CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Creación del plugin OnToology para la herramienta Protégé

Autor: Ángel del Castillo Las Heras

Director: Óscar Corcho García

MADRID, JUNIO 2017

ÍNDICE

RESUMEN

En este proyecto se plantea la creación del plugin OnToology para el editor de ontologías Protégé. OnToology, tal y como lo describen sus autores, es un servicio para automatizar parte del proceso colaborativo de desarrollo de una ontología. Así, este proyecto plantea integrarlo en Protégé como alternativa a su uso desde un navegador web.

En primer lugar, se realiza un estudio del estado del arte para, posteriormente, describir el proceso seguido en la creación del plugin, así como las herramientas empleadas en el mismo. Dado que aún no existe una API para conectarse a OnToology, la implementación del plugin resulta aún imposible, al menos completamente. Así, se ha puesto el foco en el diseño de la herramienta, la parte donde se toman la mayoría de las grandes decisiones. Se han definido diferentes casos de uso que describen cómo ha de comportarse el plugin en distintos escenarios.

Para finalizar, se presentan conclusiones y propuestas para futuros trabajos.

ABSTRACT

In this project, the creation of the plugin OnToology for the ontology editor Protegé is suggested. OnToology, as described by its authors, is a system to automate part of the collaborative ontology development process. Thus, this project proposes its integration with Protégé as an alternative to the use via a web browser.

In the first place, a research on the state of the art is done. Afterwards, the process followed in the plugin creation is described, as well as the tools used in it. Due to the fact that an API for connecting to OnToology has not been developed yet, the complete implementation of the plugin emerges impossible. Therefore, the focus has been put on the design of the tool, where most of the important decisions need to be taken. Several use cases are defined in order to describe how the plugin should behave in certain situations.

Finally, conclusions and future lines of work are presented.

1. INTRODUCCIÓN

Dentro del amplio mundo de la inteligencia artificial existe un campo que corresponde al desarrollo de ontologías. Pero ¿qué es una ontología? No existe una definición clara y concisa para este concepto; de hecho, la única acepción que aporta el diccionario de la RAE no es aplicable a la informática [1]. No obstante, uno de los principales diccionarios en lengua inglesa, concretamente el de Oxford, sí cuenta con una definición además de la netamente filosófica. Traducida al castellano, es la siguiente: *conjunto de conceptos y categorías dentro de un ámbito o dominio que muestra sus propiedades y las relaciones entre las mismas.* [2]

Podemos encontrar también definiciones dadas por expertos en el área. Algunas son las siguientes:

- *Una ontología define los términos básicos y las relaciones comprendidas en el vocabulario de un ámbito, además de las reglas para combinar términos y relaciones para definir extensiones al vocabulario. [3]*
- *Una ontología es una especificación explícita de una conceptualización. [4]*
- *Una ontología es un conjunto de términos estructurado jerárquicamente para describir un dominio que pueda ser usado como fundamento para una base de conocimiento. [5]*

De la misma manera que existen editores de texto como Emacs o entornos integrados de desarrollo como Eclipse, para construir ontologías se emplean programas denominados editores de ontologías. Estos programas permiten definir una ontología de manera gráfica para posteriormente generar ficheros en formatos estandarizados como OWL. Uno de los editores más famosos es Protégé, desarrollado por la Universidad de Stanford, si bien en versiones antiguas colaboraba también la Universidad de Manchester.

Por otra parte, el *Ontology Engineering Group* de la ETSIINF ha desarrollado una herramienta online denominada OnToology. Dicha herramienta permite, a partir de un repositorio en GitHub que contenga uno o varios ficheros OWL, generar diagramas y documentación, además de ejecutar una validación de la ontología basada en malas prácticas o *pitfalls* comunes.

