta original en el sitio de ROS Answers.
- Pregunta 1 (P1): ¿Tienes los conocimientos suficientes para responder esta pregunta?
- Pregunta 2 (P2): ¿Estarías dispuesto a responder la pregunta?
- Pregunta 3 (P3): Este correo fue enviado automáticamente según tu perfil colaborando a ROS en Github y ROS Answers. De 1 a 4, donde 1 es muy malo y 4 es excelente ¿que tan acertada crees que fue nuestra predicción?
- Pregunta 4 (P4): Espacio para ingresar comentarios acerca de porqué puede o no responder la pregunta propuesta.

Las preguntas P1, P2 y P3 son obligatorias mientras que el comentario de P4 es opcional. Toda esta información será almacenada en una base de datos, lo que permitirá analizar el desempeño de cada algoritmo de recomendación y compararlo con los resultados descritos en sus publicaciones originales.

Cada encuesta contiene las 3 consultas mencionadas anteriormente relacionadas a una pregunta en ROS Answers. Para ello se creó el modelo presente en la Figura 3.7 donde se almacena la información en 3 entidades:

aspirant: Corresponde a todas las preguntas a las que se debe encuestar a un usuario. Esto quiere decir que para cada usuario `gh_user_id` se le encuesta para todas las preguntas `ros_question_id` con las que está relacionado.

survey: Representa la lista de encuestas pendientes por enviar por mail. Para cada `gh_user_id` se debe enviar un correo con todas las preguntas relacionadas en la tabla `aspirant`.

answers: Es donde se almacenan las respuestas a las 3 consultas más el posible feedback.

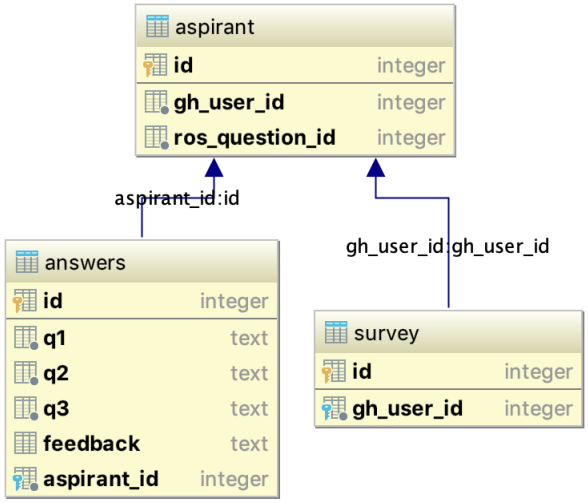

Figura 3.7: Modelo datos de la web de encuestas.

De esta manera este pequeño proceso debe buscar nuevas encuestas para enviar por mail en `survey` y luego encuestar según cuantas preguntas estén relacionadas con el usuario en `aspirant` y que no estén respondidas en `answers`. Para evitar enviar múltiples correos y distintos links para cada pregunta en ROS Answers, se envía un sólo link que llevará al encuestado a su encuesta donde podrá responder las 3 consultas para cada pregunta que tenga asociada a ROS Answers de a una a la vez. Es decir, si tiene asociada dos preguntas de ROS Answers, al entrar al link observará las 3 preguntas para la primera pregunta en ROS Answers, luego podrá apretar un botón *next* que lo llevará a la siguiente encuesta con 3 consultas para la otra pregunta relacionada a ROS Answers y así sucesivamente hasta que el usuario responda todas las encuestas.

3.2.8. Open source y experimento replicable

Puesto que esta herramienta está pensada para ser utilizada en una investigación, es importante que permita replicar experimentos. Es por esto que este proyecto será open source y los datos utilizados en los experimentos serán publicados en el mismo repositorio, de esta manera serán replicables. Pero un proyecto no es open source sólo por dejar su código visible a todo el mundo, si no que además debe facilitar la colaboración y reutilización de este. Por esto, siguiendo sugerencias de Yegor Bugayenko [5], que no define un estándar pero propone ideas interesantes sobre lo que debería contener un proyecto realmente open source, hasta ahora se ha hecho lo siguiente con el repositorio:

1. Está disponible en Github.

2. Contiene un README que explica de qué trata el proyecto e instrucciones de cómo usarlo. También incluye insignias referenciando todas las herramientas CI conectadas al proyecto, para que el estado de salud del proyecto sea visible.

3. Contiene una licencia que permite al proyecto ser modificado y utilizado por terceros.

4. Está conectado a Travis [33] para mostrar la calidad de las builds, Codecov [7] para mostrar el porcentaje de test coverage y a Codebeat [6] para mostrar la calidad del código.

5. Se crean entregas de esta herramienta cómo un artefacto en un repositorio público para facilitar su inclusión en proyectos.

6. Incluye documentación usando JavaDoc.

3.3. Resumen

En la tabla 3.1 se muestra la matriz de trazabilidad entre los requisitos de usuario (listados en la Sección 3.1) y las subsecciones de la Sección 3.2 destinadas a diseñar el cumplimiento de cada requisito.

	3.2.1	3.2.2	3.2.3	3.2.4	3.2.5	3.2.6	3.2.7
RU.1	×	×					
RU.2			×				
RU.3				×			
RU.4					×		
RU.5						×	
RU.6							×
RU.7							×

Tabla 3.1: Matriz de trazabilidad.

