Modelado colaborativo en lenguaje natural a través de redes sociales

Sara Pérez Soler

Máster en Investigación
e Innovación en TIC

MÁSTERES
DE LA UAM
2017 - 2018

Escuela Politécnica Superior

UAM Universidad Autónoma de Madrid

Universidad Autónoma de Madrid

Escuela Politécnica Superior

TRABAJO FIN DE MASTER

MODELADO COLABORATIVO EN LENGUAJE NATURAL A TRAVÉS DE REDES SOCIALES

Máster Universitario en Investigación e Innovación en TIC

Sara Pérez Soler
Tutora: Esther Guerra Sánchez
Tutor: Juan de Lara Jaramillo
Departamento de Ingeniería informática

20 de Junio del 2018

MODELADO COLABORATIVO EN LENGUAJE NATURAL A TRAVÉS DE REDES SOCIALES

Autor: Sara Pérez Soler
Tutora: Esther Guerra Sánchez
Tutor: Juan de Lara Jaramillo

Grupo de Modelado e Ingeniería del software(MISO)
Departamento de Ingeniería informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

20 de Junio del 2018

Abstract

Abstract — Model Driven Engineering (MDE) automates software development by promoting models as the main assets in software projects. Models are actively used throughout the software life cycle, as design elements, to simulate, validate, test, maintain and generate code for the final application. This way, models becomes as a fundamental part of development, and are essential to ensure the quality of software. Thus, not only the involvement of modelling experts is necessary, but also the domain experts have an important role to play in development of the domain models.

Modelling is a collaborative task performed between modelling experts and domain experts. Therefore, the tools are important to facilitate collaborative task. They must provide collaboration support and management, discussion and coordination mechanisms.

Currently, social networks have gained an enormous prominence in our daily life. They provide a lightweight, agile, ubiquitous mechanism for discussion, coordination and dissemination of information tasks. Not only general propose networks, but also many social networks like Workplace by Facebook, Slack or Yammer, have emerged to meet this need in enterprises. In software engineering, the use of social networks has been adopted as a coordination and collaboration mechanism. Social networks such as Stackoverflow have emerged to help developer communities to share and learn from each other. In addition, the advance in the processing of natural language (NL) has allowed bots or chatbots to emerge. These are programs whose interaction mechanism is based on NL, and can be used to obtain information or automate tasks.

Given the benefits of social networks, this work proposes SOCIO, a bot, which works over social networks, to assist in collaborative modelling tasks. SOCIO provides modelling support interpreting requirements in NL. Thus it facilitates the modelling task to participants with low technical experience, like domain experts. SOCIO has an extensible architecture that supports different social networks. Currently, the bot is available on Telegram and Twitter. Finally, the tool has been evaluated in a preliminary study on Telegram with very promising results, which encourages us to continue working on this approach and improve natural language processing.

Key words — Collaborative modelling, social network, chatbots, natural lenguage.

Resumen

Resumen — El desarrollo dirigido por modelos (MDE) automatiza el desarrollo del software y tiene como pieza central los modelos. Estos, además de ser elementos de diseño del sistema, se usan durante todo el ciclo de vida del sistema para validar, simular, probar, mantener y generar código de la aplicación final. Los modelos se convierten en una pieza fundamental, y es esencial para garantizar la calidad del software, su adecuada construcción. Para eso, no solo es necesaria la participación de los expertos en modelado, sino que, además los expertos de dominio juegan un papel muy importante en el desarrollo de modelos de dominio. Nace así la necesidad de modelado colaborativo entre expertos de dominio y expertos de modelado.

Para facilitar las tareas de modelado colaborativo, las herramientas juegan un papel muy importante. Y deben dar soporte, no solo al modelado, sino también a labores de gestión, comunicación y coordinación.

Por otro lado, las redes sociales han ganado importancia durante los últimos años. Han demostrado una gran eficacia en tareas de discusión, de coordinación, y de diseminación de información. Tanto es así, que no solo se han quedado en el ámbito personal, sino que muchas redes sociales han nacido para satisfacer estas necesidades en el ámbito laboral, como Workplace by Facebook, Slack o Yammer. En la ingeniería del software el uso de las redes sociales se ha adoptado como mecanismo de coordinación y colaboración. Redes Sociales como Stackoverflow, han surgido para que comunidades de desarrolladores compartan y aprendan unos de otros. Por otro lado, el avance en el procesamiento de lenguaje natural (LN) ha permitido que surjan bots o chatbots, programas con los que se interactua a través de LN y que permiten obtener información o automatizar tareas.

Aprovechándose de las ventajas que ofrecen las redes sociales para comunicación y coordinación, en este trabajo se propone SOCIO, un bot que funciona sobre redes sociales para realizar tareas de modelado colaborativo. SOCIO proporciona soporte para modelado, interpretando requisitos expresados en LN, acercando de esta manera el modelado a participantes con poca experiencia en esa área, como pueden ser expertos de dominio. El bot mantiene un historial de mensajes que ayuda a mantener la trazabilidad del modelo.

SOCIO tiene una arquitectura fácilmente extensible a diferentes redes sociales. Actualmente, el bot está disponible en Telegram y Twitter.

Por último, la herramienta ha sido evaluada en un estudio preliminar sobre Telegram

con resultados muy prometedores, que nos animan para continuar trabajando sobre este enfoque y mejorar el procesamiento de lenguaje natural.

Palabras clave — Modelado colaborativo, redes sociales, chatbots, lenguaje natural.

Índice general

Índice de tablas

Índice de figuras

1 Introducción

En esta Sección hablaremos de las motivaciones para realizar este trabajo (apartado 1.1) y de la organización de este documento (apartado 1.2).

1.1. Motivación

La aparición de los lenguajes de programación supuso un mayor nivel de abstracción en el desarrollo de software, puesto que permiten expresarse con instrucciones de más alto nivel que equivalen a varias instrucciones máquina. Del mismo modo, el desarrollo dirigido por modelos (MDE, de sus siglas *Model-driven Engineering*) eleva nivel de abstracción con respecto a los lenguajes de programación. Esto permite automatizar el desarrollo de software, obteniendo productos de calidad reduciendo el coste [1]. Los modelos de alto nivel, que se pueden representar mediante una sintaxis gráfica o textual, además de usarse para especificar el sistema en sí, se usan para validar, simular, probar, mantener y generar código para la aplicación final. Por lo tanto, en MDE el modelo formar parte de todo el ciclo de vida del producto [2]

En el desarrollo de software [3], la colaboración es una necesidad, y en la actualidad, con la globalización se presentan nuevos retos en este área, para hacer posible la colaboración desde cualquier parte del mundo, y con participantes heterogéneos [4]. En el modelado, está necesidad se acentúa, pues no solo es necesario que los desarrolladores y los expertos en modelado colaboren, sino que además, en sistemas de dominios concretos, es necesario involucrar a los expertos de esos dominios en el desarrollo del modelo, para asegurar que el sistemas cumpla, de la forma más efectiva, los requisitos [5][6]. Sin embargo, uno de los factores limitantes en el desarrollo dirigido por modelos, actualmente, es el escaso soporte por parte de las herramientas para la colaboración [6].

Para mejorar esta situación, algunos investigadores han identificado aspectos clave de los entornos de modelado colaborativo [7] como son:

- **Gestor del modelo**: un entorno de modelo distribuido para gestionar el ciclo de vida de los modelos, en el que participantes de diferente tipo (desarrolladores, expertos de modelado, expertos de dominio) editan el modelo (o parte de él), posiblemente en tiempo real. Los autores proponen que se pueda acceder al modelo siguiendo una experiencia multidispositivo, que facilita la colaboración en movilidad.
- **Mecanismos de colaboración** para permitir que los participantes en la tarea de modelado trabajen en los artefactos de modelado en colaboración.
- **Medios de comunicación** para permitir que las partes involucradas estén al tanto de lo que están haciendo las otras partes interesadas que colaboran con ellos. Estos mecanismos deben garantizar la trazabilidad entre las decisiones de diseño discutidas en los contenidos orientados a la comunicación y los artefactos de modelado.

Por otro lado, la redes sociales han ido ganando fuerza en los últimos años, gracias a sus posibilidades de comunicacion, sin importar la distancia, para organizar incluso a grandes grupos y difundir información a un gran número de personas en poco tiempo. Pero estas capacidades no solo se han aprovechado en el terreno social, sino que, cada vez son más las redes sociales dedicadas a labores de comunicación y coordinación en el ámbito laboral, como Workplace by Facebook[1] o Yammer[2]. En concreto, en la ingeniería del software las redes sociales han cambiado la manera de trabajar [8]. Es común que se creen comunidades de desarrolladores para colaborar y aprender unos de otros dentro de las redes sociales o redes sociales dedicadas a este fin, como es el caso de Stackoverflow[3] [9] o que los equipos de trabajo se coordinen mediante una red social, como Slack[4] [10]. Además, el avance en el procesamiento del lenguaje natural (LN) ha permitido el aumento de *bots* o *chatbots* dentro de las redes sociales, con los que se puede interactuar y que responden imitando las respuestas que podría dar una persona. Los usuarios usan los bots, por ejemplo, para programar el envío de recordatorios o para la automatizar tareas, como pueden ser la creación de documentación [11] o el análisis de proyectos software [12].

En este trabajo, se va a desarrollar un chatbot sobre redes sociales, que asistirá en la tarea de modelado colaborativo. Al funcionar sobre redes sociales, el chatbot será accesible desde múltiples dispositivos y facilitará la colaboración en movilidad y en tiempo real. Como sistema de comunicación entre los participantes se usará el propio de las redes sociales. Para trabajar con el chatbot en la tarea de modelado se usará LN para expresar los requisitos, que el chatbot interpretará para crear el modelo.

Nuestro enfoque se aleja de las soluciones más convencionales, en las que se trabaja con entornos de modelado gráficos o por comandos y sistemas de colaboración tradicionales. Este nuevo enfoque ofrece algunas ventajas con respecto a los sistemas tradicionales, como

[1] https://www.facebook.com/workplace
[2] https://www.yammer.com/
[3] https://stackoverflow.com/
[4] https://slack.com

por ejemplo que no hay necesidad de instalar ni aprender el manejo de herramientas nuevas, pues las redes sociales ya son usadas por la mayoría de las personas, o que facilita la interacción con personas con poco conocimiento de modelado al usar LN en lugar de los entornos de modelado gráficas o por comandos.

Este trabajo ha sido financiado por la Comunidad de Madrid con su plan de empleo juvenil (PEJD-2016-TIC3121). Además, ha dado lugar a dos publicaciones:

1. Sara Pérez-Soler, Esther Guerra, Juan de Lara, Francisco Jurado. ***The rise of the (modelling) bots: towards assisted modelling via social networks***. Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp.: 723-728 (Conferencia Core A) (porcentaje de aceptación del 22 %). Octubre 2017, Illinois (USA).

2. Sara Pérez-Soler, Esther Guerra, Juan de Lara. ***Assisted Modelling Over Social Networks with SOCIO***. Proceedings of ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems, Satellite Events, 2017, pp.: 561-565 (CEUR). (Conferencia Core A). Septiembre 2017, Austin (USA). Conferencia a la que asistí a presentar la publicación

1.2. Estructura del documento

A continuación se va a definir la estructura del documento. La Sección 2 contiene el estado del arte que se divide en dos subsecciones. La primera habla de las herramientas de modelado colaborativo 2.1 y la segunda de los bots utilizados para el desarrollo 2.2. La Sección 3 plantea el enfoque seguido (Sección 3.1) y muestra un ejemplo de creación de un modelo (Sección 3.2). La Sección 4 expone la arquitectura de la herramienta desarrollada a partir del enfoque (Sección 4.1), y se explica el funcionamiento de dicha herramienta en Telegram (Sección 4.2.1) y Twitter (Sección 4.2.2). En la Sección 5 se encuentra una evaluación del enfoque y de la herramienta. Y por último en la Sección 6 están las conclusiones y el trabajo futuro.

2

Estado del arte

En esta Sección presentamos una panorámica de las herramientas de modelado colaborativo(Sección 2.1), así como del uso de bots en ingeniería del software (Sección 2.2).

2.1. Herramientas de modelado colaborativo

El modelado colaborativo puede ser *offline* u *online*. La colaboración *offline* se realiza de forma asíncrona. Los usuarios obtienen el estado del modelo de un sistema de control de versiones, realizan cambios de forma local y lo vuelven a subir al sistema [13]. La colaboración *online* es síncrona. Los usuarios se reúnen en sesiones de colaboración. Simultáneamente editan el modelo y los cambios se propagan instantáneamente al resto de usuarios. En este apartado se analizan siete herramientas de modelado colaborativo, dos de las cuales son para modelado *offline*, cuatro *online* y una da soporte al modelado *offline* y *online*.

- DiCoMEF [14] es una herramienta de modelado colaborativo *offline*. Es un framework de Eclipse en el que cada usuario tiene una copia local del modelo y trabajan de forma asíncrona. Los usuarios tienen un rol dentro de un modelo (coordinador, editor y observador). La herramienta proporciona un sistema de *merge* o mezclado de modelos para resolver conflictos.

- AWMo [15] es una herramienta de modelado colaborativo *online*. La herramienta es una aplicación web, en la que varios usuarios, de forma simultanea, se pueden conectar y editar el mismo modelo. Tiene dos vistas para editar el modelo, una gráfica y otra textual. Los usuarios de la herramienta pueden trabajar en la vista

que resulte más cómoda. Los autores de la herramienta la recomiendan para que personas con discapacidad visual colaboren con personas sin esta discapacidad.

- Mondo [16] es una herramienta de modelado colaborativo, que proporciona soporte tanto para modelado *offline*, como modelado *online*. La herramienta usa un servidor de Subversión (SVN) para el control de versiones en el enfoque *offline*. En este servidor quedan también registrados todos los cambios *online*. Las contribuciones que ofrece la herramienta en su enfoque son: que tiene control de acceso a los modelos a nivel de elemento, se pueden bloquear los elementos del modelo y ofrece un sistema de *merge* o mezclado de los modelos automático.

- AToMPM [17] es una herramienta de modelado colaborativo *online*. En ella, por medio de una página web y un entorno de modelado gráfico, varios usuarios de forma simultanea pueden editar un modelo o parte de él. Además del entorno de modelado gráfico, la herramienta soporta el modelado a través de comandos.

- 3D UML [18] es una aplicación que proporciona una herramienta de modelado *online*. El modelado se realiza utilizando un editor gráfico de UML, con la peculiaridad de que los diagramas UML están en capas que puede conectase entre sí. La aplicación proporciona información acerca de la parte del modelo en la que está trabajando cada usuario y que cambios ha realizado.

- SCCMT [19] es una aplicación web que proporciona colaboración indirecta. Los usuarios, partiendo de su propio conocimiento del dominio elaboran un modelo propio. Este se fusiona junto con el resto de usuarios para crear un modelo colectivo. Después se produce una retroalimentación que inspira a los usuarios a mejorar su modelo en función del resultado del modelado colectivo

- WebGME [1] es una aplicación web que proporciona una herramienta de modelado *online*. La herramienta consta de una interfaz gráfica para editar los modelos. La aplicación también proporciona unos puntos de extensión para poder personalizarla, permite desde añadir *plugins* o complementos hasta reemplazar por completo la interfaz de usuario.

De los aspectos clave que los entornos de modelado colaborativo deben tener (ver la Sección 1.1), las herramientas aquí descritas centran sus esfuerzos en proporcionar un gestor de modelo. Algunas de ellas, como DiCoMEF, proporciona mecanismos de coordinación, permitiendo dar roles a los participantes. Pero en ninguna de ellas se menciona los mecanismos de discusión. Por otro lado, la edición del modelo en todas, es por medio de una editor gráfico, y dos de ellas permiten además la edición con comandos textuales. Tanto para el uso de un editor gráfico como los comandos textuales es necesario tener conocimiento sobre modelado, lo que podría dificultar el acceso de expertos en dominio por falta de experiencia.

[1] https://webgme.org/

2.2. Chatbots en la ingeniería del software

El avance en procesamiento de LN ha hecho que en los últimos años los bots o chatbots hayan ido ganando importancia. La interacción por medio de LN con el usuario resulta más natural para las personas, por lo que muchas empresas, como Google o Apple, están adoptando el uso de los bots para ampliar sus servicios (el asistente de búsqueda de Google y el asistente personal Siri de Apple). En el desarrollo de software, los bots han provocado cambios en la forma de trabajar. Con ellos se pretende obtener un aumento de la productividad, automatizando tareas. Un estudio realizado por Storey y Zagalsky [20] clasifica los bots en función de la fase del ciclo de vida del software en el que presta servicios. Los bots a continuación mencionados funcionan en Slack o para el sistema de Atlassian Hipchat[2], (red social de mensajería instantánea para grupos de trabajo).