Actualmente la única manera de usar OnToology es desde la web. En este TFG se pretende desarrollar un plugin para Protégé que haga posible usar OnToology desde el propio editor. De esta manera no se descubrirá nada nuevo, pero sí se aumentará considerablemente la productividad de las personas dedicadas al desarrollo de ontologías al poder realizar todo el proceso desde el mismo programa. Además, el plugin integrará funcionalidades básicas como cliente Git. De esta manera, además de obtener una capa de abstrac-

ción, se evitará que se tenga que usar git desde la línea de comandos o desde una aplicación gráfica adicional. Esto último es especialmente interesante para potenciales usuarios con escasa experiencia en el uso de git y GitHub.
No obstante, el grueso de este TFG no es la implementación del plugin, sino el proceso de ingeniería de software previo al mismo, donde se toman decisiones vitales en las etapas siguientes. Además, la implementación completa del plugin resulta imposible en la actualidad, debido a que no existe una API para comunicarse con OnToology.
Por último, a pesar de que este proyecto no es ni mucho una disertación sobre inteligencia artificial y ontologías, sí que es conveniente poseer nociones básicas acerca del desarrollo de las mismas. Ello permitirá poder ponerse en la piel de los potenciales usuarios del sistema planteado y, por tanto, realizar sin lugar a dudas un mejor trabajo.

1.2. Experiencia previa relacionada

Como alumno de la ETSIINF, desde el primer curso he aprendido programación. Además, al ser Java el lenguaje empleado para la mayoría de las asignaturas del campo, es en el que poseo una mayor soltura.

Además, también he cursado asignaturas de ingeniería de software. Concretamente en la asignatura de tercer curso Ingeniería de Software I aprendí lo que son los casos de uso y cómo plantearlos. En su momento he de reconocer que no me pareció lo más útil del mundo pero, sin embargo, en este trabajo se ha recurrido a ellos para describir el comportamiento del programa planteado, demostrándome a mí mismo cuan equivocado estaba.

Por otra parte, en tercero cursé la asignatura optativa Sistemas inteligentes, en la cual, entre otras cosas, se enseñan nociones básicas sobre ontologías. De hecho, el proyecto de dicha asignatura consistía en el desarrollo de una ontología empleando Protégé, por lo que ya tenía cierta experiencia con dicho editor. No obstante, esta experiencia era únicamente como usuario y, si bien sirve como punto de entrada, poco tiene que ver con los conocimientos requeridos para el desarrollo de un plugin.

2. Estado del arte

2.1. Sistemas de desarrollo colaborativo de ontologías

Desde finales de la década de los 90 han sido varias las aproximaciones que se han propuesto para transformar el desarrollo de ontologías en una actividad de ingeniería. Una vez conseguida esta meta, se hace necesario adoptar técnicas de ingeniería de software ampliamente usadas por desarrolladores. Algunos ejemplos de adaptación de herramientas comunes en la ingeniería de software al desarrollo colaborativo de ontologías son Vocol y MoKi [6]. No obstante, están lejos de poder ser consideradas una combinación de ambos mundos.

OnToology surge con la intención de facilitar la integración entre el desarrollo de ontologías y la ingeniería de software. Se centra en proveer soporte para las actividades complementarias en el desarrollo de ontologías, como pueden ser el control de versiones, la documentación, la evaluación y la publicación de las mismas. En ningún caso pretende competir con entornos de desarrollo colaborativo de ontologías como WebProtégé, una adaptación online del editor Protégé con soporte multiusuario. Podríamos considerar que WebProtégé es a los editores de ontologías lo que Google Docs a los procesadores de textos. Permite, entre otras cosas, llevar un control de los cambios, conocer la autoría de los mismos, controlar qué usuarios pueden acceder a cada ontología y con qué permisos o recibir notificaciones vía email cada vez que una ontología en cuyo desarrollo participamos sea modificada.

OnToology es una aplicación web de código abierto que detecta los cambios realizados en las ontologías de un repositorio git y realiza consecuentemente una serie de acciones orientadas a facilitar el proceso de evaluación y publicación de las mismas. Además, también permite que sea el propio usuario el que fuerce dichas acciones. Actualmente se integra únicamente con la plataforma GitHub. No obstante, se pretende que, en un futuro, esta integración pueda realizarse también con otros servicios similares como Bitbucket o GitLab. [6] En la siguiente figura, realizada por María Poveda-Villalón y Ahmad Alobaid, del *Ontology Engineering Group*; podemos ver las interacciones que se producen entre el usuario, OnToology y GitHub.

Podemos observar cómo las acciones se pueden dividir en dos grandes grupos: las que fuerza el usuario (*user triggered*) y las que realiza OnToology de manera automática una vez que ha obtenido acceso al repositorio en GitHub (*triggered by ontology change*).