Capítulo 4

Implementación

A continuación de detalla cómo se aplicó el diseño explicado en el capítulo 3, comenzando por la extracción de datos en la Sección 4.1. Seguido por la Sección 4.2, donde se explica como se implementó el algoritmo de recomendación. Luego, en la Sección 4.3, se explica cómo se chequea el algoritmo implementado. Finalmente, en la Sección 4.4, se explica como se implemento la encuesta web con la que se validará el algoritmo con los usuarios de ROS Answers.

4.1. Extraer datos de ROS Answers

Se crea un scraper, esta herramienta utiliza Jsoup, que permite descargar el contenido (DOM) de un sitio web y luego explorarlo buscando según tags de HTML, identificadores o clases. Puesto que los usuarios y las preguntas están organizadas en páginas, se debe explorar el Dom de cada una de las páginas que listan a los usuarios o preguntas de ROS Answers. Luego, con el fin de obtener toda esta información separada en páginas, se construyó un iterador de Dom que permite iterar sobre una colección de Doms. El iterador de Dom debe saber dónde comenzar, cómo obtener el siguiente Dom y la última página a la que se debe iterar. Entonces, por cada pagina el iterador devuelve un Dom.

Una vez se obtiene el Dom de una página, es necesario recorrerlo para extraer la información que se busca. Para lograr esto, se crea un iterador sobre el contenido del Dom. De esta forma existe un iterador que devuelve, por ejemplo, preguntas listadas en una página o usuarios listados en una página.

En el ejemplo de la Figura 4.1, IterateDomPages es un iterador de Dom a quién se le indica dónde comenzar (initialPage), cómo obtener el siguiente Dom (RosQuestionsPagedDom) y hasta donde llegar (LastRosAnswersPage). Además, se le indica qué tipo de contenido debe iterar, en este ejemplo son preguntas (IterateByQuestionLinks). Finalmente, este iterador devolverá un iterador de contenido por cada Dom, por ejemplo, uno que itera los links a las preguntas que contiene cada página o los usuarios que aparecen en ellas.

```
new IterateDomPages(
        new RosQuestionsPagedDom(),
        initialPage,
        new LastRosAnswersPage(
                Jsoup.connect(rootDom).get()
        ).value(),
        new IterateByQuestionLinks()
)
```

Figura 4.1: Ejemplo de iteración sobre Dom de páginas que contienen preguntas.

Por último, puesto que tenemos un iterador que devuelve otro iterador, para abstraer el hecho de "iterar iteradores" es que se abstrae el iterador de Dom y de esta forma se le entrega al usuario de esta herramienta la capacidad de iterar links o usuarios y no iteradores de estos. Por ejemplo, en la Figura 4.2 se agrega IteratePagedContent que toma el iterador que se obtiene por cada Dom y lo itera escondiéndolo al usuario que lo usa.

```
new IteratePagedContent<>(
        new IterateDomPages(
                new RosQuestionsPagedDom(),
                initialPage,
                new LastRosAnswersPage(
                        Jsoup.connect(rootDom).get()
                ).value(),
                new IterateByQuestionLinks()
        )
);
```

Figura 4.2: Abstracción de iteradores para el usuario.

De esta forma el usuario de esta herramienta puede iterar como se muestra en la Figura 4.3.

```
while (usersLinks.hasNext()) {
    final String next = usersLinks.next();
    System.out.println(
            new RosDomUser(Jsoup.connect( url: root + next).get())
    );
}
```

Figura 4.3: *userLinks* es un *IteratePagedContent* que permite iterar sobre los links a los usuarios de ROS Answers.

Asi se extrajo la información recolectada hasta ahora y se almacenó una base de datos SQLite.

4.2. Algoritmo de recomendación, DevRec

Como se describió en la sección 2.5, este algoritmo recomienda según conocimientos teóricos y técnicos a los que llama development activity (DA) y knowledge sharing activity (KA) respectivamente. Primero calcula un ranking de cada usuario por separado considerando commits, fork y watch en Github de cada usuario al que llama $rank_{DA}$. Luego crea otro ranking considerando las preguntas y tags relacionados a cada usuario y lo llama $rank_{KA}$. Finalmente

crea un ranking híbrido que considera ambas aproximaciones considerando un peso W para cada una, quedando el ranking total en la Eq. (4.1).

$$rank = KA * W_{KA} + DA * W_{DA} \tag{4.1}$$

Originalmente se utiliza el coeficiente $\frac{W_{DA}}{W_{KA}} = \frac{0,75}{0,25}$, que otorga más importancia al conocimiento teórico y esto es por que el objetivo de DevRec es recomendar expertos para desarrollar software. Sin embargo, el objetivo del proyecto de esta memoria es recomendar expertos para responder preguntas por lo que se modificará ese coeficiente a uno que prefiera el conocimiento técnico.