- Bots para facilitar la codificación. Existen diversas maneras de hacer que la codificación sea más productiva. Por ejemplo con bots que responden preguntas de codificación, como SlackOverflow[3], un chatbot que en permite realizar preguntas en Stackoverflow desde Slack, que mantienen un informe de errores, como BugBot[4] para Slack, o que a partir de grandes cambios en el código crean una rama o branch en los sistemas de control de versiones.
- Bots de pruebas tanto para análisis estáticos, como el bot Code Climate[5] de Slack , como dinámicas, como Codeship[6] de Slack.
- Bots para ejecutar código en diversos lenguajes. Tan solo es necesario enviar el fragmento de código y el bot responde con el resultado de la ejecución. Algunos ejemplos en Slack son: CodeBot[7] para ejecutar JavaScript o RunBot[8] para Python.
- Bots de soporte. Algunos servicios de atención al cliente como ZenDesk[9] o Intercom[10] tienen integración con Slack a través de los bots. Esto permite agilizar la tarea de dar soporte de atención al cliente.

Estos son solo una pequeña porción de la posibilidad que nos ofrecen los bots. Podemos encontrar bots que realicen tareas tan sencillas como programar un recordatorio, como Wonder[11] en Slack, a bots que a partir de código generen diagramas de flujo, como code2flow[12] en Atlassian. El Slack las herramientas funcionan como chatbots, mientras que en Atlassian se usan bots. Los chatbots de Slack funcionan por medio de comandos.

[2]https://es.atlassian.com/software/hipchat
[3]https://github.com/karan/slack-overflow
[4]https://slack.com/apps/A0ERKPDGV-bugbot
[5]https://slack.com/apps/A0F81R5AB-code-climate
[6]https://slack.com/apps/A0F81FKT8-codeship
[7]https://slack.com/apps/A2YG4UA3A-codebot
[8]https://slack.com/apps/A29FG0WF2-runbot
[9]https://slack.com/apps/A9WFQ3M0B-zendesk
[10]https://slack.com/apps/A40CLA0DP-intercom
[11]https://slack.com/apps/A2AEA936K-wonder
[12]https://marketplace.atlassian.com/apps/1211531/code2flow-for-jira?hosting=server&tab=overview

Además hay que remarcar que no se ha encontrado ningún bot o chatbot que permita realizar modelos de dominio.

3 Enfoque

En este apartado se explicará el enfoque seguido en este trabajo. En las Sección 3.1 se explicará de forma general este enfoque, y en las distintas subsecciones (3.1.1, 3.1.2, 3.1.3) se explicará con mas detalle. Por último en la Sección 3.2 se mostrará un ejemplo de creación de un modelo.

3.1. Planeamiento general

En este TFM, basándonos en la motivación expuesta en la Sección 1.1, proponemos un sistema de modelado colaborativo interpretando requisitos en LN sobre redes sociales. El sistema se aprovecha de la inherente capacidad de comunicación y coordinación de las redes sociales. Además, como estas son muy usadas hoy en día, en la mayoría de los casos, no es necesario instalarse ni aprender el funcionamiento de una herramienta nueva. En la tarea de modelado se podrán involucrar un gran número de personas, y gracias al uso del LN para crear el modelo no hace falta que tengan mucha experiencia en las tareas de modelado.

Para asistir en estas tareas, un bot, llamado SOCIO, interpreta los requisitos expresados por los usuarios, usando LN, para crear un modelo de dominio (diagramas de clase sin operaciones). Los modelos se almacenan dentro de un proyecto, que pueden tiener tres niveles de visibilidad: *Public* (accesible para todo el mundo), *Protected* (accesible a nivel de lectura a todo el mundo, para escritura es necesario tener permisos) y *Private* (hacen falta permisos para acceder al proyecto). Cada proyecto pertenecerá a un usuario, que será el administrador del proyecto. SOCIO no necesita un proceso de registro explícito, sino que recoge la información de los usuarios de la red social. Para facilitar la tarea de modelado, también dispondrá de funcionalidades para validar el modelo, gestionar los

usuarios que participan y guardar la trazabalidad del modelo.

Para este trabajo el bot estará operativo sobre dos redes sociales, Twitter[1], una red social de micro-blogging, en la que los usuarios tienen 280 caracteres para publicar estados, y Telegram[2], una aplicación de mensajería instantánea que puede usarse en dispositivos móviles, ordenadores o a través de su página web.

La Figura 3.1 muestra como se desarrolla la interacción entre los usuarios y SOCIO. Los usuarios a través de la red social de su preferencia podrán discutir y organizarse. En el momento que ellos elijan, podrán dirigir mensajes al bot. La manera de dirigir los mensajes al bot varía según la red social, por ejemplo, en Telegram los mensajes al bot irán precedidos por "/", mientras que en Twitter, para dirigirse al bot es necesario mencionar su alias (@ModellingBot).

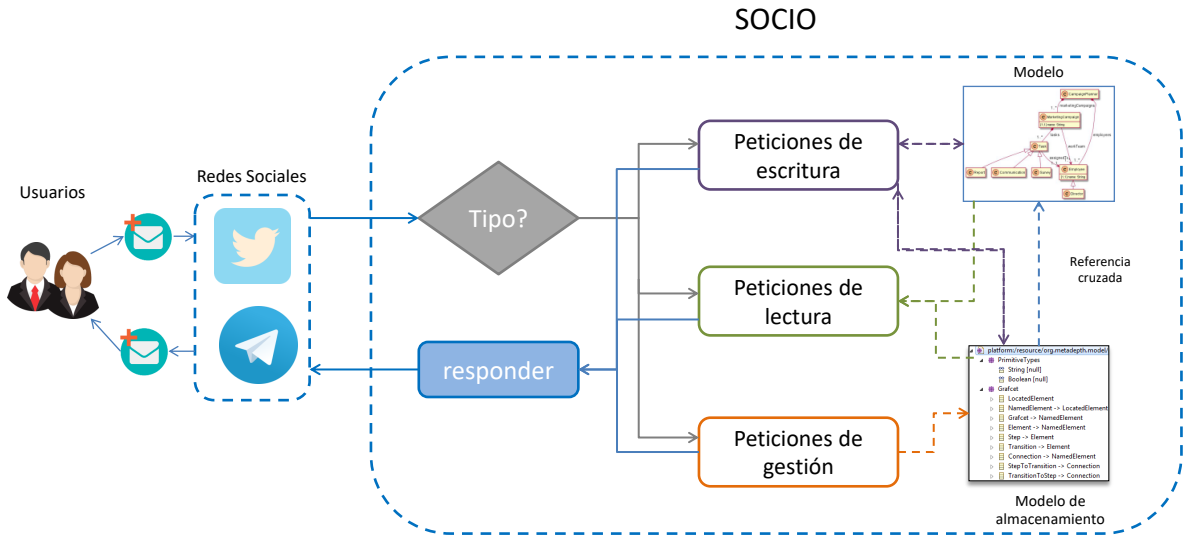


Figura 3.1: Interacción con SOCIO

Los mensajes enviados al bot pueden provocar tres tipos de peticiones: de gestión, de escritura y de lectura. Las peticiones de gestión las pueden realizan los administradores sobre sus propios proyectos y permiten: crear y eliminar proyectos, cambiar la visibilidad o gestionar los permisos a los usuarios. Para este fin, las peticiones de gestión pueden modificar la información de los proyectos a través del modelo de almacenamiento. Todas las peticiones de gestión tienen como respuesta la información actualizada del proyecto. Las peticiones de escritura se encargan de actualizar el modelo y almacenar el historial de los mensajes, el usuario que mandó el mensaje y las modificaciones que desencadena. Las peticiones de escritura pueden ser:

- **Modificar el modelo**. La Figura 3.2 muestra en detalle los pasos a seguir cuando el bot recibe un mensaje en LN para modificar el modelo. Usando el analizador sintáctico *The Stanford Parser*[21], separa el mensaje en frases, analiza cada frase

[1]https://twitter.com/
[2]https://telegram.org/

y obtiene un árbol sintáctico en el que cada palabra está etiquetada con la función que desempeña dentro de la frase, las dependencias (sujeto, complemento directo, etc), y categoría gramatical. Por ejemplo, nombres (plurales o singulares, comunes o propios...) verbos (tercera persona del singular, pasado, gerundio...), determinantes, etc. SOCIO tiene un conjunto ampliable de reglas con las que busca ciertas estructuras en las frases para traducir los requisitos en elementos del modelo. Las reglas actuales están basadas en el trabajo de Arora y colaboradores [22], aunque las hemos adaptado a las necesidades de nuestro sistema.

Una vez seleccionada la regla, se comprueba el estado del modelo para ver qué elementos son necesarios añadir y cuales existen previamente. Con el estado actual y la regla seleccionada, se llevan a cabo las acciones necesarias para actualizar el modelo. Para que los usuarios puedan analizar posteriormente la procedencia de los elementos del modelo, el mensaje, el usuario que mandó el mensaje y las modificaciones realizadas sobre él, se guardan en el modelo de almacenamiento. Por último, usando PlantUML[3], SOCIO genera una imagen, con el estado actual del modelo, resaltando en otro color los cambios, y la envía como respuesta del mensaje, en la red social que esté usando el usuario.

Las secciones 3.1.1, 3.1.2 y 3.1.3 explican en mas detalle las partes más importantes de este tipo de peticiones.

- **Deshacer** las acciones generadas por los mensajes. Para ello, primero obtiene del historial el último mensaje de modificación recibido y deshace todas las acciones. Finalmente, genera una imagen con el estado actual del modelo y la envía como respuesta.

- **Rehacer** las acciones deshechas. Para ello, busca el último mensaje deshecho y vuelve a rehacer las acciones. Una vez se lleva acabo esta modificación, genera una imagen, con el estado actual del modelo y la envía como respuesta.

Por último las peticiones de lectura permiten obtener información sobre el modelo o el historial del proyecto:

- **Mostrar el estado actual del modelo**: lee el modelo, genera una imagen con su estado actual y la envía como respuesta.

- **Validar el modelo**: lee el modelo y lo valida, es decir, revisa el modelo buscando errores como, herencias cíclicas o atributos o referencias sin tipo. Envía como respuesta los errores, en caso de que los tuviera, o un mensaje de éxito.

- **Obtener el modelo**: envía el fichero donde se almacena el modelo.

- **Obtener el historial de mensajes** de un proyecto, completo o filtrado: por fecha, por elemento del modelo, por usuario, por tipo de acción.

[3] http://plantuml.com/

- **Obtener estadísticas** del proyecto. El bot ofrece estadísticas sobre número de mensajes enviado o acciones realizadas por los usuarios. Ambas se ofrecen bien distribuidas en el tiempo o de forma absoluta; así como de todos los usuarios en general o de uno en concreto. También dispone del porcentaje de autoría de cada usuario y el número de acciones de cada tipo, realizadas por todos los usuarios o por uno en concreto. Para ello primero obtiene la información del historial y usando la librería JFreeChart[4] genera gráficas que serán las que se envíen como respuesta.

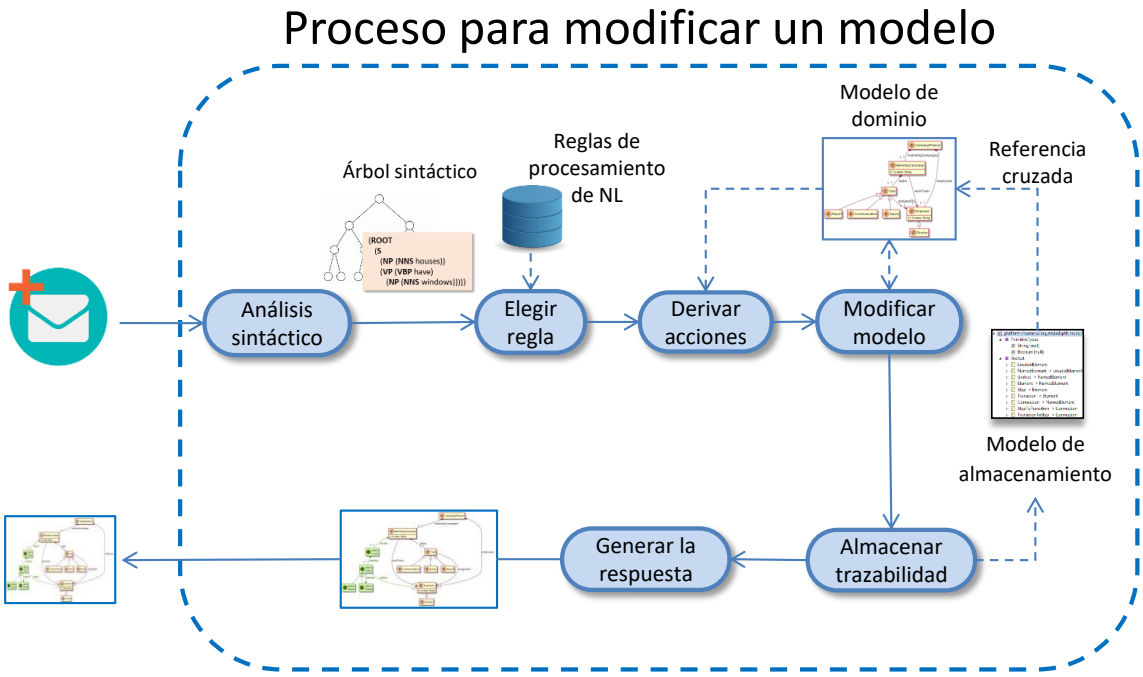


Figura 3.2: Procesamiento de LN

3.1.1. Análisis sintáctico y selección de la regla

Para realizar el análisis sintáctico se usa el analizador de *Stanford*. Este análisis se debe realizar en cada una de las frases que contenga el mensaje. Por ello, se aplica un pre-procesamiento, que separa los mensajes en frases independientes. *The Stanford Parser* ya ofrece un separador de frases. No obstante, para facilitar el trabajo a la hora de identificar que tipo de acciones se pueden realizar, SOCIO separa también las frases subordinadas en frases independientes. Por ejemplo, la frase "*The order, that is associated with a client, has several products*" contiene la frase subordinada: "*that is associated with a client*". Por tanto, la frase inicial se separará en dos frases "*The order is associated with a client*" y "*The order has several products*".

Una vez realizado este pre-procesamiento, se realiza el análisis sintáctico para cada frase, con el que se obtiene un árbol sintáctico. La Figura 3.3 muestra el árbol sintáctico

[4]http://www.jfree.org/jfreechart/

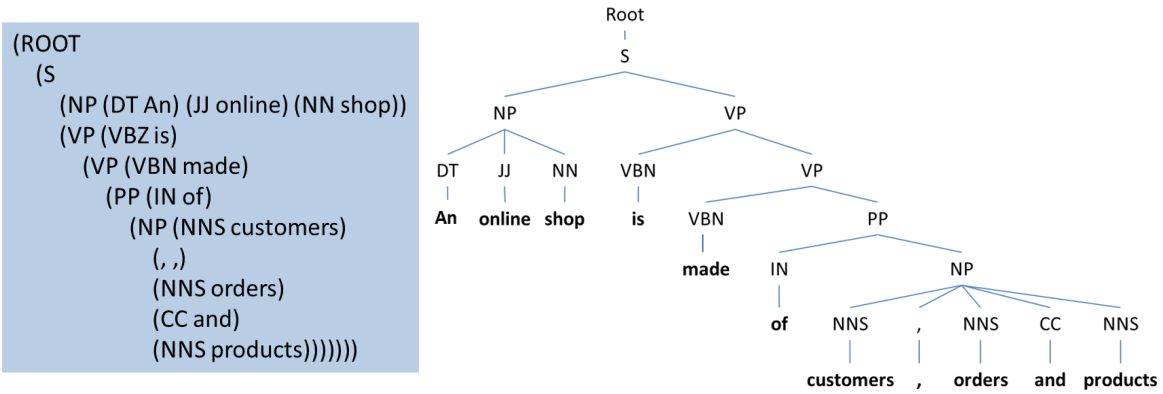


Figura 3.3: Árbol de sintaxis abstracta de la frase "*An online shop is made of customers, orders and products*"

de la frase "*An online shop is made of customers, orders and products*". Todos los árboles tienen como raíz un nodo llamado ***root***, el cual sólo tiene un hijo, el nodo **S**, que representa la sentencia completa. En este caso la sentencia tiene dos hijos. El primero se denomina frase nominal (**NP**) y es una porción de la frase en la que la palabra dominante es un sustantivo. El segundo es una frase verbal (**VP**), un fragmento en el que la palabra dominantes es un verbo. En ambos casos, van acompañados de sus modificadores. En el lado derecho (contenido en la frase verbal) vemos además una frase preposicional (**PP**), subconjunto de la frase que comienzan por una preposición, en este caso es la preposición que acompaña al verbo. Normalmente los complementos de los PPs son NPs. Por último, tenemos las categorías gramaticales de las palabras, y las palabras de la frase como hojas del árbol. Las categorías gramaticales más importantes que podemos encontrar son:

- Sustantivos: **NN** se usa para etiquetar nombres comunes en singular o incontables, **NNS** se para usa etiquetar nombres comunes en plural, **NNP** para nombres propios en singular y **NNPS** para nombres propios en plural.