Se observa asimismo como OnToology, a su vez, depende de otros servicios externos, a saber: AR2DTool[1], OOPS![2], Widoco[3] y VocabLite[4].

[1] AR2DTool: https://github.com/idafensp/ar2dtool
[2] OOPS!: http://oops.linkeddata.es/
[3] Widoco: https://github.com/dgarijo/Widoco
[4] VocabLite: https://github.com/dgarijo/vocabLite

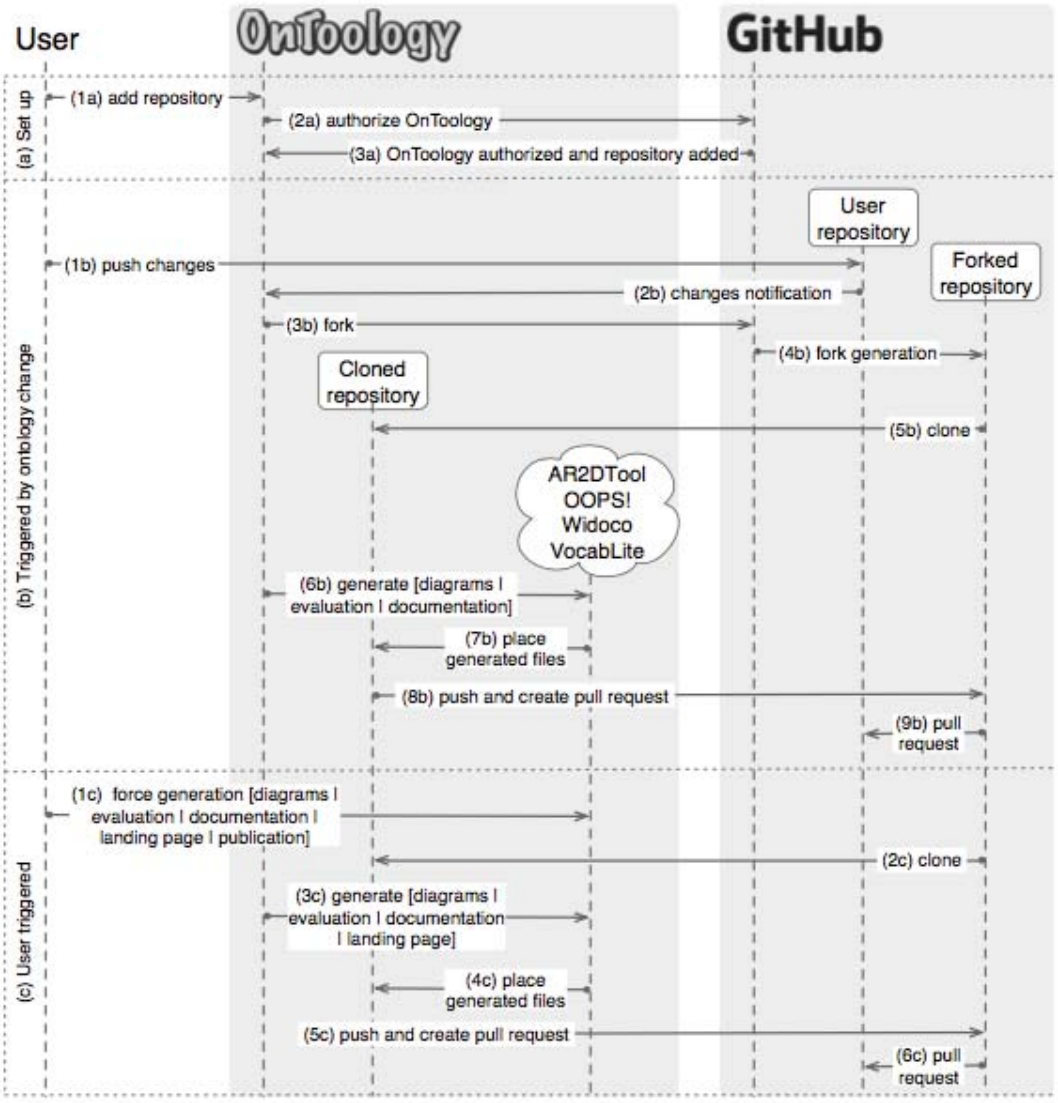

FIG 1: INTERACCIONES ENTRE EL USUARIO, ONTOOLOGY Y GITHUB [7]