4.2.1. Procesar datos

Para el algoritmo se deben manejar matrices que por ejemplo relacionan todos los tags T con todos los usuarios U, luego el tamaño de la tabla será $|U| * |T|$ que según los datos recolectados hasta ahora es 21388×14253. El algoritmo calcula cada relación y debe localizar la coordenada donde se relaciona un usuario con un tag, puesto que es una matriz grande, antes de comenzar el calculo se mapean los tags y los usuarios asignándoles un índice dentro de la matriz. Puesto que cada usuario y tag tiene un índice único en la matriz, primero se relaciona cada identificador de un usuario a un índice en la matriz, luego se hace lo mismo para tags.

4.2.2. Aplicar algoritmo

A continuación se describe la implementación del algoritmo, esta se divide en 7 pasos.

1. **Matriz R_{u-p}:** En este paso se crea la matriz de asociación de usuarios y proyectos R_{u-p} basado en su actividad en Github. La matriz se completa con 1 si el usuario u participa en el proyecto p y 0 si no. La Figura 4.4 incluye la forma en que se calcula R_{u-p}. Donde se procede a completar la matriz con 1 o 0 según corresponda.

2. **Matriz R_{u-u} DA:** En este paso se calcula la relación entre usuarios basado en la matriz R_{u-p}. Esta se calcula usando el algoritmo de Jaccard [37].

3. **Puntaje DA:** Ahora se calcula el puntaje basado en la contribución de un usuario u a un proyecto p. Esto se calcula utilizando la matriz R_{u-u} y la Eq. (2.1).

4. **Matriz R_{u-t}:** Esta matriz se calcula con el método TD-IDF. Sea $Uu_1, u_2, ..., u_n$ los usuarios en StackOverflow, $T_u = \{t_1, t_2, ..., t_n\}$ los tags relacionados al usuario u y $C(t, u)$ el número de veces que el tag t se relaciona con el usuario u. Entonces la matriz se calcula usando la Eq. (4.2).

```java
// <identificador, indice en la matriz>
private final Map<Integer, Integer> projects = ...;
private final Map<Integer, Integer> users = ...;

for (int userId : users.keySet()) {
    // indice en la matriz
    int userIndex = users.get(userId);
    // identificadores de proyectos en que el usuario colabora
    List<Integer> collaborations = new FetchIndexProjectsByUser(
            userId
    ).list();
    for (int projectId : collaborations) {
        // indice en la matriz
        int projectIndex = projects.get(projectId);
        // usuario particpa en proyecto
        rup[userIndex][projectIndex] = 1;
    }
}
```

Figura 4.4: Cálculo de la matriz R_{u-p}

$$R_{u-t}(u,t) = \frac{C(t,u)}{\sum_{\mathrm{i}=1}^{|T_u|} C(T_u[\mathrm{i}],u)} \times log(\frac{\sum_{k=1}^{|U|}\sum_{q=1}^{T_{U[k]}} C(T_{U[k]}[q],U[k])}{\sum_{j=1}^{|U|} C(t,U[j])}) \tag{4.2}$$

Puesto que se debe calcular la relación entre tag y usuario para cada usuario y tag, la Eq. (4.2) se calculó por partes para evitar calcular más de una vez el mismo resultado. Se pueden identificar 4 valores que se repiten.

(a) $C(t,u)$: Cantidad de veces que se relaciona el tag t con el usuario u.

(b) $C(T_u[\mathrm{i}],u)$: Para cada tag t relacionado al usuario u, cuenta la cantidad de veces que se relacionan.

(c) $C(t,U[j]$: Para un tag t, la cantidad de veces que se relaciona con cada usuario $U[j]$.

(d) $\sum_{k=1}^{|U|}\sum_{q=1}^{T_{U[k]}} C(T_{U[k]}[q],U[k])$: La suma de todos las relaciones de cada usuario u con sus tags t. Esta suma es constante por lo que se calcula una sola vez.

Por lo tanto, cada vez que se calcula una relación $C(t,u)$ se guarda para evitar calcularla otra vez. En la Figura 4.5 se ejemplifica el calculo de $C(t,U[j])$ con un tag fijo. Los demás casos se manejan de la misma manera.

5. **Matriz R_{u-u} KA:** Utilizando la matriz $R_{u,t}$ se calcula la relación entre usuarios según su preferencia en tags usando el algoritmo Vector Space Similarity.

6. **KA:** Finalmente se calcula KA utilizando la Eq. (2.1) utilizando la matriz R_{u-u} para KA.

7. **Guardar datos procesados:** Crear la matriz R_{u-u} para KA y DA toma aproximadamente 5 minutos. Luego para agilizar el proceso de calcular el ranking es que estas

matrices son guardadas en el disco luego de que son calculadas, de esta manera el calculo del ranking baja de 5 minutos a menos de 1 minuto.

```
// guarda la relacion C(t,u)
private final Map<Integer, Integer> mapFxt = new HashMap<>();
// calcula la relación C(t,u)
private double countWithFixedTag(int tagId) {
    // si ya fue calculada se retorna
    if (mapFxt.containsKey(tagId)) {
        return mapFxt.get(tagId);
    } else {
        // calcula la relacion(t,u)
        int count = new FetchTagCount(tagId).count();
        mapFxt.put(tagId, count);
        return count;
    }
}
```

Figura 4.5: Guarda resultados de la relación $C(t, U[j])$.