- Verbos: **VB** etiqueta verbos en infinitivo, **VBP** etiqueta los verbos en presente pero no en tercera persona del singular, **VBZ** etiqueta los verbos en tercera persona del singular del presente, **VBN** los verbos en participio pasado, **VBD** los verbos en pasado, **VBG** los verbos en gerundio y **MD** etiqueta los verbos modales (*will*,*can*, *could*, *may*, *shall*, *would*, *must*...).

- Adjetivos: **JJ** etiqueta los adjetivos, **JJS** etiqueta los adjetivos superlativos y **JJR** los adjetivos comparativos.

- Determinantes: **DT** se usa para etiquetar los determinantes.

- Conjunciones coordinantes como *and* y *or* se representan con **CC**.

- Preposiciones y conjunciones subordinantes se etiquetan con **IN**.

det	shop-3	An-1
amod	shop-3	online-2
nsubjpass	made-5	shop-3
auxpass	made-5	is-4
root	ROOT-0	made-5
case	customers-7	of-6
nmod:of	made-5	customers-7
nmod:of	made-5	orders-9
conj:or	customers-7	orders-9
cc	customers-7	or-10
nmod:of	made-5	products-11
conj:or	customers-7	products-11

Tabla 3.1: Dependencias de la frase "*An online shop is made of customers, orders and products*"

La lista completas de etiquetas para las categorías gramaticales se puede encontrar en `https://catalog.ldc.upenn.edu/docs/LDC99T42/tagguid1.pdf`.

Además del árbol de sintaxis, el analizador de Stanford, devuelve una lista de dependencias. Una dependencia está formada por dos palabras y una etiqueta. Cada palabra va acompañada de su posición en la frase. En las dependencias es donde se identifican los sujetos de las frases, los complementos directos e indirectos o el verbo principal.

La tabla 3.1 muestra las dependencias que se obtienen de la frase de la Figura 3.3. La dependencia **root** indica el verbo principal de la frase, en este caso **made**, que tiene un verbo auxiliar pasivo (*is*) identificado con la etiqueta **auxpass**. Al ser un verbo pasivo hay un sujeto paciente, **nsubjpass** (*shop*), que tiene un adjetivo modificador, *amod* (*online*) y un determinante, **det** (*A*). El verbo además tiene una preposición identificada con **case** y tres sustantivos modificadores que acompañan a la preposición, **nmod:of** (*customers*, *orders* y *products*). Se puede encontrar información más detallada de las dependencias del analizador de Stanford en `https://nlp.stanford.edu/software/stanford-dependencies.html`.

Basándose en el trabajo de Arora y colaboradores [22], SOCIO procesa los resultados del analizador para crear el modelo. Como norma general, los NPs son susceptibles de ser elementos del modelo, ya sean clases, atributos o referencias. En función del verbo principal y la estructura de la frase, se busca en un conjunto de reglas en la que encaja la frase. Actualmente, el bot dispone de las siguientes reglas:

- **Verbo *to be***: Las frases que contengan este verbo pueden indicar una relación de herencia entre dos clases (ejemplo, "*Payment method can be cash, paypal or credit card*"), el tipo de una propiedad (ejemplo, "*price is double*") o una clase abstracta (ejemplo, "*Payment method is abtract*").

- **Verbo *to have***: Las frases con el verbo *to have* o sinónimo, identifican propiedades

de una clase (atributos o referencias), por ejemplo, "*An order has a price*". Para esto, también se tiene en cuenta el genitivo sajón.

- **Verbo *to contain***: Las frases con este verbo o algún sinónimo señalan la relación de composición entre dos clases. Por ejemplo,"*The order contains a payment method*" .
- **Verbos transitivos**: Cuando se encuentra una frase con un verbo, sujeto y objeto directo, o verbo con preposición, sujeto y objeto indirecto, el verbo indica una referencia entre dos clases que corresponden al sujeto y al objeto directo o indirecto. Por ejemplo, "*The order is associated with a client*".
- ***Add***: Las frases imperativas que llevan el verbo *add*, o algún sinónimo, crean una clase o propiedad. Por ejemplo, "*Add a numeric code in Product*" o "*create costumer*".
- ***Remove***: Este regla es similar a la anterior, pero eliminando en lugar de añadiendo Por ejemplo, "*Delete code from product*" o "*Remove costumer*".

Una frase puede encajar en más de una regla. Por ejemplo, "*The order contains a payment method*", encaja en la regla del verbo transitivo ("*The order*" es el sujeto, "*contains*" es el verbo y "*a payment method*" el complemento directo) y en la del verbo *to contain*. Por ello, las reglas tienen un orden de prioridad – que es su posición en la lista anterior –, y sólo la de mayor prioridad será seleccionada.

3.1.2. Modificación del modelo

Una vez seleccionada la regla, y teniendo en cuenta el estado del modelo, se derivan las acciones necesarias para actualizarlo. Hay tres tipos de acciones que se pueden realizar: añadir, eliminar o modificar clases, atributos y referencias. Todos los elementos del modelo se representan por un nombre, que además el bot, de forma automática, dará formato *camel case*. Como norma de estilo, las clases estarán en singular y empezarán en mayúsculas y mientras que los atributos y referencias empezarán por minúscula. En los atributos y las referencias es necesario indicar a qué clase pertenecen a la hora de crearlos, pero el tipo se puede indicar posteriormente. La diferencia entre los atributos y las referencias la da el tipo, todos las características de una clase cuyo tipo sea primitivo son atributos, y los demás son referencias en las que el tipo es otra clase del modelo. De esta manera, cuando una característica no tiene tipo se define como atributo sin tipo, y cuando se decida el tipo, o bien se convierte en una referencia o bien solo es necesario añadir el tipo. Las modificaciones que se pueden hacer sobre una clase son: poner o quitar un supertipo y hacerla abstracta o concreta. En los atributos y referencias las modificaciones que se pueden hacer son: dar o cambiar el tipo, cambiar la cardinalidad, y cambiar de atributo a referencia y viceversa.

Una vez que las reglas (explicadas en la Sección 3.1.1) detectan los elementos de interés, se ha de comprobar si estos existen en el modelo. Para evitar la redundancia y ser más flexible, SOCIO usa WordNet [23], un diccionario de palabras en inglés, para

detectar sinónimos. De esta manera, es posible referirse a los elementos que ya existen en el modelo por medio de sinónimos. Por ejemplo, podemos usar *client* para referirnos a una clase existente *customer*. También puede influir el estado del modelo en el significado de la frase. Por ejemplo, la frase "*The Father is a person*", en principio se identificará como que una clase*Father* hereda de otra *Person*, pero si existiera previamente una característica que se llame *father*, en cualquier clase, se llevarán a cabo las acciones necesarias para que *father* sea una referencia a la clase *Person*.

Una vez se han derivado todas las acciones se procede a ejecutarlas en orden. Cada modelo está guardado en un fichero independiente en formato ecore, que es el formato que usa el framework de Eclipse para modelado (Eclipse Modelling Framework, EMF) [24].

3.1.3. Trazabilidad y generación de la respuesta

SOCIO tiene un modelo para almacenar los proyectos, los usuarios y el historial de cada proyecto. La Figura 3.4 muestra este modelo. SOCIO contiene un listado de los proyectos (*projects*) y los usuarios (*users*), y estos (*Project* y *User*) entre ellos tienen una relación bidireccional de propiedad (*owner*). Los usuarios tienen un canal (*channel*), que corresponde a la red social con la que se comunican con SOCIO, un nombre (*name*), un alias (*nick*) y un identificador (*id*). Estos dos últimos deben ser únicos en cada red social. Los usuarios también tienen una lista de contribuciones (*contributions*) en proyectos. Las contribuciones (*Contribution*) además del proyecto al que referencian, tienen un nivel de acceso (*access*: *edit* o *read*). Como de momento SOCIO solo da soporte la creación de modelos de domino, estos se pueden almacenar en formato ecore. Los proyectos tienen una referencia al fichero donde se almacena el modelo en este formato, (*model*). Además, también tienen un nombre (*name*) y un historial (*history*).

El historial de un proyecto contiene la fecha de creación (*createAt*) y un listado con todos los mensajes *messages* que modifican el modelo. Los mensajes (*Message*) tienen una referencia del usuario *user* al que pertenecen, una fecha (*date*), un texto *text* y una lista de frases (*sentences*). Las frases (*Sentence*) contienen una lista de las acciones (*actions*) que provocan. Estas acciones (*Action*) pueden ser de los tres tipos mencionados en la Sección 3.1.2: añadir (*Add*), eliminar (*Delete*) o modificar (*Update*) un objeto. Todas las acciones tienen una referencia al objeto (*element*) afectado del modelo. Las acciones que modifican los elementos, además de esta referencia, también tienen una copia del objeto antes (*oldElement*) y después (*newElement*) de modificarlo, para que queden reflejado los cambios.

Tras actualizar el historial, se genera una imagen con extension png, con el estado actual del modelo, donde los elementos creados o modificados se muestran en verde, y los elementos eliminados en rojo. La imagen facilita la visualización del modelo, y de los cambios realizados y es un formato soportado por la mayoría de redes sociales. Para crear está imagen, se usa PlantUML.

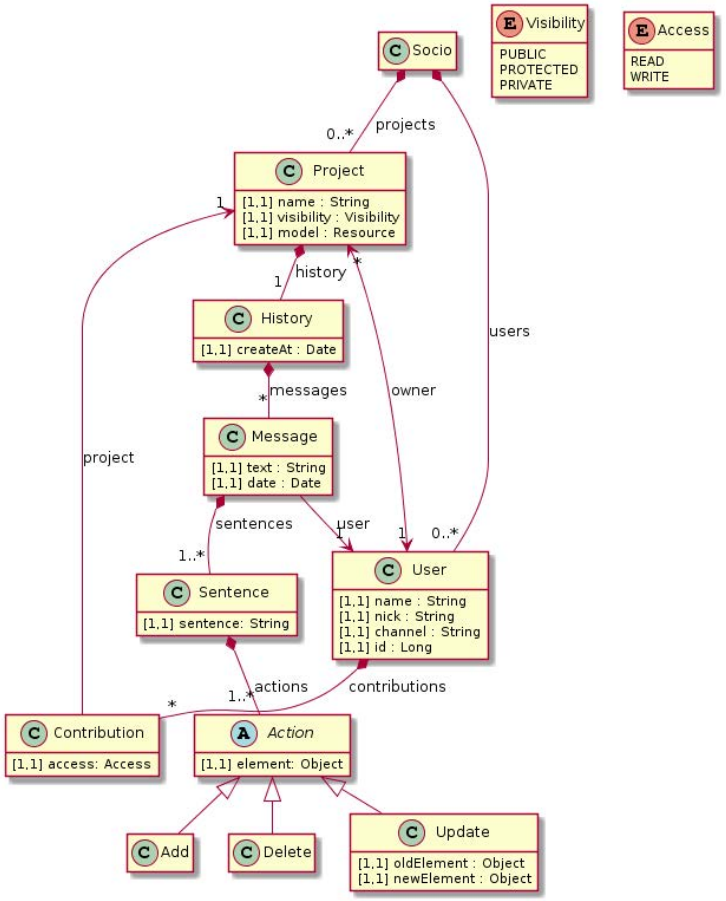


Figura 3.4: Meta-modelo de almacenamiento de SOCIO

3.2. Ejemplo de creación de un modelo

La Figura 3.5 muestra varios mensajes expresados en LN usados para crear un modelo conceptual para una tienda online. A continuación explicamos cuál es el resultado de procesar cada mensaje:

1. El mensaje "*An online shop is made of customers, orders and products. The products are identified by a numeric code*" contiene dos frases, que SOCIO separa y analiza de forma independiente:

   - "*An online shop is made of customers, orders and products*". Esta frase selecciona la regla *to contain*, los NPs *online shop*, *customers*, *order* y *products* se añaden como clases del modelo, y se crean relaciones de contención en *OnlineShop* al resto de clases, y debido a que están en plural, la cardinalidad es de 1 a muchos.
   - "*The products are identified by a numeric code*". Se selecciona la regla del verbo *to have*. *Products* es una clase que ya existe por lo que no es necesario añadirla, y se añade el atributo *code* de tipo numérico (*int*).

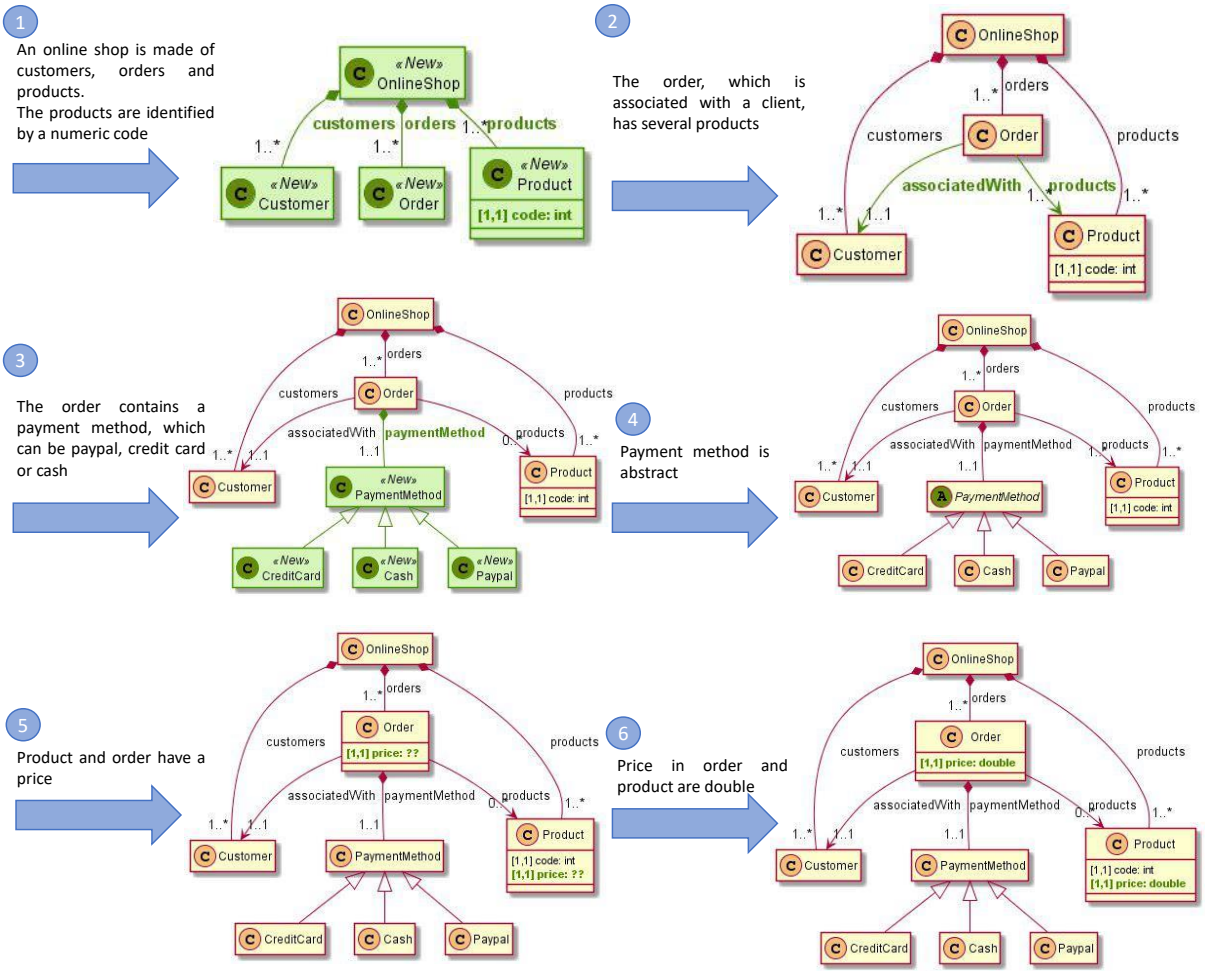


Figura 3.5: Ejemplo de creación de un modelo conceptual para una tienda online

2. "*The order, which is associated with a client, has several products*" contiene una subordinada, por lo que también se divide en dos frases independientes:

   - "*The order is associated with a client*" lanza la regla de los verbos transitivos. La clase *Order* ya existe, y *client* es sinónimo de la clase que ya existe *Customer*, por lo que no es necesario añadir ninguna de las dos clases. Se crea una referencia desde la clase *Order* a la clase *Customer* llamada *associateWith*. Como *client* está en singular, la cardinalidad de la referencia es uno.
   - "*The order has several products*" selecciona la regla del verbo *to have* y crea una referencia de uno a muchos en *Order* a *Product*.