2.2. PLUGINS PARA PROTÉGÉ

Desde el principio, Protégé se desarrolló como un sistema completamente modular, abriendo así la puerta a que la comunidad desarrollara plugins. Además, existe un repositorio oficial donde los desarrolladores pueden publicarlos, la *Protégé Plugin Library*. De esta manera es muy fácil que terceras personas los encuentren.

El lenguaje elegido en el desarrollo de Protégé es Java, por lo que los plugins también se desarrollan en dicho lenguaje para poder usar la API e integrarse de manera adecuada.

Existe documentación oficial para desarrolladores de Protégé donde se explica cómo compilar las fuentes de Protégé, como integrarlo en un entorno de desarrollo como

Eclipse para poder usar sus librerías, los diferentes componentes de un plugin y las distintas posibilidades que ofrece la API [8]. Esta documentación es muy completa en anchura, si bien en profundidad hay aspectos que podrían estar mejor explicados.

3. Tecnologías empleadas en el desarrollo

3.1. Lenguaje de programación

Como se ha dicho en puntos anteriores de la presente Memoria, Protégé es un software escrito en Java. Así, para desarrollar un plugin se debe emplear dicho lenguaje. En mi caso esto está lejos de ser un problema, ya que Java es el lenguaje con el que aprendí a programar en el primer curso del Grado y, por tanto, me resulta totalmente familiar. Es más, es un lenguaje que me planteo elegir para casi cualquier proyecto de ingeniería de software al que me tenga que enfrentar.

Java es un lenguaje de programación creado en el año 1995 por *Sun Microsystems*, compañía que fue comprada por *Oracle* en 2010. Se trata de un lenguaje multiparadigma y orientado a objetos. Se caracteriza por su ejecución en una máquina virtual, llamada JVM (Java Virtual Machine). El código fuente se compila y se transforma en un código intermedio, *bytecode*, que es ejecutado por la JVM. Esto permite que el mismo código se pueda ejecutar en máquinas con diferentes sistemas operativos y arquitecturas y se conoce bajo el acrónimo WORA, del inglés *Write Once, Run Anywhere*.

3.2. Bibliotecas y APIs

3.2.1. Protégé

Al ser el programa objeto de este TFG un plugin para Protégé, es necesario emplear las bibliotecas de dicho editor. Estas están disponibles en GitHub[5], ya que Protégé es un software de código abierto.

3.2.2. Swing

Para la implementación de la interfaz gráfica de usuario (GUI) se ha optado por una de las bibliotecas presentes en la API de Java: Swing. Esta biblioteca es también la que usa el propio Protégé, por lo que la razón de su elección está clara.

3.2.3. JGit

Esta biblioteca integra un cliente Git completo. Es usada, entre otros proyectos, por el plugin Git del IDE Eclipse. Se usará para las acciones Git en local, si bien para la comunicación con GitHub se empleará la biblioteca descrita en el siguiente apartado.

[5] Protégé en GitHub: https://github.com/protegeproject/protege

3.2.4. GitHub Java API

No todas las comunicaciones del plugin son con el servicio OnToology, también hay algunas que son directamente con GitHub. Esto es así porque se pretende que el plugin tenga también funcionalidades de cliente git básico, algo que no tiene sentido implementar en una aplicación web como OnToology. El objetivo de esto es hacer el proceso de subida de las ontologías a GitHub lo más transparente al usuario posible. De esta manera, se pretende aumentar la productividad al integrar en el plugin y, por tanto, en Protégé, ciertas tareas que de otra manera tendrían que hacerse desde un cliente git externo, ya fuese este gráfico o textual. Todo ello será especialmente útil para aquellos usuarios expertos en el desarrollo de ontologías, pero sin una base en el manejo de sistemas de control de versiones en general y git y GitHub en particular.