4.2.3. Listar recomendados

Se listan los resultados en una lista de `Aspirantes`. Como se aprecia en la Figura 3.4, estos aspirantes contienen el identificador de usuario en Github y el ranking. Esta lista se ordena por ranking.

4.3. Chequeo de algoritmo de recomendación

Como se muestra en la Figura 4.6, se implementa un Chequeo que recibe una forma de validar `Accuracy`, un algoritmo `Algorithm` a evaluar y una muestra de preguntas `TaggedItem` con las que se validará el algoritmo.

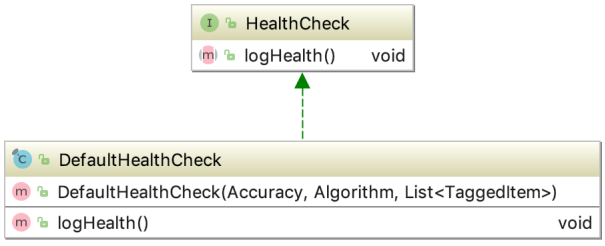

Figura 4.6: Implementación de Health Check

Este algoritmo no pretende predecir qué usuario obtendrá el voto de la mejor respuesta por parte del usuario que realiza la pregunta ya que eso depende de otros factores al conocimiento del aspirante como por ejemplo el mismo conocimiento del que realiza la pregunta ya que generalmente este va a considerar como mejor respuesta lo que le sea más útil en el momento. Es por esto que no se consideró cómo correcto tener a la mejor respuesta entre los aspirantes

con mejor ranking, si no más bien que en la primera mitad los usuarios al menos tengan la mitad de los tags relacionados a la pregunta.

4.4. Encuesta web

Esta encuesta web fue implementada en Java con Servlets y Jsp. Su función es enviar por correo solicitudes para responder una encuesta y almacenar las respuestas en una base de datos para ser analizadas. Se creó un repositorio en Github [30] donde se puede descargar el proyecto y realizar encuestas.

4.4.1. Configuración

Esta necesita de dos fuentes de datos, la primera es donde almacena las respuestas y lee las nuevas encuestas, llamémosla `survey.db`. La segunda es donde obtendrá la información como el título y contenido de la pregunta relacionada a cada encuesta, llamémosla `rosgh.db`.

Para obtener `survey.db` se debe crear el modelo de la Figura 3.7. Para ello se incluye en el proyecto un script (model) para crear el modelo, las instrucciones de cómo crearlo se incluyen en el `README.md` del proyecto. Luego para crear la base de datos simplemente se puede hacer con la línea de comandos si es que se instala SQLite:

```
$ sqlite3 survey.db
sqlite> .read {PATH_MODEL}/model
```

Luego se debe configurar el proyecto, donde se debe indicar la dirección de `survey.db`, `rosgh.db` y credenciales del correo que será utilizado para enviar encuestas por mail. Como se aprecia en la configuración:

```
# path to the sqlite db that has the information for survey
sqlite.path=

# path to sqlite v1.2.db that has the rosgh information
rosgh.db.path=

# mail configuration
user=noreply.rosgh.survey@gmail.com
pass=...
smtp.ssl.trust=*
smtp.host=smtp.gmail.com
smtp.port=587
smtp.auth=true
smtp.starttls.enable=true
```

Finalmente se debe hacer deploy de este proyecto en un servidor. Este proyecto fue probado

en Wildfly [38] y Tomcat [32]. Para estos dos casos sólo es necesario crear el `war` del proyecto usando el comando `mvn package`.

4.4.2. Contenido de la encuesta

Como se menciona en la sección 3.2.7, se envía un correo a cada usuario con una invitación a participar en la encuesta. El contenido del correo es el siguiente:

> *"Question-Answers sites like ROS Answers have grown in popularity and importance, this is why to find the accurate person or expert to answer a question has a big impact not only in these platforms but in all the industries that benefit of those sites as well. Currently, the are different researches about algorithms to recommend experts to answer questions. So we are surveying ROS Answers users to validate some algorithms that recommend users to answer ROS Answer's questions.*
>
> *Please participate in our survey, available at {URL-TO-SURVEY}. We anticipate it requiring 5 minutes (max) of your time. Our detailed consent form is available at http://survey.elbraulio.com*
>
> *Your participation is highly appreciated!*
>
> *Braulio López (University of Chile)"*

La primera URL (URL-TO-SURVEY) lleva a la encuesta dónde puede responder las consultas por cada pregunta asignada. Cómo se aprecia en la Figura 4.7, la encuesta incluye la pregunta de ROS Answers y un enlace a la pregunta original, luego incluye 4 consultas de las cuales 3 son obligatorias. Finalmente, se muestra un botón para continuar la encuesta y pasar a la siguiente pregunta de ROS Answers. La segunda URL lleva a una descripción del proyecto similar a la incluida en el correo enviado.

4.4.3. Enviar encuesta

Para realizar encuestas sólo hay que seguir estos 3 pasos:

1. Se deben agregar todos los aspirantes a expertos a los que se quiere encuestar a la tabla `aspirant`. Incluyendo `gh_user_id` y `ros_question_id` que se obtienen de la base de datos `rosgh.db`.