3. "*The order contains a payment method, which can be paypal, credit card or cash*" también tiene una subordinada, y se vuelve a dividir en dos frases independientes:

   - "*The order contains a payment method*" lanzar la regla del verbo *to contain*, creando una clase llamada *PaymentMethod* y una relación de contención entre *Order* y *PaymentMethod*. Como *PaymentMethod* es singular, la cardinalidad de la relación es uno.

- "*Payment method can be paypal, credit card or cash*" lanza la regla del verbo *to be*, creando las clases *Paypal*, *CreditCard* y *Cash* y una relación de herencia entre estas y la clase *PaymentMethod*.

4. "*Payment method is abstract*" sólo tiene una frase y no tiene ninguna subordinada. Lanza la regla del verbo *to be*, y convierte la clase *PaymentMethod* en abstracta.

5. "*Product and order have a price*" es sólo una frase y no tiene subordinadas. Lanza la regla *to have* y crea una característica *price* en las clases *Product* y *Order*. Como no se puede obtener de la información que da la frase, *price* se queda sin tipo. Aunque SOCIO permite que esto suceda, para facilitar la creación del modelo, esto es un error que se reportará si se valida el modelo.

6. "*Price in order and product are double*" es sólo una frase sin subordinadas. Lanza la regla del verbo *to have* y a las características *price* de las clases *Product* y *Order* les da el tipo *double*. Como *double* es un tipo primitivo, la característica se queda como atributo.

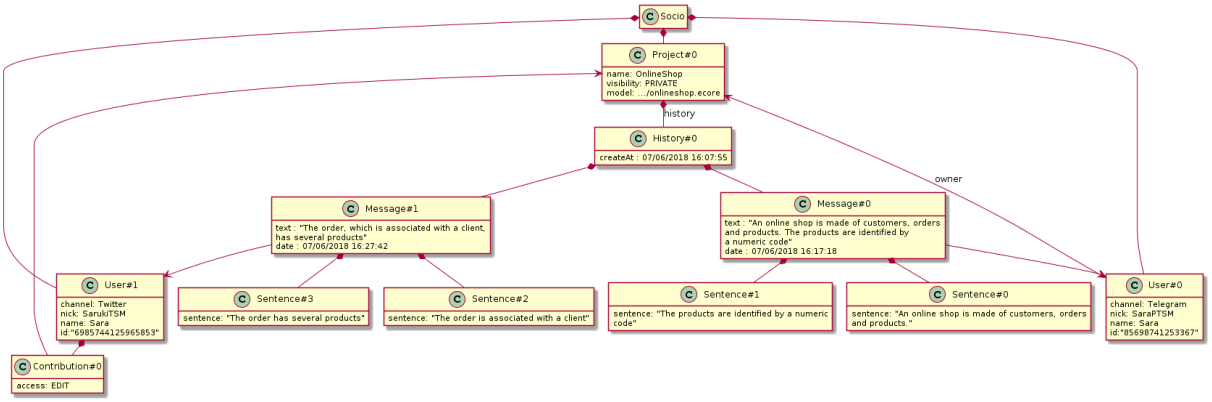


Figura 3.6: Parte del modelo de instanciación del modelo de almacenamiento

La Figura 3.6 muestra parte de la instanciación del modelo mostrado en la Figura 3.4 tras crear y mandar las dos primeras frases. En esta instancia, hay dos usuarios: el usuario *User#0* se comunica con SOCIO desde Telegram, su alias es SaraPTSM y su nombre es Sara. El usuario *User#1* se comunica con SOCIO desde Twitter, su alias es SarukiTSM y su nombre es Sara.

También encontramos una instancia de un proyecto, *Project#0* que se llama *OnlineShop*. Este proyecto pertenece al usuario *User#0*. *User#1* tiene una contribución (*Contribution#0*) con acceso de edición al proyecto *Project#0*. En el historial del proyecto, podemos encontrar dos mensajes: el primero, *Messsage#0*, enviado por *User#0* y el segundo, *Message#1* enviado por *User#1*. En el texto del mensaje *Message#0* encontramos: "*An online shop is made of customers, orders and products. The products are identified by a numeric code*", el primer mensaje mostrado en la Figura 3.5. Contiene dos frases: *Sentence#0*, "*An online shop is made of customers, orders and products*", y *Sentence#1*, "*The products are identified by a numeric code*". El mensaje *Message#1* tiene

el texto "*The order, which is associated with a client, has several products*", y las frases "*The order is associated with a client*" y "*The order has several products*".

Para no dificultar la legibilidad del modelo, se ha reducido el tamaño y no se han incluido las acciones que produce cada frase, sino que se muestran en la Tabla 3.2. La frase *Sentence#0*, produce acciones para crear las clases *OnlineShop*, *Customer*, *Order* y *Product*, luego crea las referencias *customers*, *orders* y *products* en la clase *OnlineShop* y por último modifica estas referencias para darles el tipo que les corresponde: a la referencia *customers* el tipo *Customers*, a la referencia *orders* el tipo *Order* y a *products* el tipo *Product*. La frase *Sentence#2* produce dos acciones: crear el atributo *code* en la clase *Product* y modificar el tipo de *code* para que sea *int*. *Sentence#3* produce dos acciones: crear la referencia *products* en la clase *Order* y cambiar el tipo de la referencia a *Product*.

Sentence#0: "An online shop is made of customers, orders and products"	
Add#0:	element: class onlineShop#OnlineShop
Add#1:	element: class onlineShop#Customer
Add#2:	element: class onlineShop#Order
Add#3:	element: class onlineShop#Product
Add#4:	element: reference onlineShop#OnlineShop.customers
Add#5:	element: reference onlineShop#OnlineShop.orders
Add#6:	element: reference onlineShop#OnlineShop.products
Update#0	element: reference onlineShop#OnlineShop.customers oldElement: reference customer in OnlineShop, [1,*], type = null newElement: reference customers in OnlineShop, [1,*], type = Customer
Update#1	element: reference onlineShop#OnlineShop.orders oldElement: reference orders in OnlineShop, [1,*], type = null newElement: reference orders in OnlineShop, [1,*], type = Order
Update#2	element: reference onlineShop#OnlineShop.products oldElement: reference products in OnlineShop, [1,*], type = null newElement: reference products in OnlineShop, [1,*], type = Product
Sentence#1: "The products are identified by a numeric code"	
Add#7	element: atributte onlineShop#Product.code
Update#3	element: atributte onlineShop#Product.code oldElement: atributte code in Product, [1,1], type = null newElement: atributte code in Product, [1,1], type = int
Sentence#2: "The order is associated with a client"	
Add#8	element: reference onlineShop#Order.associateWith
Update#4	element: reference onlineShop#Order.associateWith oldElement: reference associateWith in Order, [1,1], type = null newElement: reference associateWith in Order, [1,1], type = Customer
Sentence#3: "The order has several products"	
Add#9	element: reference onlineShop#Order.products
Update#5	element: reference onlineShop#Order.products oldElement: reference products in Order, [1,*], type = null newElement: reference products in Order, [1,*], type = Product

Tabla 3.2: Tabla de las acciones que producen cada frase

4

Arquitectura y herramienta

En este capítulo se explicará la arquitectura de SOCIO (Sección 4.1), y cómo usar la herramienta en Telegram y Twitter (Sección 4.2). Es posible modificar un modelo desde dos redes sociales a la vez, lo que se explicará en la Sección 4.2.3).

4.1. Arquitectura

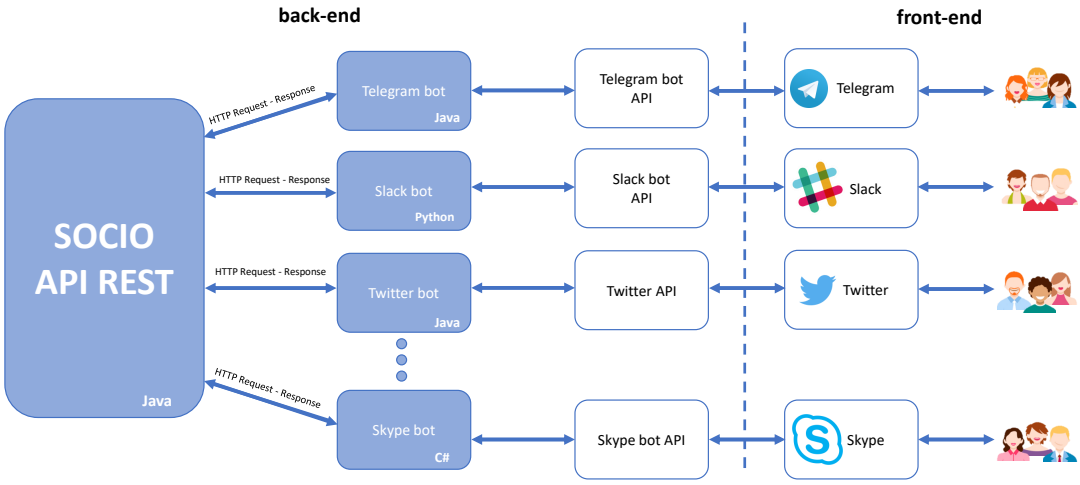


Figura 4.1: Arquitectura del bot

Se ha diseñado una arquitectura extensible, basada en servicios web, como base para SOCIO. Una de las principales ventajas de esta arquitectura, es que es extensible a otras redes sociales de manera externa: no es necesario el código fuente de SOCIO para

extenderla. La arquitectura también es multi-plataforma: distintas redes sociales pueden requerir soporte en diferentes lenguajes de programación, y la arquitectura basada en servicios de SOCIO lo soporta. Por último, no es necesario parar el servicio principal para añadir otra red social.

La Figura 4.1 muestra esta arquitectura. Por un lado, está el SOCIO, que es un servicio Rest, implementado en Java, usando la API Jersey[1]. Los bots de las redes sociales se comunican con SOCIO a través de peticiones HTTP. Estos pueden estar implementados en cualquier lenguaje.

En este apartado se va explicar la arquitectura interna de SOCIO, dejando de lado los detalles del servicio Rest. La Figura 4.2 presenta las partes más importantes del diagrama de clases de SOCIO. Como se puede observar, es muy similar al modelo de almacenamiento de la aplicación (ver Figura 3.4). Los dos modelos siguen la misma estructura para automatizar la integración entre el código fuente y el almacenamiento.

Para explicar el diagrama de clases se va a ampliar lo ya dicho en la Sección 3.1.3. Las clases añadidas en este diagrama son: *Parser* y *Rule*. La clase *Project* contiene un *Parser* que se encarga de separar las frases y realizar el análisis sintáctico del texto. También contiene una lista de todas las reglas. Para simplificar el modelo no se han añadido las clases que heredan de *Rule*, pero existe una para cada una de las reglas explicadas en la Sección 3.1.1. Lo mismo sucede con las acciones, hay una clase para cada una de las acciones explicadas en la Sección 3.1.2.

La clase *Socio* controla toda la aplicación, y tiene por tanto los métodos para satisfacer las peticiones explicadas en la Sección 3.1. En la Figura, los métodos de esta clase se han separado en tres compartimentos. En el primer compartimento se encuentran los métodos para las peticiones de gestión: *createProject* crea un proyecto nuevo, *deleteProject* elimina un proyecto y *addContribution*, *deleteContribution* y *changeContribution* gestionan los permisos de los usuarios en los proyectos.

El segundo compartimento contiene los métodos para responder las peticiones de edición: *do* modifica el modelo interpretando mensajes en lenguaje natural, *undo* deshace el último mensaje que modifica el modelo y *redo* rehace el último mensaje deshecho. En el último compartimento se encuentran los métodos para responder a las peticiones de lectura: *getHistory* obtiene los mensajes del historial del proyecto, *getStatistics* calcula y genera las gráficas de las estadísticas del proyecto, *validate* valida el proyecto y *get* obtiene el fichero con el modelo.

Los métodos para modificar y leer el modelo los gestiona la clase *Project* con métodos con el mismo nombre. El método *do* es el que se encarga de realizar el procesamiento del lenguaje natural y las acciones necesarias para actualizar el modelo. Para ello usa los métodos de la clase *Parser*. El primero, *splitSentences*, separa el texto en frases independientes. El segundo, *parseSentence*, realiza el análisis sintáctico, comprueba que reglas (clase *Rule*) encajan con la frase (método *isValid* de la clase *Rule*) y devuelve la que tiene mayor prioridad (*getPriority* de la clase *Rule*). Una vez tiene la regla seleccionada, el método *do*, le pasa el modelo y le pide que cree las acciones necesarias para modificar

[1] https://jersey.github.io/

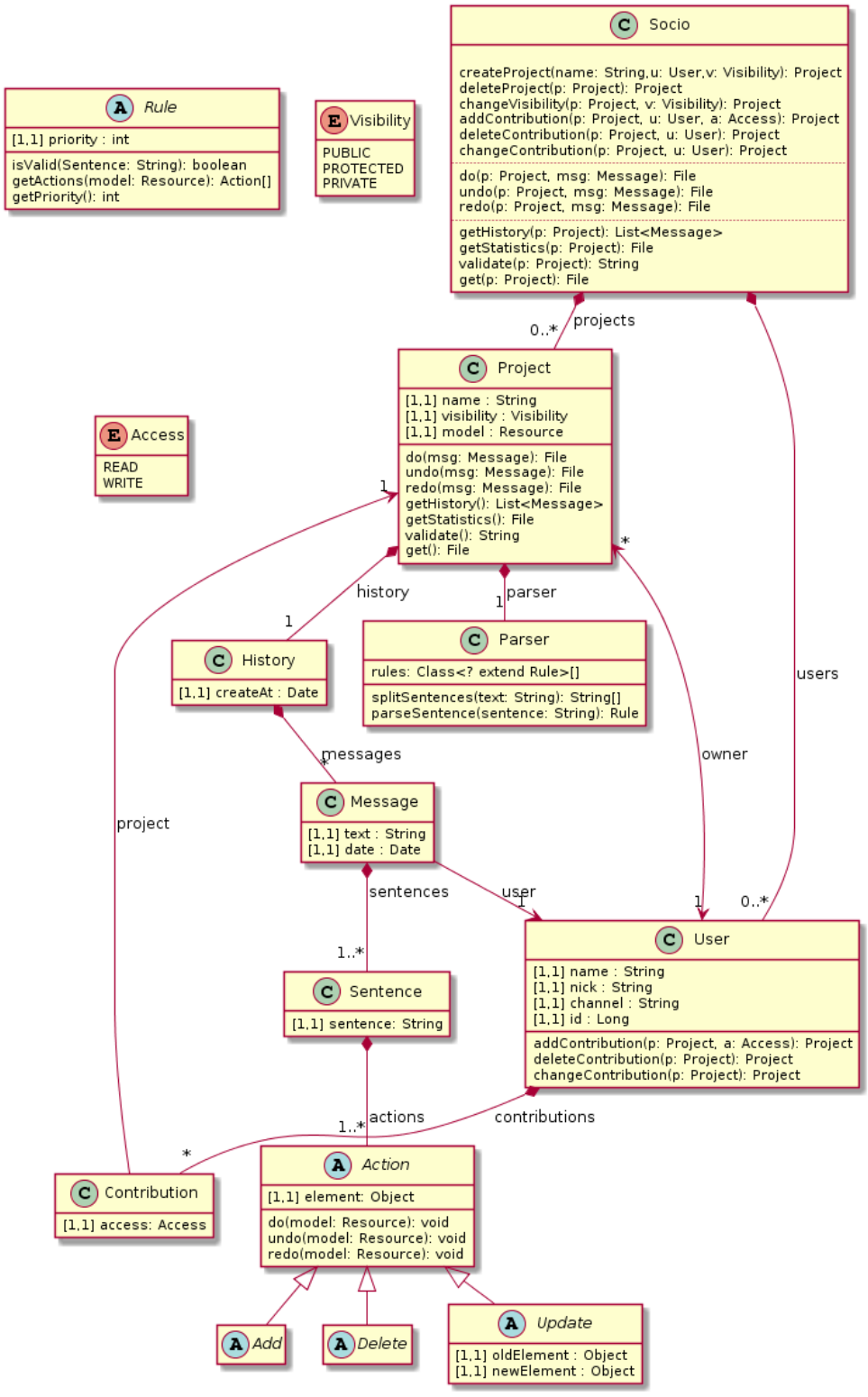


Figura 4.2: Parte del diagrama de clases de SOCIO

el modelo (método *getActions* de la clase *Rule*). Las acciones se ejecutan con el método *do* de la clase *Action*. Por último, se guarda el mensaje y las acciones en el historial.