Concretamente se pretende que el plugin sea capaz de:

- Realizar un *commit* con los cambios en la ontología que se esté editando en ese momento y hacer el *push* a GitHub.
- Hacer *pull* desde GitHub a nuestra máquina, ya que es muy frecuente que varios usuarios estén trabajando en el mismo repositorio, por lo que debemos asegurarnos siempre de estar trabajando con la última versión.
- Aceptar una *pull request* que se haya hecho previamente mediante una llamada a la API de OnToology. Hasta ahora, las *pull request* hechas desde la aplicación web de OnToology debían ser aceptadas desde GitHub. Esto seguirá siendo posible para las hechas desde el plugin, ya que cabe la posibilidad de que el usuario quiera revisarlas antes de aceptarlas. No obstante, se le dará la posibilidad de aceptarlas directamente desde el plugin, disminuyendo de esta manera la complejidad del proceso y ahorrando tiempo.

GitHub ofrece una API REST. Existen bibliotecas oficiales que implementan la API. No obstante, solo están disponibles para los lenguajes Ruby, Objective-C y .NET [9]. Sin embargo, en la propia web de desarrolladores de GitHub se referencian bibliotecas desarrolladas por terceros. En concreto para el lenguaje Java hay 3, y se ha optado por emplear la GitHub Java API. [6]

3.2.5. OnToology

Para la comunicación con OnToology se pretende usar su API REST. No obstante, al no estar ésta disponible aún, se dejará esta comunicación para trabajos futuros. En cualquier caso, el plugin debe diseñarse de manera que todo esté claro una vez se desarrolle dicha API y pueda ser incluida en el proyecto.

[6] GitHub Java API: https://github.com/eclipse/egit-github/tree/master/org.eclipse.egit.github.core

3.3. Entorno de desarrollo

El IDE elegido ha sido Eclipse por varias razones. En primer lugar, es el entorno con el que más veces he trabajado. Si bien he manejado NetBeans para algún proyecto concreto, es con Eclipse con el que en general me manejo mejor y poseo una mayor soltura. Por otro lado, en la documentación para desarrolladores de Protégé se ofrece ayuda a la hora de configurar Eclipse para integrar todas las bibliotecas necesarias para el desarrollo de un plugin como el que es objeto de este TFG. Si usamos otro IDE este proceso, de por sí más complejo de lo que en un principio pueda parecer, lo sería aún más al tener que realizarse mediante un mecanismo de prueba y error.

Así pues, las dos razones expuestas han hecho que no tuviese ninguna duda a la hora de elegir el entorno de desarrollo en el que programar. En concreto se ha trabajado con la versión Eclipse Neon sobre el sistema operativo macOS Sierra.

3.4. Control de versiones

Con el fin de llevar un control del progreso y para poder revertir de manera rápida fallos encontrados en el código, se ha creado un repositorio git para el código del plugin. Este repositorio se encuentra en local y también en remoto, empleando la plataforma líder en el sector: GitHub.

4. Desarrollo

4.1. Investigación y lectura de documentación

Antes de empezar con el proyecto propiamente dicho, al no haber realizado nunca un proyecto similar, el primer paso consistió en conocer las APIs que se iban a usar en el código con el objetivo de poder hacer un diseño robusto y fiable.

Al ser el objeto de este TFG un plugin para Protégé, el por dónde empezar no podía estar más claro. En la web oficial de Protegé hay una sección de desarrolladores, donde se ofrece documentación, además de una lista de correo electrónico donde plantear dudas. Se puede ver también el archivo de dicha lista, ya que las preguntas más típicas se encuentran ya respondidas. Al principio esta documentación parece muy completa, pues en anchura sin duda lo es. Sin embargo, cuando buscas algo específico es fácil hallar que no está presente, siendo esto un problema importante. Es decir, si bien en anchura la documentación que ofrece Protégé es muy completa, en profundidad no lo es tanto.

Además de con Protégé, el programa desarrollado ha de interactuar con GitHub, por lo que es vital tener claro el funcionamiento de este servicio a nivel de API. Una de las prácticas de la asignatura de tercer curso Sistemas Orientados a Servicios consistía precisamente en desarrollar una API REST, por lo que no he hallado apenas dificultades a la

hora de comprender el funcionamiento de la de GitHub. Dicha API es muy extensa y completa, pero obviamente solo ha sido necesario estudiar lo que iba a tener aplicación en este TFG, que eran aspectos básicos.