2. Agregar `gh_user_id` a la tabla `survey`. De esta forma, se indica que se desea enviar un correo a esta persona.

3. Se debe ir en el navegador a la dirección `host/sendmail`. Esto ejecutará el envío de correos. Por ejemplo, si se hace deploy en `localhost` se va a `localhost/sendmail`.

Para analizar los resultados, simplemente se debe ir a `survey.db` y verificar la tabla `answers`.

need topological map online, graph_slam doesn't work

Has anybody had success using the graph_slam package described at http://www.ros.org/wiki/graph_slam I tried checking out the source and building it but the dependencies do not build.

Or do you have any suggestions how I can build a topological map out of a SLAM gridmap fairly efficiently on the fly?

Edit: I am using the cturtle-pr2all stack

1 minute survey

(*) : answer required

Given the topic and question above. Do you have knowledge about the content of the question? (*)

Yes
No

Would you be willing to answer the question? (*)

Yes
No

This survey was sent according to your profile, collaborating with ROS in Github and ROS Answers. From 1 to 4, where 1 is very bad and 4 is excellent, how accurate do you think our prediction was about your knowledge to answer the question above? (*)

1
2
3
4

Please give us feedback about why you can answer or not the question above

Next

Figura 4.7: Ejemplo de encuesta. Disponible en http://survey.elbraulio.com/survey?id=42 (accedido: 06.01.2019)

Capítulo 5

Resultados

Al momento de validar el algoritmo de recomendación implementado en la sección 4.2, se utilizaron diferentes instrumentos que fueron creados para medir cuantitativamente los resultados, con el fin de obtener información específica del impacto. Además, parte de esta validación se obtuvo a través de retroalimentación por parte de los diversos expertos que aportan su conocimiento y habilidades en el desarrollo de software en ROS. A continuación, en la Sección 5.1 se explica cómo se realizaron pruebas a cada herramienta contenida en la memoria. Luego, en la sección 5.2 se evidencia los resultados que se obtuvieron durante la validación. Finalmente, en sección 5.3, se discuten los resultados evidenciados en este capítulo.

5.1. Diseño

En esta sección, se explica el diseño de las distintas validaciones que se aplicaron en este trabajo.

5.1.1. Extracción de datos

En el capítulo 3 se definió la forma para extraer datos de ROS Answers y Github en las secciones 3.2.1 y 3.2.2 respectivamente. De ROS Answers se espera obtener el perfil, preguntas y respuestas de al menos el 90 % de los usuarios. Por otro lado, se espera obtener la información sobre la participación de usuarios en paquetes de ROS en Github de al menos el 80 % de ellos. El porcentaje esperado de Github es más bajo puesto que no todos los paquetes listados en el repositorio de ROS están alojados en Github.

Puesto que la información sobre paquetes de ROS contenida en Github y la información sobre participación de usuarios en ROS Answers son independientes entre si, se dividió la extracción en dos procesos para ser ejecutados de forma paralela para disminuir el tiempo de extracción de datos. En el primero, se extrae información de ROS Answers, mientras que en el segundo se extrae información de Github.

Con el objetivo de extraer información de usuarios, tags, preguntas y respuestas de ROS Answers, se crea un script que utiliza el scraper (sección 2.2.4) y la Api de askbot (sección 2.2.1). Puesto que tanto el scraper como la Api necesitan conectarse reiteradamente a sus recursos, se estima que la extracción tomará del orden de horas. Luego se empaqueta en un `jar` para ser ejecutado en una máquina con disponibilidad de tiempo en linea de al menos 12 horas para completar la extracción en caso de que esta tome más de 5 horas para terminar.

De la misma forma en que se prepara el script para extraer datos de ROS Answers, se crea un script para extraer datos de Github utilizando su API (sección 2.2.5) y se empaqueta en un `jar`.

Toda esta información es almacenada en una base de datos SQLite.

5.1.2. Enlace de cuentas de usuarios

Luego de la extracción de datos, para cada cuenta en ROS Answers se debe identificar, si es que existe, una cuenta en Github. Este enlace de cuentas es necesario para perfilar el comportamiento de un usuario colaborando a ROS respondiendo preguntas en ROS Answers y colaborando en paquetes de ROS en Github.

Anteriormente se definió FRIL (sección 3.2.3) como la herramienta que será utilizada para crear este enlace. Para ello se exportó desde la base de datos SQLite a un `csv` la información que puede identificar a un usuario en ROS Anwers y en Github. De ROS Answers se exportó el nombre de cada usuario. Mientras que de Github se exportó el nombre, mail y login.

Luego se define qué datos van a ser comparados y a cada comparación se le asigna un peso. En la tabla 5.1 se detalla la comparación que se realizó y el peso que se asignó a cada una.

github	ros answers	peso
login	nombre	40
nombre	nombre	40
prefijo mail	nombre	20

Tabla 5.1: Enlace de cuentas FRIL

Con un peso mínimo de 20 se acepta el enlace. Es decir, si al menos una de las comparaciones es aceptada, se enlaza las dos cuentas. Esto es para obtener la mayor cantidad de enlaces posibles ya que es difícil lograr identificar todas las cuentas con tan poca información disponible en ROS Answers.