4.2. Herramienta

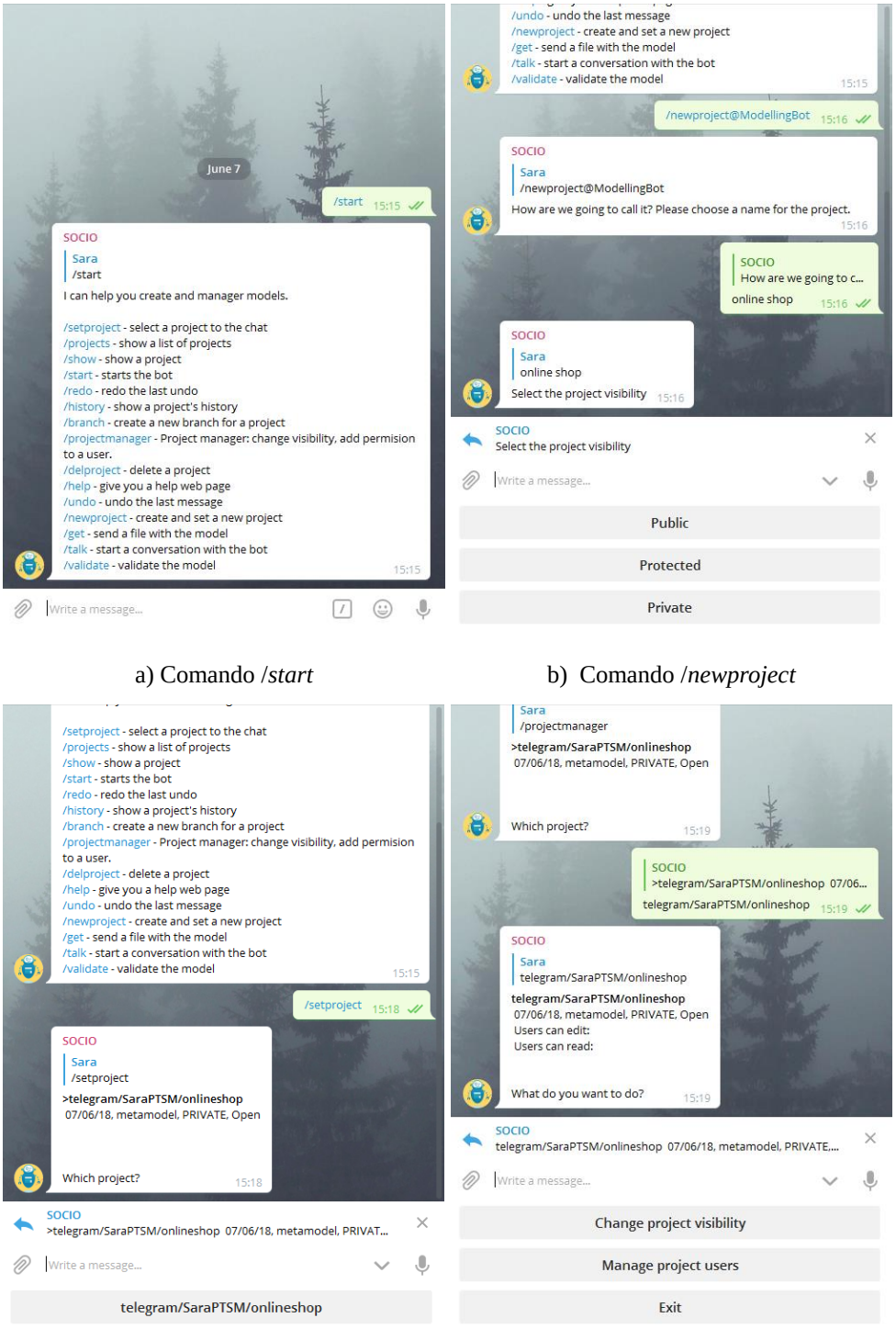
En esta Sección se explicará en front-end de la Figura 4.1, es decir, cómo los usuarios interactuan con SOCIO a través de las redes sociales. Esta interacción es distinta dependiendo de la red social. A continuación se explicará la herramienta en las redes sociales en las que actualmente está disponible: Telegram y Twitter.

4.2.1. Telegram

Como ya se ha mencionado, Telegram es un sistema de mensajería instantánea, similar a Whatsapp, donde los bots son frecuentes. La interacción entre los usuarios puede ser a través de chats privados con una sola persona o bot, o en grupos con hasta 100.000 miembros (que pueden ser personas o bots indistintamente).

A la hora de interactuar con los bots, hay leves diferencias a como se haría con las personas. Mientras que en un grupo, cada mensaje que se envíe llegará a todas las personas, los bots, por defecto, no tienen acceso a todos los mensajes. Para poder comunicarte con el bot están los comandos, palabras precedidas por "/". Los mensajes dirigidos al bot dentro de un grupo deben comenzar por un comando y pueden tener argumentos (todo el texto que haya a continuación del comando). Los comandos se pueden completan con "@<alias del bot>" para aclarar a qué bot está referenciando. Para facilitar todo esto, las aplicaciones de Telegram disponen de un sistema de auto completado. SOCIO en Telegram tiene los siguientes comandos:

- `/start`: muestra un mensaje de bienvenida y lista todos los comando disponibles del bot (Figura 4.3.a).
- `/newproject`: crea un nuevo proyecto (Figura 4.3.b).
- `/delproject`: elimina un proyecto.
- `/setproject`: selecciona un proyecto para un chat concreto. De esta manera cada vez que se realice una acción no habrá que especificar el proyecto, sino que siempre será sobre el mismo (Figura 4.3.c).
- `/projects`: lista todos los proyectos que tiene SOCIO.
- `/projectmanager`: permite cambiar la visibilidad o gestionar los permisos a los usuarios (Figura 4.3.d).
- `/talk`: manda un mensaje en LN para modificar el modelo (Figura 4.4 a y b).
- `/undo`: deshace el último mensaje (Figura 4.4.c).
- `/redo`: rehace el último mensaje deshecho (Figura 4.4.d).
- `/show`: muestra una imagen con el estado actual del modelo (Figura 4.5.a).



a) Comando */start*

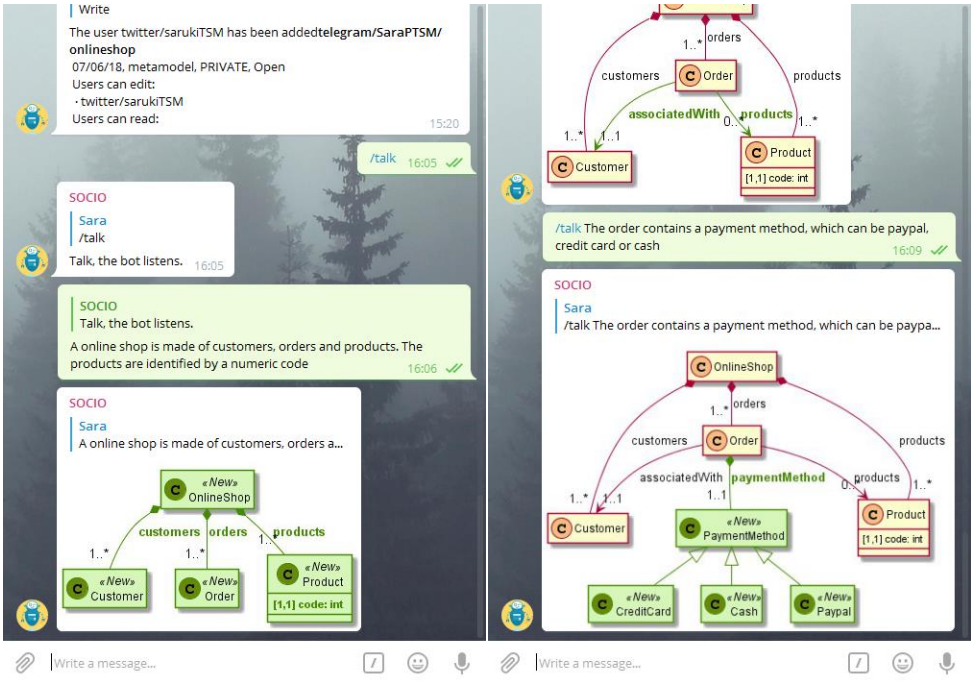
b) Comando */newproject*

c) Comando */setproject*

d) Comando */projectmanager*

Figura 4.3: Ejemplo del uso de SOCIO en Telegram. Comandos *start*, *newproject*, *setproject* y *projectmanager*

- `/get`: obtiene un fichero con el modelo en formato ecore.
- `/validate`: valida el modelo (Figura 4.5 a y b).



a) Comando */talk*

b) Comando */talk* compacto

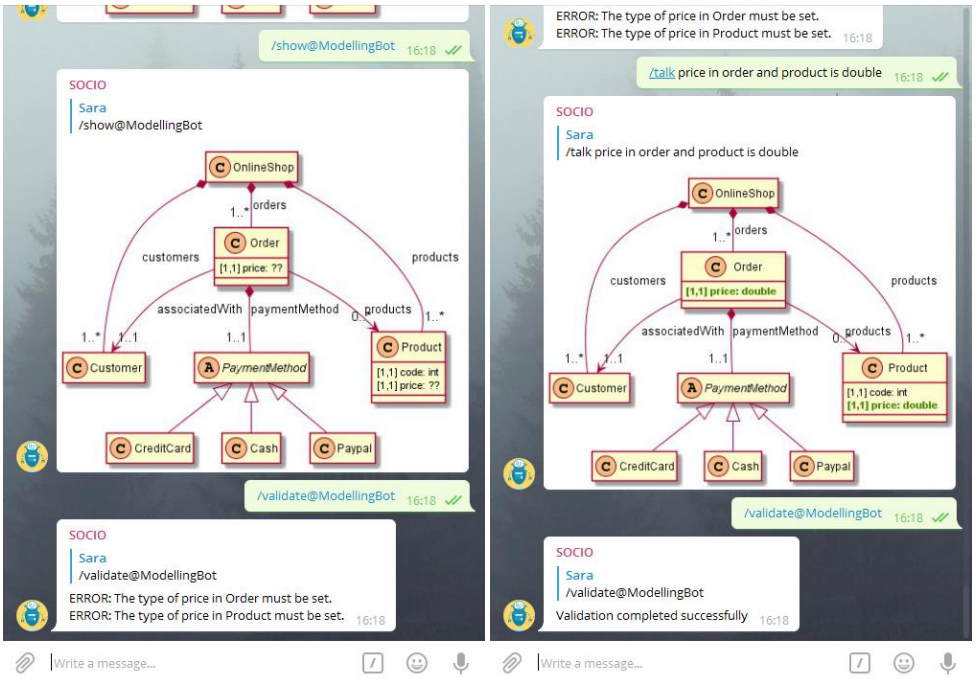
c) Comando */undo*

d) Comando *redo*

Figura 4.4: Ejemplo del uso de SOCIO en Telegram. Comandos *talk*, *undo* y *redo*

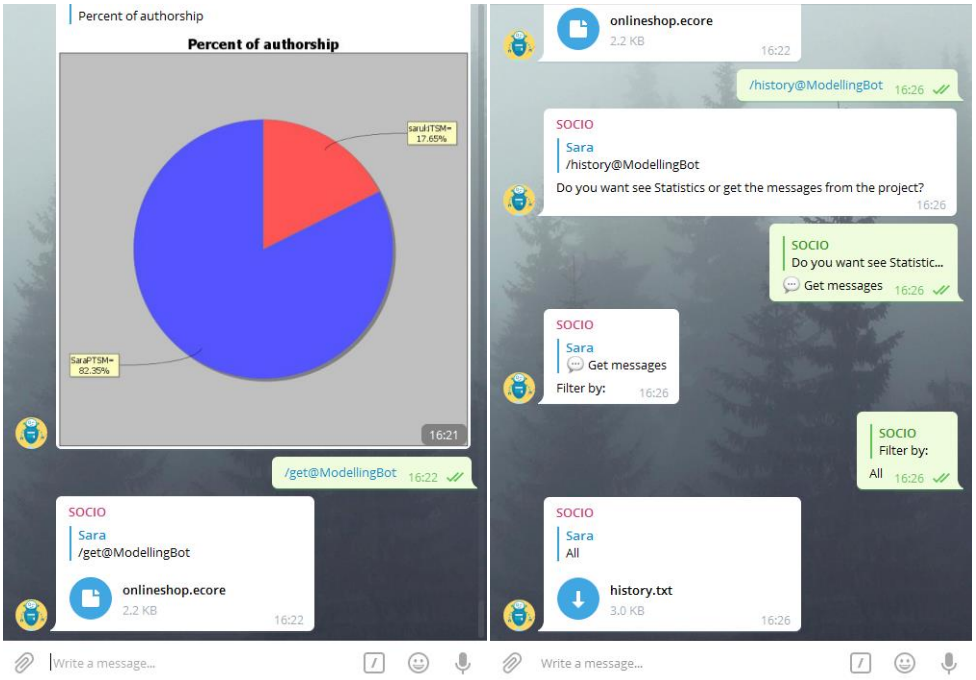
- `/history`: obtiene el histórico de los mensajes, pudiendo indicar el filtro deseado, y las estadísticas del proyecto.

- `/help`: envía un mensaje con un enlace a la página web de ayuda.



a) */show* y */validate* con errores

b) Comando */validate* sin errores



a) Comandos */statistics* y */get*

a) Comandos */statistics* y */get*

Figura 4.5: Ejemplo del uso de SOCIO en Telegram. Comandos *show*, *validate*, *get*, *statistics* y *history*

En el siguiente enlace hay un vídeo con una demostración del uso de SOCIO en Telegram: `https://www.youtube.com/watch?v=kHf2qNtfNW4`

Las figuras 4.3, 4.4 y 4.5 muestran ejemplos del uso de SOCIO dentro de Telegram. Una

de las características de SOCIO en Telegram es que no es necesario indicar en cada mensaje todo los datos necesarios, sino que, el bot podrá mantener una conversación con el usuario, y preguntar los datos que le faltan. Ejemplos de esto se puede ver en varios comandos: *newproject* (Figura 4.3.b), *setproject* (Figura 4.3.c), *projectmanager* (Figura 4.3.d) y *talk* (Figura 4.4.a). En el comando *newproject* (Figura 4.3.b), por ejemplo, el bot primero pregunta el nombre del proyecto y luego el nivel de visibilidad.

Además, Telegram permite facilitar un teclado al usuario, y tan sólo es necesario pulsar uno de los botones. Este tipo de teclado se le proporciona al usuario en varías ocasiones, por ejemplo, cada vez que tiene que elegir un proyecto, se proporciona un teclado donde hay un botón para cada proyecto que se pueda seleccionar (Figura 4.3.c). También es posible pasar los argumentos directamente al comando. Un ejemplo de esto es la Figura 4.4.b, en la que al comando *talk* se le pasa el mensaje en NL como argumento del comando, evitando que el bot pregunte.