Por último, es obviamente vital entender cómo funciona el servicio web OnToology, desarrollado por el Ontology Engineering Group. En la web oficial [7] se explica cómo usarlo. No obstante, para obtener una visión en profundidad que vaya más allá del nivel usuario, el funcionamiento me fue explicado por los miembros del OEG María Poveda Villalón e Idafen Santana Pérez. Me proporcionaron además el *paper* que escribieron cuando se lanzó OnToology, el cual también fue estudiado [6].

4.2. DISEÑO

En esta sección se pueden distinguir tres partes: el diseño de las funcionalidades, el de la interfaz gráfica de usuario y el planteamiento de diferentes casos de uso.

4.2.1. FUNCIONALIDAD

En un principio el plugin iba a ofrecer exactamente las mismas funcionalidades que On-Toology. Posteriormente esto fue modificado: se le añadieron al plugin funcionalidades a nivel de repositorios locales y se dejó para trabajos futuros la plena integración con OnToology. Así pues, las funcionalidades propuestas para la primera versión del plugin son las siguientes:

- Conectarse con GitHub.
- Detectar si la ontología que se está editando está en un repositorio git.
    - Si es así, hacer *commit* y *push* de los cambios en la ontología directamente desde Protégé.
    - Ser asimismo capaces de realizar *pull* del repositorio remoto, puesto que otra persona que trabaje en la misma ontología ha podido realizar cambios desde su máquina, por lo que nuestra copia local estará desactualizada.
- Dejar el entorno preparado para poder aceptar *pull requests* una vez que la API de OnToology esté disponible y, por tanto, éstas puedan ser creadas.

Un aspecto muy importante en esta fase es tener claro con qué servicios externos ha de interactuar el plugin, a saber, OnToology y GitHub. Se debe explicitar contra qué servicio se realiza cada comunicación para reducir al máximo el margen de error en la fase de implementación, donde el coste, tanto en tiempo como en dificultad, de subsanar un fallo

[7] http://ontoology.linkeddata.es/

es mucho mayor. Además, un fallo de diseño puede dar lugar a múltiples fallos en el funcionamiento, perjudicando así gravemente al usuario final.

4.2.2. Interfaz gráfica de usuario (GUI)

Por otra parte, aunque la funcionalidad del plugin no sea aún total, sí que se puede crear una GUI completa. Es decir, puede estar todo adecuadamente maquetado, aunque algunas componentes de la interfaz no realicen ninguna función por debajo.

Existen diferentes tipos de plugins para Protégé [10]. Por la funcionalidad que realiza este plugin, desde el principio quedó claro que debía ser del tipo *WorkspaceTab*. Así, al entrar en Protégé podremos, desde el menú, abrir una pestaña llamada en este caso OnToology desde donde se ejecutará el plugin.