5.1.3. Chequeo de resultados de algoritmo

Para el chequeo del algoritmo, se utilizó como conjunto de datos las preguntas relacionadas a tags que han sido relacionadas a un repositorio en Github. Puesto que al trabajar con tags

no relacionados a proyectos o usuarios no relacionados a proyectos en Github, el chequeo no será completo, puesto que se medirá su rendimiento sólo respecto a la actividad en ROS Answers ya que los repositorios representan un porcentaje pequeño del conjunto de datos y estos pueden parecer despreciables a la hora de rankear usuarios. Es decir, puede pasar que el algoritmo sea bueno rankeando según KA y esto opaque el resultado de rankear junto a DA.

5.1.4. Encuesta web

Con el objetivo de verificar tanto el funcionamiento de la encuesta web como el mismo resultado del algoritmo de recomendación DevRec, se creó un servidor con Tomcat para alojar la encuesta web, se ejecutó el algoritmo de recomendación y se envió la solicitud para responder una encuesta por correo a los usuarios con mejor ranking para 3 preguntas en ROS Answers. Los tags de las preguntas corresponden a:

- map, graph, topological, SLAM, gridmap y online.
- marker, color, visualization, points y rviz.
- Kinect, colour, pointcloud, rviz y openni.

Por cada una de estas preguntas se ejecutó el algoritmo de DevRec y se tomó el 50 % mejor rankeado para enviar la encuesta.

5.2. Datos

En esta sección, se detallan los datos que se obtuvieron al ejecutar las validaciones diseñadas en la sección anterior.

5.2.1. Extracción de datos

Esta etapa se completó extrayendo: 1.679 repositorios de Github, 2.858 usuarios de Github, 21.388 usuarios de ROS Answers, 40.995 preguntas de ROS Answers y 14.253 tags en ROS Answers.

5.2.2. Enlace de cuentas de usuarios

Se logró enlazar un 73 % (2.109) de los usuarios obtenidos desde Github con usuarios en ROS Answers. Estos enlaces fueron incluidos en la base de datos SQLite.

5.2.3. Chequeo de resultados de algoritmo

A continuación, en la tabla 5.2 se presenta el resultado del chequeo a DevRec considerando la combinación $\frac{W_{KA}}{W_{DA}}$ utilizada por Zhang et al. y otras dos combinaciones que prueban el resultado aislando un tipo de conocimiento.

W_{KA}	W_{DA}	**precisión**	**sensibilidad**	**MRR**
0.75	0.25	0.718	0.252	0.693
1.0	0.0	0.718	0.252	0.708
0.0	1.0	0.468	0.180	0.156

Tabla 5.2: Resultados del chequeo de DevRec

5.2.4. Encuesta web

Como un usuario puede ser recomendado a responder más de una pregunta, el total de usuarios distintos recomendados es de 558. De estos usuarios que deben recibir mail, sólo 217 de ellos tiene registrado un mail.

A continuación, se muestran respuestas a las preguntas definidas en la sección 3.2.7. Los resultados son difíciles de analizar ya que tienen un bajo volumen (6 respuestas), además una de las personas que respondió es experta en Robótica [13] por lo que es muy probable que pueda tener conocimiento y responder la mayoría de las preguntas. En la tabla 5.3 se muestra las respuestas a la encuesta.

P1	**P2**	**P3**	**P4**
yes	yes	4	Expertise in robotics, PhD + 15 years in academia, teaching mobile robotics
yes	yes	4	Same as previous questions.
yes	yes	4	Same as before
yes	yes	4	
yes	yes	4	
yes	yes	4	

Tabla 5.3: Respuestas de expertos a la encuesta web.

5.3. Discusión

Los datos extraídos de ROS Answers fueron los esperados. Se recolectaron casi todas las preguntas y usuarios que existían a la fecha. Esto fue posible ya que toda esa información es publica y accesible al scraper utilizado. Sin embargo, algunas preguntas y respuestas no fueron rescatadas ya que están marcadas como "wiki", esto significa que cualquier usuario

puede mejorarla y no está ligada a ningún usuario en particular. Por otro lado, la extracción de los datos de repositorios y colaboradores fue difícil de manejar ya que no siempre estaba disponible el proyecto con el código fuente, que es el objetivo principal porque contiene información de los colaboradores de cada paquete. Ante la falta de estos repositorios se optó por explorar otros tipos de repositorios que contenían documentación relacionada a paquetes de ROS y recopilar información de usuarios desde allí.

El enlace de cuentas de usuarios obtuvo un alto porcentaje de aciertos, pero lamentablemente debido a la poca información disponible para identificar a un usuario en ROS Answers, esta contiene enlaces fallidos. Una mejora posible a este resultado es aportar más información para comparar y obtener menos enlaces erróneos.