4.2.2. Twitter

Twitter es un servicio de micro-blogging, que permite enviar mensajes de texto plano de una longitud máxima de 280 caracteres, a los que se pueden adjuntar imágenes, que se denominan *tweets*. Los tweets son por lo general públicos. A los mensajes se les puede añadir etiquetas denominadas *hashtags*, que son palabras precedidas por "#". Un tweet puede mencionar a cualquier usuario, el cual recibirá una notificación. Como Twitter no tiene un trato especial para los bots, sino que son cuentas normales manejadas por programas en lugar de personas, el concepto de comando no existe. Para indicar las acciones que queremos que realice el bot, los comandos del bot se envían mediante *hashtags*. Los comandos que soporta SOCIO en Twitter son:

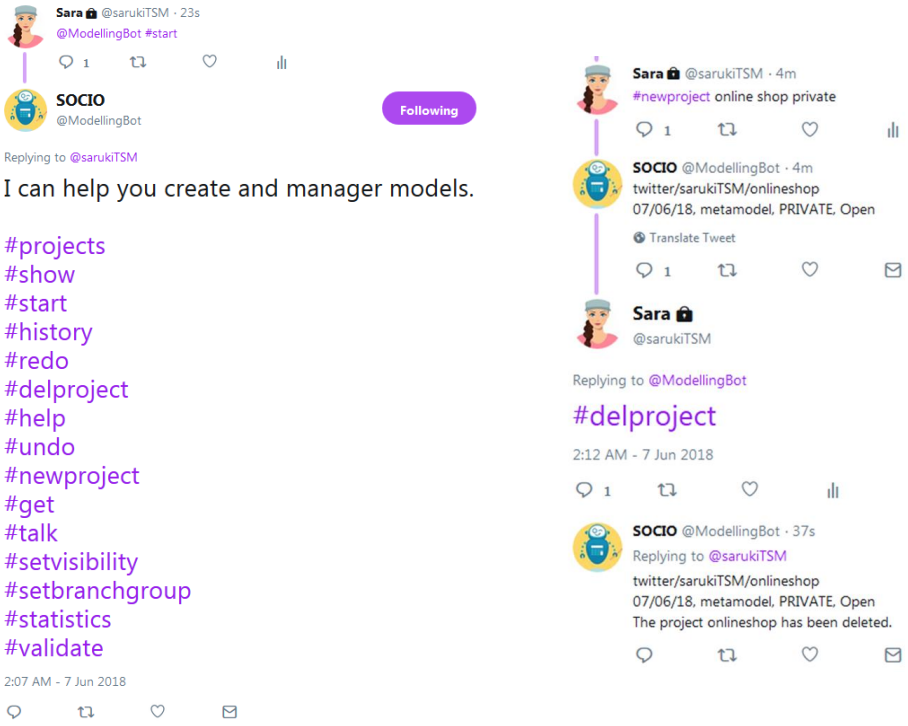
- `#start`: manda un mensaje de bienvenida (Figura 4.6.a).
- `#newproject`: crea un proyecto. Es necesario indicar el nombre del proyecto, y opcionalmente su visibilidad. Si no se señala visibilidad, se toma *public* por defecto (Figura 4.6.b).
- `#delproject`: elimina un proyecto que ha de identificarse mediante un nombre (Figura 4.6.b).
- `#projects`: envía un tweet por cada proyecto disponible, con toda la información del proyecto (Figura 4.7.a).
- `#setvisibility`: cambia la visibilidad de un proyecto. Es necesario identificar el proyecto e indicar la visibilidad.
- `#talk`: se usa para editar el modelo (Figura 4.7.a). Al no existir el concepto de comando para bots en Twitter, se puede omitir `#talk`, porque todos los mensajes que recibe el bot sin comando, los interpreta como si pertenecieran a *talk* (Figura 4.7.b). Para este comando es necesario identificar el proyecto y mandar un mensaje en NL para indicar las modificaciones.

- `#undo`: deshace el último mensaje de un proyecto(Figura 4.7.c), que es necesario identificar.
- `#redo`: rehace el último mensaje deshecho de un proyecto, que es necesario identificar (Figura 4.7.d).
- `#show`: devuelve una imagen con el estado actual de un proyecto concreto 4.8.b).
- `#get`: devuelve un enlace a un servidor para poder descargar un fichero con el modelo en formato ecore. Es necesario identificar el proyecto y se puede añadir el tiempo que se quiere que el fichero esté disponible. Para ello, se debe indicar un número y la unidad. Las posibles unidades son minutos ("m", "min" o "minutes"), horas ("h" o "hours") y días ("d" o "days") (Figura 4.9.a).
- `#validate`: valida un proyecto e indica cuales son los errores en caso de que los hubiera (Figura 4.7 a y b).
- `#history`: devuelve un enlace para poder descargar un fichero con todos el historial de los mensajes de un proyecto. Al igual que con *get*, es posible añadirle el tiempo que se desea que esté el fichero disponible (Figura 4.9.b).
- `#statistics`: muestra todas las gráficas de las estadísticas de un proyecto(Figura 4.9.c).
- `#help`: envía un mensaje de ayuda, con un enlace a la página web de ayuda.

A diferencia de en Telegram, en Twitter no hay un proyecto asociado a los usuarios, por lo que normalmente es necesario identificar en qué proyecto vamos a realizar la acción. Para ello hay dos maneras, la primera es indicándolo de forma explicita con: "/<canal del administrador>/<alias del administrador>/<nombre del proyecto>" (ver Figura 4.8.b, comando *show*). Para facilitar esto a los usuarios, todos los mensajes que SOCIO envía comienzan con un identificador de proyecto. Respondiendo a estos mensajes, directamente se asocia la respuesta con el proyecto en cuestión. Un ejemplo de esto lo podemos ver en la Figura 4.6, tras solicitar la creación de un nuevo proyecto, SOCIO responde con la identificación del proyecto. Después el usuario responde a este mensaje con el comando *delproject* sin necesidad de especificar el proyecto, y el bot asocia el comando con el proyecto anteriormente mencionado. Gracias a esto, se puede mantener una conversación con el bot, respondiendo a sus mensajes, y simplificando la interacción.

4.2.3. Trabajando con las dos redes sociales a la vez

Dos usuarios pueden trabajar en el mismo proyecto, desde redes sociales distintas. Las Figuras 4.4.a, 4.4.b ,4.7.a y 4.7.b, muestran a dos usuarios, uno de Telegram y el otro de Twitter, trabajando simultáneamente en el mismo proyecto. Para que esto sea posible, SOCIO mantiene informados a los usuarios de una red social de los cambios que surgen en la otra, cuando sea necesario. Esta sincronización también se realiza si hay varios grupos de Telegram trabajando sobre el mismo proyecto. La Figura 4.10 muestra cómo SOCIO

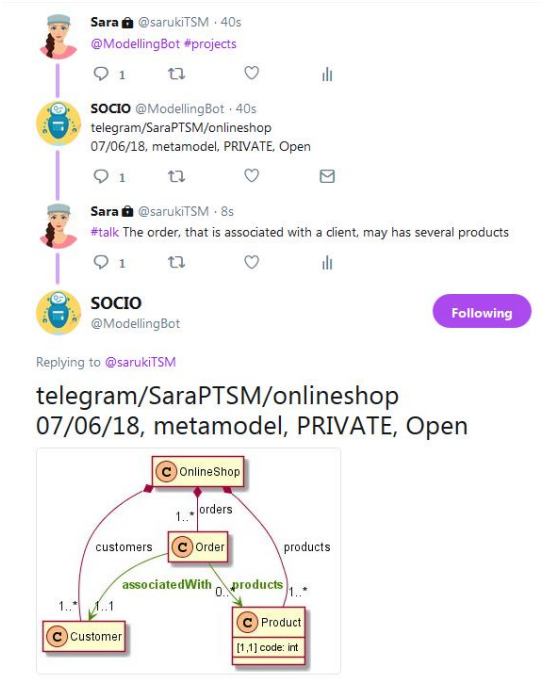


a) Comando *#start*

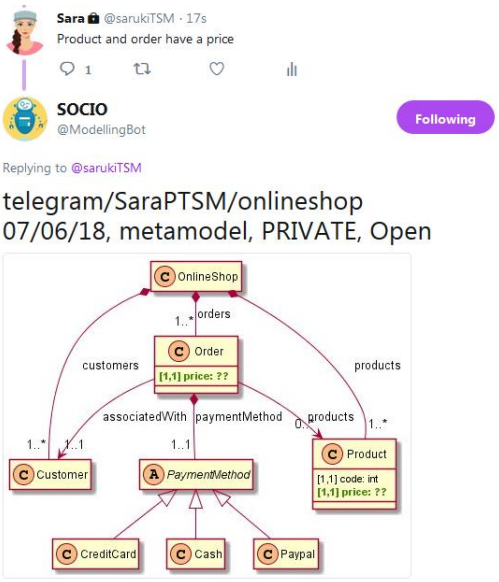
a) Comandos *#newproject* y *#delproject*

Figura 4.6: Ejemplo del uso de SOCIO en Twitter. Comandos *start*, *newproject* y *delproject*

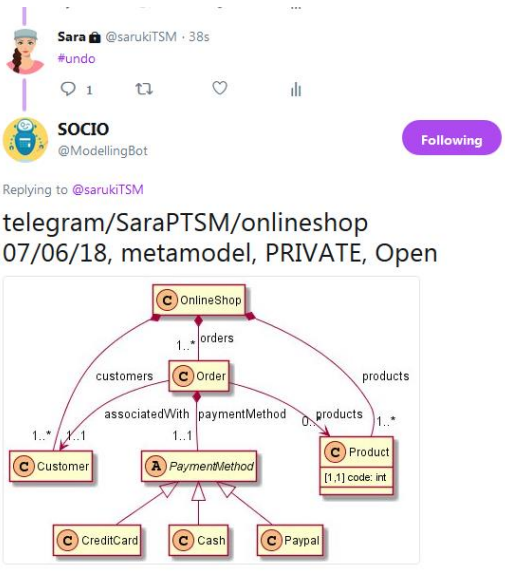
actualiza esta información en Twitter cuando se realiza algún cambio en Telegram. De manera similar, la Figura 4.11 presenta una actualización en Telegram cuando el modelo se modifica en Twitter.



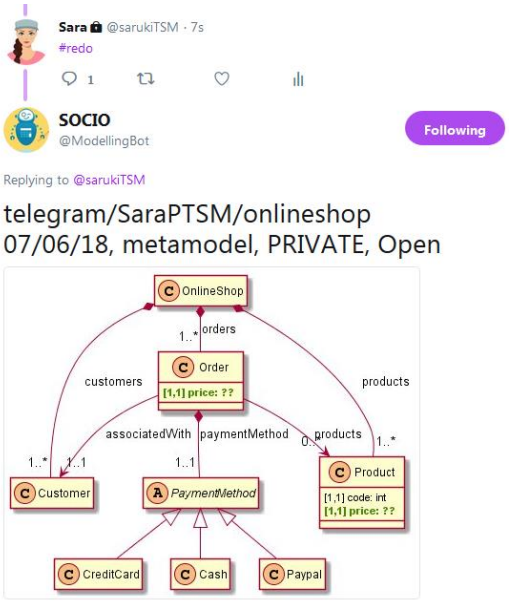
a) Comandos *#projects* y *#talk*



b) Comando *#talk*

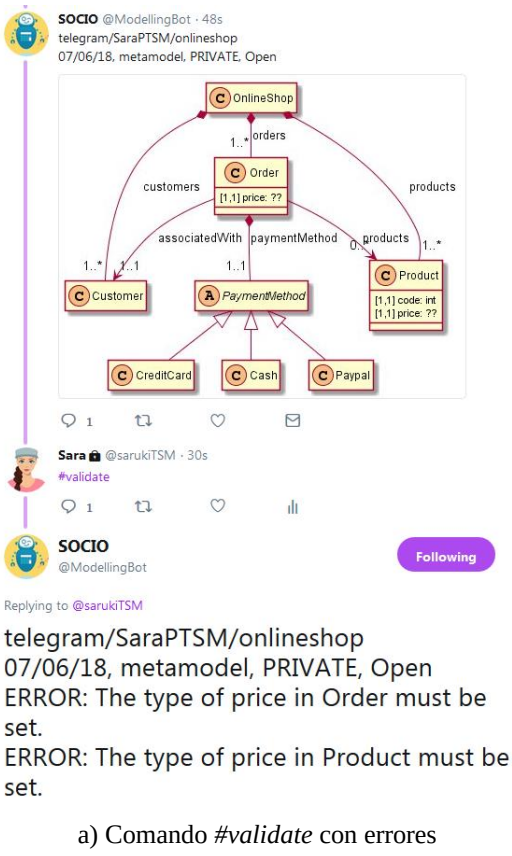


c) Comando *#undo*

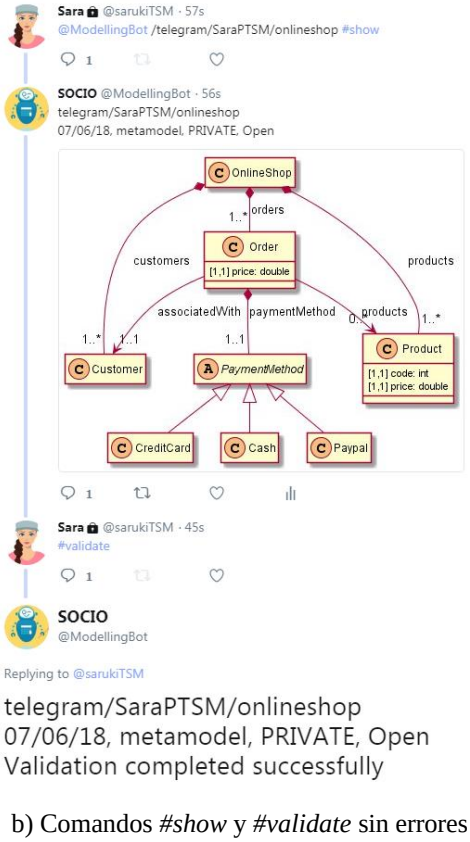


d) Comando *#redo*

Figura 4.7: Ejemplo del uso de SOCIO en Twitter. Comandos *projects*, *talk*, *undo* y *redo*

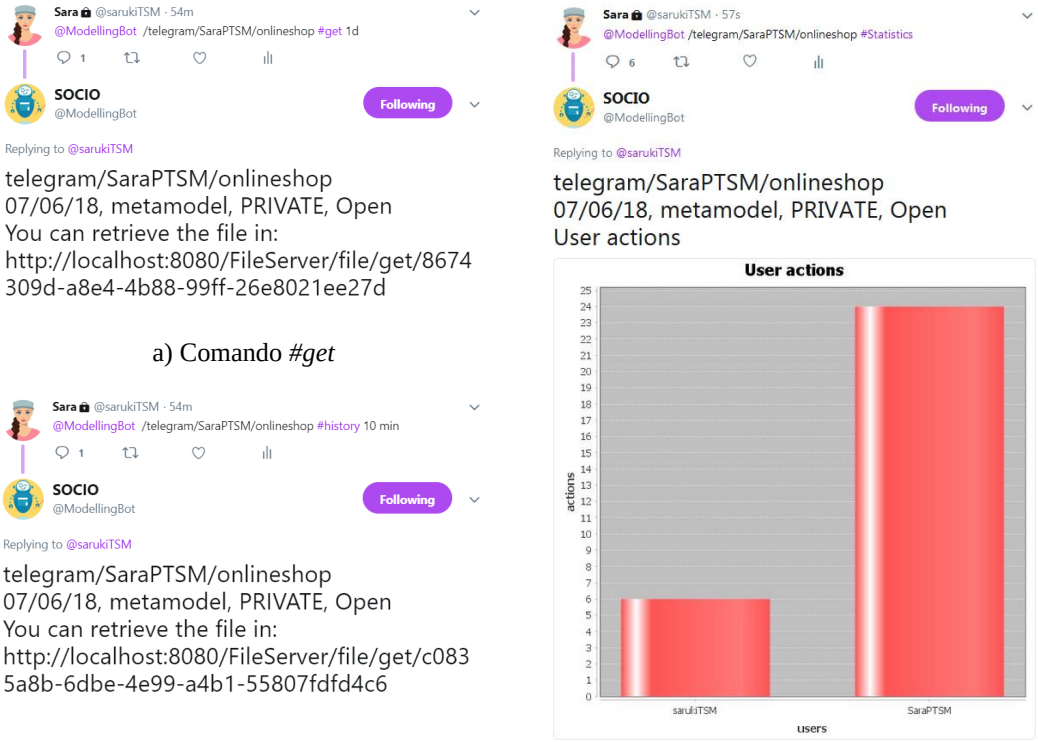


a) Comando *#validate* con errores



b) Comandos *#show* y *#validate* sin errores

Figura 4.8: Ejemplo del uso de SOCIO en Twitter. Comandos *show* y *validate*



a) Comando *#get*

b) Comando *#history*

c) Comando *#statistics*

Figura 4.9: Ejemplo del uso de SOCIO en Twitter. Comandos *get*, *history* y *statistics*

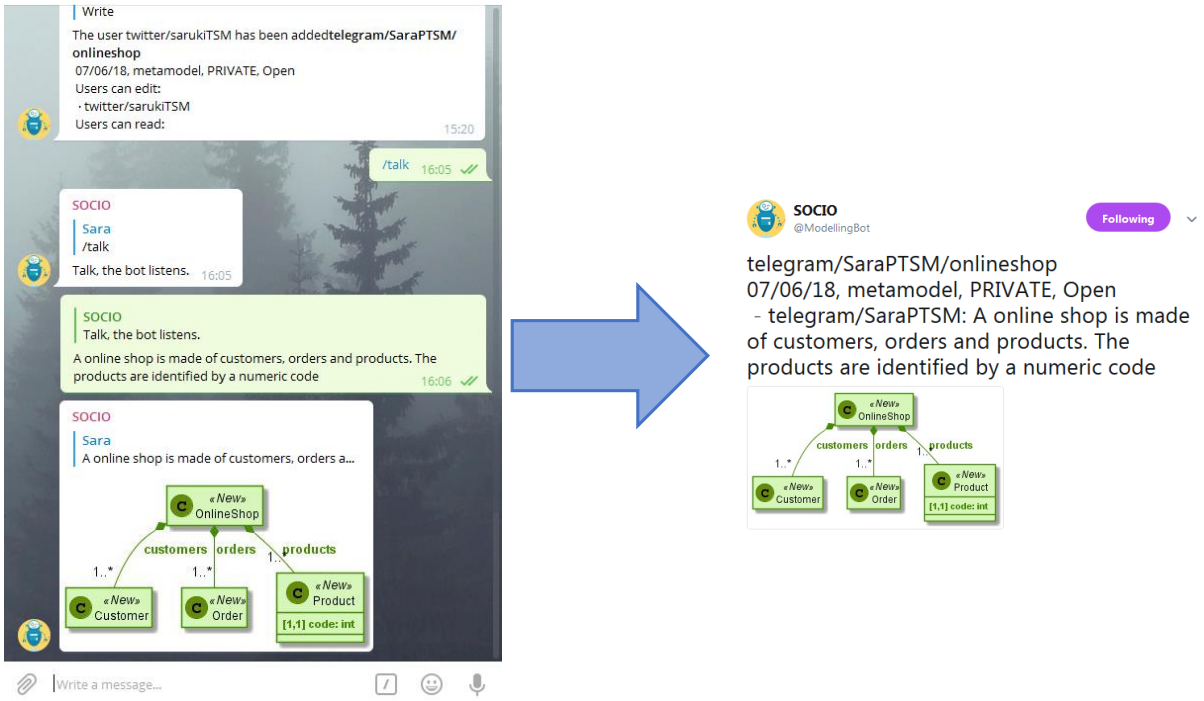


Figura 4.10: Actualización de la información en Twitter cuando el proyecto se modifica en Telegram

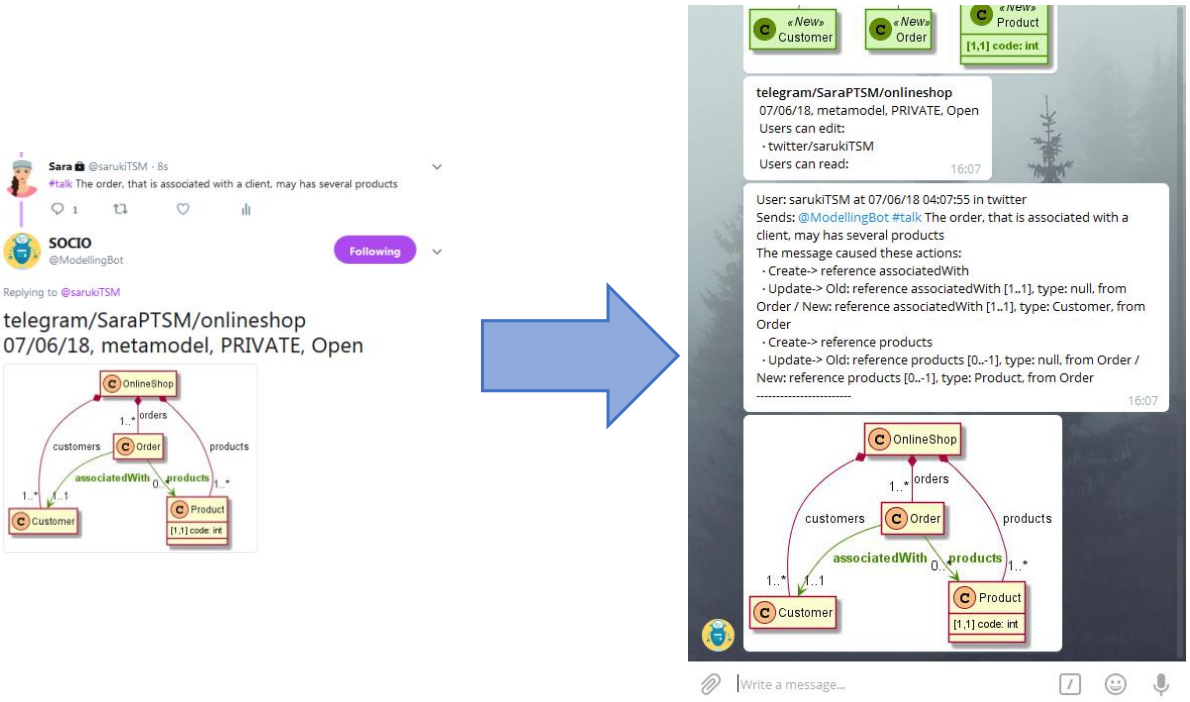


Figura 4.11: Actualización de la información en Telegram cuando el proyecto se modifica en Twitter

5 Evaluación

En está Sección se va a contar el estudio llevado a cabo para evaluar a SOCIO. El estudio se realizó en una fase temprana del desarrollo para evaluar el enfoque. Los pasos que se siguieron son: planteamiento de la evaluación (Sección 5.1), descripción del estudio llevado a cabo (Sección 5.2), obtención e interpretación de los resultados (Sección 5.3) y amenazas a la validez del estudio y discusión (Sección 5.4).

5.1. Planteamiento de la evaluación

Se ha llevado un estudio para evaluar a SOCIO. En el enfoque (Sección 3) se propone un sistema para elaborar modelos de dominio de forma colaborativa sobre redes sociales, usando LN. De este enfoque se puede obtener dos aspectos fundamentales a evaluar: la adecuación del LN para modelar frente a los editores de modelado más convencionales y si la integración con las redes sociales facilita la colaboración. Por otro lado, también se quiere evaluar la herramienta implementada: la usabilidad, la precisión del procesamiento del LN y si la funcionalidad proporcionada es suficiente. Por lo tanto, nuestro estudio busca respuesta a las siguientes preguntas de investigación:

- RQ1: ¿Es el LN adecuado y suficiente para realizar tareas de modelado?
- RQ2: ¿La integración de la herramienta con las redes sociales favorece la tarea de modelado colaborativo?
- RQ3: ¿La herramienta implementada es adecuada y cumple las expectativas del enfoque?

5.2. Descripción del estudio

Para llevar a cabo este estudio se ha contado con la participación de 10 voluntarios, todos ellos graduados o estudiantes del último curso de Grado en Ingeniería Informática. Se les dividió en cuatro grupos, dos grupos de dos personas y dos grupos de tres personas, y se les pidió que realizaran una tarea de modelado de forma colaborativa usando a SOCIO en Telegram. La tarea consistía en crear un modelo para una plataforma de venta online. A los usuarios se les describió verbalmente esta tarea. A continuación hay una breve descripción de la plataforma:

Mientras los clientes navegan por el sitio, pueden seleccionar los productos deseados, los cuales contienen un código y un precio, que se incluirá en el pedido. Cuando cliente realiza la compra, el pedido se asocia con él, y puede realizar el pago con tarjeta de crédito, paypal o en efectivo.

Para realizar la tarea, primero se explicó brevemente, en diez minutos, cómo funcionaba el bot en Telegram, se les dejó quince minutos para realizar la tarea y se les pidió que contestaran a un cuestionario. El cuestionario completo se encuentra en el Apéndice A. En la primera parte del cuestionario se recoge información sobre el participante: si es estudiante o graduado en informática, la versión de Telegram que ha usado durante el estudio (en SmartPhone, versión Web o versión de Escritorio), el grado de uso en redes sociales (del 1 al 5, 1 poco o ninguno y 5 intensivo), el grado de conocimiento sobre modelado (del 1 al 5, 1 inexperto y 5 experto) y por último el grado de conocimiento de inglés, pues al bot es necesario dirigirse en inglés (del 1 al 5, 1 inexperto y 5 experto).

La segunda y la tercera parte del cuestionario contienen preguntas que se evalúan con la escala Likert, en la que se expresa el nivel de acuerdo o desacuerdo con una declaración, del 1 al 5, siendo 1 totalmente en desacuerdo y 5 totalmente de acuerdo. En la segunda parte se realiza el cuestionario de la Escala de Usabilidad del Sistema (System Usability Scale - SUS) [25], un estandar para medir la usabilidad de productos software. Como los participantes eran hispano hablantes, y el cuestionario original está en inglés, ha sido traducido al español.

La tercera parte consta de ocho cuestiones, específicas para la herramienta, en las que se quiere medir: (1) lo apto del LN para modelar frente a los editores de modelos más convencionales (preguntas **Q11** y **Q12**), (2) la precisión del bot para interpretar el LN (preguntas **Q13 y *Q14***), (3) si SOCIO tiene suficiente funcionalidad para crear modelos (preguntas **Q15** y **Q16**), y (4) si les resulta útil combinar una herramienta de modelado con las redes sociales o prefieren una herramienta de modelado colaborativo separada (preguntas **Q17** y **Q18**). La última parte del cuestionario son tres preguntas abiertas en las que se pregunta los puntos fuertes y débiles de la herramienta y qué aspectos mejorar.

El cuestionario se realizó en papel y no se les puso a los participantes limitaciones de tiempo.

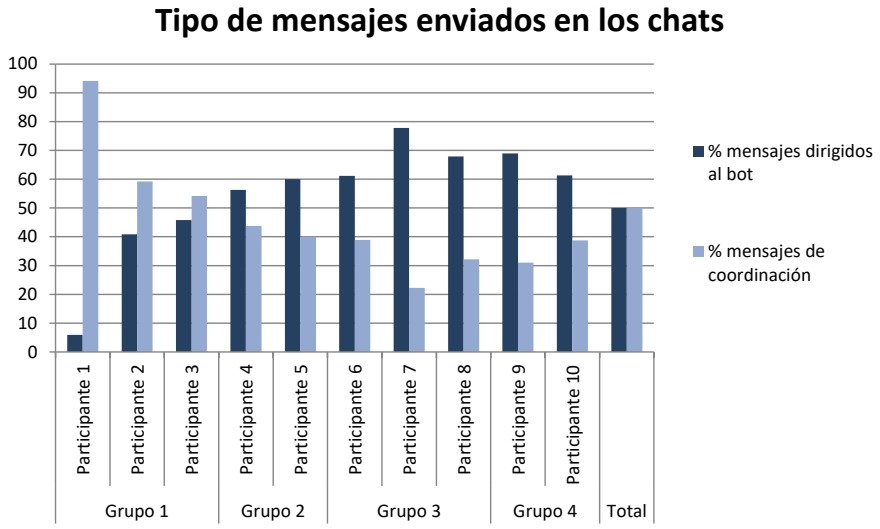


Figura 5.1: Gráfica del porcentaje de mensajes dirigidos al bot y de coordinación.

5.3. Resultados de la evaluación

De los mensajes enviados en los grupos de Telegram, el 50 % de ellos fueron mensajes intercambiados por los participantes para coordinar la construcción del modelo y el 50 % dirigidos al bot para construir el modelo. En la Figura 5.1 se muestra una gráfica con el porcentaje mensajes tanto de coordinación como dirigidos al bot para cada participante. En la gráfica se ve como en la mayoría de los grupos el porcentaje de mensajes enviados al bot, es igual o ligeramente mayor al porcentaje de coordinación, excepto en el *Grupo 1*, en la que el *Participante 1* tomó el papel de coordinador y mandaba mensajes al resto de participantes, mientras el resto de participantes del grupo crean el modelo.

En la primera parte del cuestionario se obtuvo que todos los participantes son estudiantes o graduados en informática, y que seis de ellos usaron la aplicación de Telegram en SmartPhone, 2 la aplicación de Escritorio y 2 la aplicación Web. La Tabla 5.1 muestra la media, la mediana y la desviación típica de la información de los participantes. En la gráfica no hay una gran diferencia entre las medias y las medianas. Las desviaciones son muy similares en los tres casos, y no muy altas. Para el nivel de uso de redes sociales se puede ver una media de 3.2 y una mediana de 3 sobre 5, lo implica un uso de redes sociales medio. La desviación típica (1.13) en este caso es la más alta de las tres, es decir, es la pregunta con más dispersión de resultados de las tres. El conocimiento de modelado tiene una media de 3.5 y la misma mediana, es decir, un conocimiento medio. Por último, el conocimiento de inglés es la media y mediana más alta de las tres, 3.9 y 4, un conocimiento medio alto. El que las desviaciones típicas sean relativamente bajas en este apartado y que todos sean informáticos, indica que los participantes tienen un perfil bastante similar.

La Tabla 5.2 muestra los valores de la media, mediana y la desviación típica de las preguntas realizadas para evaluar la usabilidad de la herramienta. La cuestión con los valores de media y mediana más altos es "**Q7**: Creo que la mayoría de la gente puede

	Media	Mediana	Desviación típica
uso de redes sociales	3,2	3	1,1352924
conocimiento modelado	3,5	3,5	0,8498366
conocimiento inglés	3,9	4	0,7378648

Tabla 5.1: Media, mediana y desviación típica de la información recogida de los participantes.

	Media	Mediana	Desviación típica
Q1	3,8	4	0,6324555
Q2	1,6	1,5	0,6992059
Q3	3,6	4	0,6992059
Q4	2,5	2	0,9718253
Q5	4	4	0,8164966
Q6	1,7	2	0,6749486
Q7	4,2	4,5	1,0327956
Q8	1,9	1,5	1,1005049
Q9	3,5	3	0,9718253
Q10	1,8	2	0,7888106

Tabla 5.2: Media, mediana y desviación típica del cuestionario de la Escala de Usabilidad del Sistema

aprender a usar esta herramienta muy rápidamente", y la de los valores más bajos es "**Q2**: Encontré esta herramienta innecesariamente compleja". Es decir, los participantes consideraron que la herramienta es rápida de aprender y sencilla. La usabilidad del sistema percibida es del 74 %, lo que indica una usabilidad buena.

La Tabla 5.3 contiene la media, mediana y desviación típica de las preguntas concretas de la herramienta. La pregunta "**Q14**: He encontrado desajustes entre las oraciones escritas y los modelos generados" tiene resultados altos, lo que nos indica necesidad de mejora en la implementación del procesamiento de LN. El resultado porcentual obtenido en los aspectos que quiere medir este apartado es: de un 75 % para la cuestión (1) lo apto del LN para modelar frente a los editores de modelos más convencionales, un 56 % para (2) la precision del bot para interpretar el LN, un 60 % para (3) si SOCIO tiene suficiente funcionalidad para crear modelos, y un 76 % para (4) si les resulta útil combinar una herramienta de modelado con las redes sociales o prefieren una herramienta de modelado colaborativo separada.