Al probar el algoritmo, el cambio más notorio ocurre al usar $W_{KA} = 0$ y $W_{DA} = 1$. Esto hace pensar que DA es menos importante a la hora de recomendar a un usuario para responder una pregunta. Pero este bajo MRR puede ser la consecuencia de un mal enlace de cuentas de usuarios. Es decir, un usuario de Github con DA alto para la pregunta seleccionada puede quedar erróneamente enlazado a usuario de ROS Answers que no participó en la pregunta puesta a prueba, y como el MRR es medido buscando al usuario entre las mejores respuestas, si este no se encuentra entonces obtendrá un MRR bajo. Un caso interesante encontrado al hacer el chequeo del algoritmo es para el usuario de ROS Answers *akoz002*. Este tiene un alto KA al estar relacionado con el usuario *sachin_ chitta* de la misma plataforma. Esto porque *sachin_ chitta* tiene alta participación con distintos tags incluyendo tags en los que *akoz002* nunca ha participado. Además, el usuario *sachin_ chitta* tiene poca participación (pero la tiene) en el tag `pr2_controllers`. Entonces cuando KA es calculado para `pr2_controllers`, el usuario *sachin_ chitta* tiene un alto puntaje incluso cuando no tiene ninguna participación en el tag `pr2_controllers`. Esto ocurre porque está relacionado con un usuario que está muy bien rankeado para ese tag. De lo anterior se puede concluir que DevRec recomienda gente que no necesariamente está directamente relacionada a un tag, más bien recomienda usuarios que tienen intereses similares con otros usuarios que participan en los tags (el paper dice eso cuando asume que usuarios con competencias similares prefieren tags similares).

Capítulo 6

Conclusiones y trabajo a futuro

Este trabajo de memoria de título está destinado a servir de herramienta en la validación de algoritmos de recomendación abarcando la extracción y disponibilidad de datos, un ejemplo de implementación de un algoritmo de recomendación, la validación local realizada por el cliente más la validación en linea resuelta por los usuarios.

Se logró extraer datos de un sitio de preguntas y respuestas y enlazar esa información con otro sitio donde se aloja y versiona código de proyectos relacionados al sitio de preguntas y respuestas. Luego, se implementó para ROS Answers un algoritmo de recomendación diseñado originalmente para recomendar usuarios de StackOverflow para colaborar en proyectos en Github. Las diferencias de volumen de datos y de información disponible en ambas plataformas tuvo consecuencias en el tiempo de implementación de este algoritmo e impidió implementar un segundo. Sin embargo se consideró dos algoritmos distintos a la hora de diseñar las herramientas de chequeo y encuesta web. La herramienta de chequeo mide precisión, sensibilidad y MRR, mientras que la encuesta web consulta directamente a los usuarios recomendados por la exactitud de la recomendación. También se realizó una encuesta web con invitaciones por mail para participar en ella, esta validó la implementación del algoritmo de recomendación sin embargo tuvo baja participación.

Todas estas herramientas facilitan información y validación a futuros investigadores de algoritmos de recomendación. No solo en ROS Answers, si no que también puede ser utilizado para validar algoritmos StackOverflow por ejemplo, puesto que fue diseñado programando orientado a Interfaces lo que facilita la extensión de este. No así la extracción de datos que es especializada a ROS Anwers.

Para incluir un algoritmo de recomendación, se necesita que este sea implementado en Java y que implemente la interfaz diseñada en esta memoria `Algorithm`. Esto le permite ejecutar el chequeo a este nuevo algoritmo utilizando las fuentes de datos provistas en esta memoria. Para la recolección de datos, se puede utilizar el scraper incluido en la memoria o alguna fuente externa de datos ya que es la implementación del algoritmo quien decide de qué manera obtendrá los datos.

Un aspecto importante a destacar fue el uso de herramientas CI: Travis, codecov y code-

beat. Puesto que si bien al comienzo suponen un esfuerzo para configurar la conexión con el proyecto, luego influyeron positivamente en el desarrollo de la memoria. Esto se evidenció una vez el proyecto tomó forma y acumuló una gran cantidad de código, ya que esas herramientas sirven para verificar que los cambios en el código no perjudiquen el proyecto. Por otro lado, se publicó en Jitpack un artefacto que contiene el proyecto para facilitar el uso de las herramientas creadas en esta memoria en cualquier otro proyecto que se maneje con maven o gradle, simplemente incluyéndolo en la lista de librerías.

Gran parte del tiempo del proyecto se invirtió en la implementación de DevRec y la validación. Inicialmente se propuso la implementación de dos algoritmos de recomendación puesto que ambos algoritmos fueron publicados en papers y se esperó que existiera la información suficiente para replicarlos. Sin embargo, en la publicación de DevRec no se detallan algunos puntos importantes que luego impactaron fuertemente al momento de la implementación como por ejemplo la forma en que se relacionan los tags con proyectos en Github, por otro lado sumó dificultad el hecho de que fuera orientado a otra plataforma que es mucho más popular y por ende contiene muchos más datos para validar como por ejemplo los mails de los usuarios. Aun así, es un algoritmo interesante puesto que considera dos tipos de conocimientos de dos fuentes de datos distintas para rankear y recomendar usuarios.

Como trabajo a futuro es valioso incluir un enlace automático de cuentas de usuarios entre ROS Answers y Github sin utilizar FRIL ya que este paso manual dificulta la automatización de todo el proceso desde que se recolecta el dato hasta que se realiza el chequeo. Además, contar con distintas implementaciones de algoritmos de recomendación promovería la utilización de este trabajo como herramienta para validar nuevos algoritmos y de esta forma, por que no, definir un estándar en la validación de estos trabajos.