En cuanto a las cuestiones de respuesta abierta, el punto fuerte en el que coinciden la mayoría de los participantes es la facilidad de uso de la herramienta, lo que concuerda con los resultados obtenidos en las preguntas previas. También se destaca la rapidez con la que se crean los modelos, no hay que tener mucha experiencia para crear modelos y la posibilidad en dispositivos móviles como puntos positivos. Como aspectos negativos los más destacados de son: problemas con para especificar la cardinalidad de los atributos o las referencias, que a la hora de visualizar los proyectos y el historial es un poco confuso y problemas con la interpretación del LN. Un aspecto a mejorar, que aunque no ha sido el

	Media	Mediana	Desviación típica
Q11	3,8	4	0,7888106
Q12	1,8	2	0,421637
Q13	3,5	3,5	0,8498366
Q14	3	3	1,1547005
Q15	3,889	4	0,9938081
Q16	3,1	3	1,197219
Q17	4,3	4	0,6749486
Q18	2,2	2	0,9189366

Tabla 5.3: Media, mediana y desviación típica de las preguntas concretas de la herramienta

más comentado si que es interesante como trabajo futuro es el incluir vocabulario técnico en el procesamiento del LN, como por ejemplo *create containment reference from... to....*

5.3.1. Respuesta de las preguntas de investigación

En cuanto a las preguntas de investigación planteadas en la Sección 5.1, con los resultados obtenidos se pueden contestar de la siguiente manera:

- RQ1: ¿Es el LN adecuado y suficiente para realizar tareas de modelado? El 75 % obtenido en el primer aspecto específicos de la herramienta (cuestiones **Q11** y **Q12**) indica es el LN es adecuado para realizar las tareas de modelado, por lo que es un punto positivo de nuestro enfoque.
- RQ2: ¿La integración de la herramienta con las redes sociales favorece la tarea de modelado colaborativo? El segundo punto que queríamos medir del enfoque, también sale positivo en la evaluación con un 76 % (cuestiones **Q17** y **Q18**).
- RQ3: ¿La herramienta implementada es adecuada y cumple las expectativas del enfoque? En cuanto a la implementación de la herramienta, tiene una usabilidad buena (74 %), la funcionalidad es suficiente, y el procesamiento del LN aceptable, aunque es necesario enriquecerlo.

Los resultados obtenidos, aunque se pueden mejorar, son prometedores y dejan el camino abierto para futuras investigaciones. Principalmente, en lo que hay que mejorar es en el procesamiento de LN.

5.4. Amenazas a la validez del experimento y discusión

A continuación, se analizara las amenazas a la validez del estudio. El estudio que se ha realizado ha sido un estudio preliminar, con un bajo número de participantes, con grupos

de modelado también de número reducido. Además el perfil de todos los participantes es muy similar, por lo que es posible que en perfiles más heterogéneos no se obtengan los mismos resultados. Además, el enfoque del bot, esta dirigido especialmente a expertos de dominio, y en este estudio ninguno de los participantes es expertos de domino, no obstante el dominio de la aplicación es bastante genérico. Por último, los participantes no son de habla nativa inglesa, y al bot hay que dirigirse en inglés, por lo que la formulación de los requisitos ha podido verse afectada.

Por otro lado, al realizarse el experimento en una fase temprana del desarrollo, la mayoría de los aspectos a mejorar, ya han sido modificados en la herramienta presentada en este documento. Por ejemplo, se ha cambiado la visualización del historial. En un principio era a través de imágenes generadas con PlantUML, para facilitar su uso en redes sociales como Twitter, que tiene un límite de caracteres. Pero cuando el historial era muy grande, la visualización se hacía confusa. Actualmente, se muestra en formato texto, con lo que queda más claro, y para solventar el problema con Twitter, se ha añadido un servidor en el cual se puede descargar un fichero de texto que contiene esta información. Cuando se pedía información sobre los proyectos, se mostraba la de todos los proyectos, independientemente de los permisos del usuario, actualmente solo se muestran los proyectos en los cuales el usuario que realiza la petición tiene algún tipo de permiso. La cardinalidad de los atributos y referencias y el procesamiento del LN también han sido modificadas. Por ejemplo, cuando se realizó el estudio SOCIO no soportaba las subordinadas y la cardinalidad solo se obtenía de la pluralidad de los sustantivos, actualmente la subordinadas se soportan dividiendo las frases subordinas en frases independientes, y hay más maneras de expresar cardinalidades, como por ejemplo, adjetivos como *several*, *many* o *much*, o directamente expresando los valores numéricos. No obstante, tanto el procesamiento de LN y la cardinalidad es necesario seguir mejorando. Se tiene planeado como trabajo futuro (ver Sección 6.2) realizar otra experimentación para evaluar todas las modificaciones introducidas en la herramienta.

6

Conclusiones y trabajo futuro

En esta apartado, se van a exponer las conclusiones (Sección 6.1) y el trabajo futuro (Sección 6.2).

6.1. Conclusiones

Para este trabajo se ha propuesto un enfoque para modelado colaborativo por medio de chatbots en redes sociales. Este enfoque se aprovecha de los mecanismos de discusión y coordinación inherentes de las redes sociales. Para realizar tareas de modelado los usuarios expresan requisitos en LN y el sistema los procesa y genera el modelo. De esta forma, se logra acercar la tarea a colaboradores con baja experiencia en el modelado, como pueden ser los expertos de dominio.

Para probar el enfoque se ha desarrollado una herramienta llamada SOCIO. Un bot que funciona sobre las redes sociales de Telegram y Twitter, y es ampliable fácilmente a otras, de manera externa. Los modelos en la herramienta se gestionan dentro de proyectos. Los proyectos pueden tener restricciones de visibilidad y los usuarios permisos para acceder a los proyectos restringidos. La herramienta facilita funcionalidades de gestión de los proyectos, así como mecanismos de procesamiento de requisitos que junto con el estado actual de modelo, desencadenas una serie de acciones para actualizarlo. Estas acciones pueden deshacerse o rehacer. Por último la herramienta guarda un historial con todos los mensajes enviados, permitiendo obtener información acerca de la trazabilidad del modelo o estadísticas del trabajo de los usuarios en el modelo. Para probar el enfoque, se realizó una evaluación preliminar con resultados muy prometedores.

6.2. Trabajo futuro

Este trabajo se continuará como proyecto de tesis doctoral. Los puntos más importantes que se van a trabajar son:

- Elaborar un sistema de propuesta de alternativas y de toma de decisión en grupo a través de consenso suave. Este punto se encuentra actualmente en desarrollo , pero ha dado lugar a la publicación:

  "**Collaborative modelling and group decision-making using chatbots within social networks**". Sara Pérez, Esther Guerra, Juan de Lara. IEEE Software, special issue on collaborative modelling (en prensa).

- Ya que la evaluación realizada, fue en una fase muy temprana del desarrollo y con pocos usuarios para evaluar la viabilidad del enfoque, se planea realizar otro estudio que permita evaluar las modificaciones introducidas a la herramienta.

- Ampliar el procesamiento de LN para que permita instanciar los modelos de dominio definidos.

- Implementar generadores de código a partir de los modelos, e integrarlos con el flujo de trabajo del bot.

- Integrar otros servicios de modelado, como por ejemplo herramientas para el análisis de calidad de los modelos, dentro de las redes sociales, también en forma de bots o chatbots.

Bibliografía

[1] Vicente Pelechano Antonio Valecillos Juan Manuel Vara Cristina Vicente-Chicote Jesús García Molina Félix O. García Rubio. *Desarrollo del software dirigido por modelos: Conceptos, métodos y herramientas*. Ra-Ma Editorial, 2013. ISBN: 978-84-9964-215-4.

[2] A. Cicchetti y col. «Automating Co-evolution in Model-Driven Engineering». En: *2008 12th International IEEE Enterprise Distributed Object Computing Conference*. 2008, págs. 222-231. DOI: 10.1109/EDOC.2008.44.

[3] Christof Ebert, Marco Kuhrmann y Rafael Prikladnicki. «Global Software Engineering: Evolution and Trends». En: *11th IEEE International Conference on Global Software Engineering, ICGSE*. 2016, págs. 144-153.

[4] André van der Hoek Jim Whitehead Ivan Mistrík John Grundy. *Collaborative Software Engineering: Challenges and Prospects*. Springer, Berlin, Heidelberg, 2010. ISBN: 978-3-642-10294-3. DOI: https://doi.org/10.1007/978-3-642-10294-3_19.

[5] Robert O. Briggs Barry Boehm Paul Grünbacher. «Developing groupware for requirements negotiation: lessons learned». En: *IEEE Software* 18.3 (mayo de 2001), págs. 46 -55. DOI: https://doi.org/10.1109/52.922725.

[6] Ivano Malavolta Davide Di Ruscio Mirco Franzago. «Envisioning the Future of Collaborative Model-Driven Software Engineering». En: *Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on* 39 (2017), págs. 219-221. DOI: https://doi.org/10.1109/ICSE-C.2017.143.

[7] M. Franzago y col. «Collaborative Model-Driven Software Engineering: a Classification Framework and a Research Map». English. En: *IEEE Transactions on Software Engineering* (2019). ISSN: 0098-5589. DOI: 10.1109/TSE.2017.2755039.

[8] Margaret-Anne Storey y col. «The (R) Evolution of Social Media in Software Engineering». En: *Proceedings of the on Future of Software Engineering*. FOSE 2014. Hyderabad, India: ACM, 2014, págs. 100-116. ISBN: 978-1-4503-2865-4. DOI: 10.1145/2593882.2593887. URL: http://doi.acm.org/10.1145/2593882.2593887.

[9] Megan Squire. «Should We Move to Stack Overflow?: Measuring the Utility of Social Media for Developer Support». En: *Proceedings of the 37th International Conference on Software Engineering - Volume 2*. ICSE '15. Florence, Italy: IEEE Press, 2015, págs. 219-228. URL: http://dl.acm.org/citation.cfm?id=2819009.2819042.

[10] Bin Lin y col. «Why Developers Are Slacking Off: Understanding How Software Teams Use Slack». En: *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. CSCW '16 Companion. San Francisco, California, USA: ACM, 2016, págs. 333-336. ISBN: 978-1-4503-3950-6. DOI: 10.1145/2818052.2869117. URL: http://doi.acm.org/10.1145/2818052.2869117.

[11] Tian y col. «APIBot: question answering bot for API documentation». En: *ASE*. ACM, 2017, págs. 153-158.

[12] Beschastnikh, Lungu y Zhuang. «Accelerating Software Engineering Research Adoption with Analysis Bots». En: *ICSE-NIER*. IEEE, 2017, págs. 35-38.

[13] Petra Brosch y col. «An Introduction to Model Versioning». En: *Proceedings of the 12th International Conference on Formal Methods for the Design of Computer, Communication, and Software Systems: Formal Methods for Model-driven Engineering*. SFM'12. Bertinoro, Italy: Springer-Verlag, 2012, págs. 336-398. ISBN: 978-3-642-30981-6. DOI: 10.1007/978-3-642-30982-3_10. URL: http://dx.doi.org/10.1007/978-3-642-30982-3_10.

[14] Amanuel Alemayehu Koshima y Vincent Englebert. «Collaborative editing of EMF/Ecore meta-models and models: Conflict detection, reconciliation, and merging in DiCoMEF». En: *Science of Computer Programming* 113 (2015). Model Driven Development (Selected & extended papers from MODELSWARD 2014), págs. 3 -28. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2015.07.004. URL: http://www.sciencedirect.com/science/article/pii/S0167642315001380.

[15] Filipe Del Nero Grillo y Renata Pontin de Mattos Fortes. «Accessible Modeling on the Web: A Case Study». En: *Procedia Computer Science* 27 (2014). 5th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion, DSAI 2013, págs. 460 -470. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2014.02.050. URL: http://www.sciencedirect.com/science/article/pii/S1877050914000520.

[16] Csaba Debreceni y col. «The MONDO Collaboration Framework: Secure Collaborative Modeling over Existing Version Control Systems». En: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2017. Paderborn, Germany: ACM, 2017, págs. 984-988. ISBN: 978-1-4503-5105-8. DOI: 10.1145/3106237.3122829. URL: http://doi.acm.org/10.1145/3106237.3122829.

[17] Raphael Mannadiar Conner Hansen Van Mierlo Eugene Syriani Hans Vangheluwe y Huseyin Ergin. «AToMPM: A Web-based Modeling Environment». En: *In Proc of MODELS Satellite Events (CEUR Workshop Proceedings)*. 2013, págs. 21-25. URL: http://ceur-ws.org/Vol-1115..

[18] M. Ferenc, I. Polasek y J. Vincur. «Collaborative Modeling and Visualization of Software Systems Using Multidimensional UML». En: *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. 2017, págs. 99-103. DOI: 10.1109/VISSOFT.2017.19.

[19] Y. Jiang y col. «SCCMT: A Stigmergy-Based Collaborative Conceptual Modeling Tool». En: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 2016, págs. 401-404. DOI: 10.1109/RE.2016.27.

[20] Margaret-Anne Storey y Alexey Zagalsky. «Disrupting Developer Productivity One Bot at a Time». En: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: ACM, 2016, págs. 928-931. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2983989. URL: http://doi.acm.org/10.1145/2950290.2983989.

[21] M. Marneffe, B. Maccartney y C. Manning. «Generating Typed Dependency Parses from Phrase Structure Parses». En: *Proc. LREC*. https://nlp.stanford.edu/software/lex-parser.shtml. 2006.

[22] Chetan Arora y col. «Extracting domain models from natural-language requirements: approach and industrial evaluation». En: *Proc. MoDELS*. ACM, 2016, págs. 250-260.

[23] George A. Miller. «WordNet: A Lexical Database for English». En: *Comm. ACM* 38.11 (1995). https://wordnet.princeton.edu/, págs. 39-41.

[24] David Steinberg y col. *EMF: Eclipse Modeling Framework 2.0*. 2nd. Addison-Wesley Professional, 2009. ISBN: 0321331885.

[25] John Brooke. «SUS: A Retrospective». En: *J. Usability Studies* 8.2 (2013), págs. 29-40. ISSN: 1931-3357.

Apéndices

A

Cuestionario de evaluación

Cuestiones generales: Para cada una de las siguientes cuestiones marca la casilla correspondiente.

¿Eres estudiante o graduado en informática? ..... Si ☐ No ☐

Versión de Telegram empleada durante la sesión: ..... SmartPhone o Tablet ☐ Escritorio ☐ Web ☐

Puntúa tu grado de uso de las Redes Sociales (1-poco/ninguno, 5-intensivo)	1	2	3	4	5
Puntúa tu grado de conocimiento sobre Modelado (1-novato, 5-experto)	1	2	3	4	5
Puntúa tu nivel de conocimiento de Inglés (1-novato, 5-experto)	1	2	3	4	5

Instruciones: Para las siguientes afirmaciones, marca la casilla que mejor describa tus reacciones a la herramienta

	totalmente de acuerdo → ← totalmente en desacuerdo				
Q1: Creo que me gustaría usar esta herramienta con frecuencia	1	2	3	4	5
Q2: Encontré esta herramienta innecesariamente compleja	1	2	3	4	5
Q3: Creo que la herramienta es fácil de usar	1	2	3	4	5

Q4: Creo que necesitaría ayuda para poder usar esta herramienta	1	2	3	4	5
Q5: He encontrado que las diversas funciones de esta herramienta estaban bien integradas	1	2	3	4	5
Q6: Creo que hay demasiadas funciones inconsistentes en esta herramienta	1	2	3	4	5
Q7: Creo que la mayoría de la gente puede aprender a usar esta herramienta muy rápidamente	1	2	3	4	5
Q8: He encontrado esta herramienta muy engorrosa/incómoda de usar	1	2	3	4	5
Q9: Me sentí muy seguro de lo que hacía al usar esta herramienta	1	2	3	4	5
Q10: Tengo que aprender un montón de cosas antes de poder usar esta herramienta	1	2	3	4	5

Q11: Creo que el lenguaje natural y los comandos proporcionan suficiente funcionalidad para realizar la tarea	1	2	3	4	5
Q12: He echado en falta un editor de modelos en lugar del lenguaje natural y los comandos mientras realizaba la tarea	1	2	3	4	5
Q13: Creo que los modelos generados coinciden con las oraciones escritas	1	2	3	4	5
Q14: He encontrado desajustes entre las oraciones escritas y los modelos generados	1	2	3	4	5
Q15: La herramienta tiene suficiente funcionalidad para crear modelos	1	2	3	4	5
Q16: He echado de menos ciertas funcionalidades durante la realización de la tarea de modelado	1	2	3	4	5
Q17: Creo que la combinación de redes sociales y este tipo de herramientas puede favorecer la tarea de modelado colaborativo	1	2	3	4	5
Q18: Preferiría un editor de modelado colaborativo específico, más que el uso de redes sociales	1	2	3	4	5

Por favor, indica tres aspectos positivos que quieras resaltar sobre la herramienta:

Por favor, indica tres aspectos negativos de la herramienta:

_____

_____

_____

¿Tienes alguna sugerencia de mejora?:

_____