Capítulo 7

Bibliografía

[1] Askbot. Sitio oficial. https://askbot.com (accedido: 06.09.2018).

[2] M. Baker. 1,500 scientists lift the lid on reproducibility. https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970 (accedido: 06.09.2018).

[3] Bitbucket. Sitio oficial. https://bitbucket.org (accedido: 02.01.2019).

[4] D.M. Blei, A.Y. Ng, M.I. Jordan, and J. Lafferty. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:55–68, 2003.

[5] Y. Bugayenko. Recomendaciones de y. bugayenko para un proyecto open source. https://www.yegor256.com/2018/05/08/open-source-attributes.html (accedido: 02.01.2019).

[6] Codebeat. code quality. https://codebeat.co (accedido: 02.01.2019).

[7] Codecov. test coverage. https://codecov.io (accedido: 02.01.2019).

[8] FRIL. Sitio oficial. http://fril.sourceforge.net/ (accedido: 20.12.2018).

[9] Gitana. Issue creado en el repositorio de gitana por integración con ros answers. https://github.com/SOM-Research/Gitana/issues/33 (accedido: 02.01.2019).

[10] Gitana. Repositorio oficial en github. https://github.com/SOM-Research/Gitana (accedido: 02.01.2019).

[11] Github. Api github. https://developer.github.com/v3/ (accedido: 02.01.2019).

[12] Github. Mysql dump que incluye información de repositorios y usuarios en github. http://ghtorrent.org/downloads.html (accedido: 06.09.2018).

[13] Github. Perfil de cedric pradalier. https://github.com/cedricpradalier (accedido: 02.01.2019).

[14] Github. plataforma de desarrollo de software. https://github.com/ (accedido: 02.01.2019).

[15] Jcabi-github. herramienta para consultar la api de github. https://github.com/jcabi/jcabi-github (accedido: 20.12.2018).

[16] Jsoup. Sitio oficial jsoup. https://jsoup.org (accedido: 02.01.2019).

[17] MySQL. Sitio oficial. https://www.mysql.com (accedido: 02.01.2019).

[18] Perceval. Documentación perceval. https://perceval.readthedocs.io/en/latest/perceval.backends.core.html#module-perceval.backends.core.askbot (accedido: 02.01.2019).

[19] Perceval. Repositorio perceval en github. https://github.com/chaoss/grimoirelab-perceval (accedido: 02.01.2019).

[20] ROS. Indicadores 2018. http://download.ros.org/downloads/metrics/metrics-report-2018-07.pdf (accedido: 02.01.2019).

[21] ROS. Repositorio de releases. https://github.com/ros/rosdistro (accedido: 06.09.2018).

[22] ROS. Sitio oficial de la plataforma. http://www.ros.org (accedido: 06.09.2018).

[23] ROS Answers. Ejemplo de pregunta con usuarios no registrados en ros answers. https://answers.ros.org/question/9083/ros-lego-nxt-kinect/ (accedido: 02.01.2019).

[24] ROS Answers. Lista de usuarios registrados. https://answers.ros.org/users/ (accedido: 20.12.2018).

[25] ROS Answers. Sitio oficial de preguntas y respuestas para ros. https://answers.ros.org/ (accedido: 06.09.2018).

[26] rosgh. Repositorio del recomendador de expertos para ros answers. https://github.com/elbraulio/ros_gh (accedido: 02.01.2019).

[27] SQLite. Sitio oficial. https://www.sqlite.org/index.html (accedido: 02.01.2019).

[28] Stack Exchange. Dump que incluye información del sitio stackoverflow. https://archive.org/details/stackexchange (accedido: 06.09.2018).

[29] StackOverflow. Sitio de preguntas y respuestas. https://stackoverflow.com (accedido: 06.09.2018).

[30] Survey. herramienta basica para enviar encuestas por correo. https://github.com/elbraulio/simple_research_survey (accedido: 02.01.2019).

[31] Y. Tian, P. S. Kochhar, E. Lim, F. Zhu, , and D. Lo. Predicting best answerers for new questions: An approach levebenja.rraging topic modeling. *Science of Computer*

Programming, pages 55–68, 2014.

[32] Tomcat. implementación open source para servlets y jsp. `https://tomcat.apache.org` (accedido: 02.01.2019).

[33] Travis. Sitio oficial. `https://travis-ci.org` (accedido: 02.01.2019).

[34] Trello. Sitio oficial. `https://trello.com` (accedido: 20.12.2018).

[35] Wikipedia. Definición de tf-idf. `https://en.wikipedia.org/wiki/Tf\T1\textendashidf` (accedido: 06.09.2018).

[36] Wikipedia. Función de hash md5. `https://en.wikipedia.org/wiki/MD5` (accedido: 06.09.2018).

[37] Wikipedia. Jaccard index. `https://en.wikipedia.org/wiki/Jaccard_index` (accedido: 02.01.2019).

[38] Wildfly. Ejecución de aplicaciones. `http://wildfly.org` (accedido: 02.01.2019).

[39] X. Zhang, T. Wang, G. Yin, C. Yang, Y. Yu, and y H. Wang. DevRec: A Developer Recommendation System for Open Source Repositories. *Lecture Notes in Computer Science*, 10221, 2017.