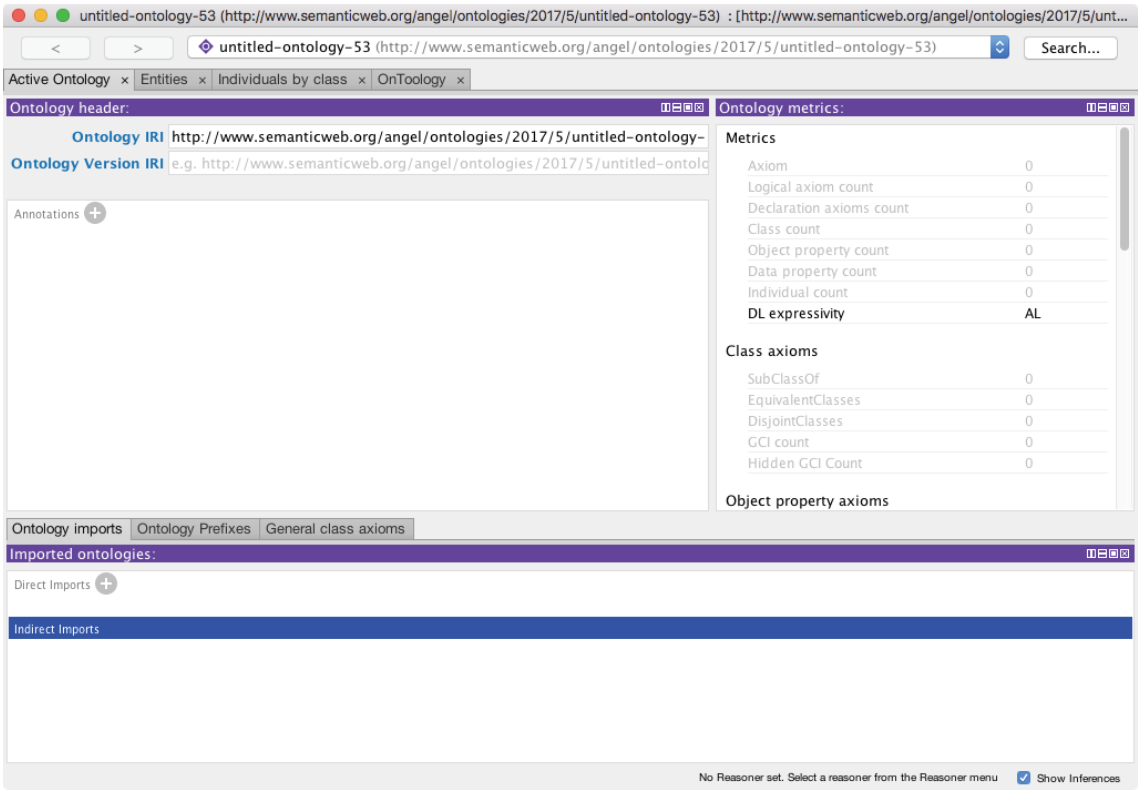

FIG 2: VISTA INICIAL DE PROTÉGÉ. SE OBSERVA LA NUEVA PESTAÑA

Si bien el tipo de plugin quedó claro desde un principio tras documentarse adecuadamente, con el contenido visual del mismo no ocurrió lo mismo. Los bocetos se sucedieron hasta que finalmente se dio con el diseño definitivo.

Finalmente, se decidió que la pestaña *OnToology* contara con dos espacios diferenciados: uno para loguearse en GitHub y otro para mostrar nuestros repositorios que contienen ficheros OWL, es decir, ontologías.

4.2.3. CASOS DE USO

A partir de las funcionalidades descritas anteriormente y la apariencia básica de la interfaz gráfica, se pueden definir diferentes casos de uso para el plugin.

Caso de uso UC1: identificarse en GitHub

Actor principal: usuario del plugin.

Personal involucrado e intereses:

- Usuario: para poder realizar cualquier acción con el plugin, primero deberá completar este caso de uso.

Precondiciones: El usuario posee una cuenta en GitHub.

Garantías de éxito (postcondiciones): se produce el *login* y para cualquier acción posterior que implique una interacción con GitHub se usarán las credenciales introducidas.

Escenario principal de éxito:

1. El usuario escribe su nombre de usuario y su contraseña de GitHub en los cuadros de texto destinados a tal efecto.
2. Hace click en “Log in” y el sistema envía vía API a GitHub las credenciales.
3. GitHub reconoce las credenciales como correctas y el usuario queda identificado.
4. El sistema modifica ligeramente la interfaz gráfica: ya no aparecen los cuadros de texto sino el nombre del usuario logueado. Además, el botón “Log in” pasa a ser “Log out”.

Flujos alternativos.

3. GitHub no reconoce las credenciales como correctas y devuelve un error.
4. El sistema informa al usuario de que el nombre de usuario o la contraseña que ha introducido son incorrectas.

Caso de uso UC2: desconectarse de GitHub

Actor principal: Usuario

Personal involucrado e intereses:

- Usuario: por la razón que sea (por ejemplo, va a dejar su ordenador a otra persona y no quiere arriesgarse a que modifique ontologías y las suba), quiere cerrar sesión en GitHub.

Precondiciones: El usuario está autenticado desde el plugin en GitHub.

Garantías de éxito (postcondiciones): El usuario dejará de estar identificado en GitHub. Como consecuencia, no podrá completar ninguno de los casos de uso que restan.

Escenario principal de éxito:

1. El usuario hace click en “Log out”.
2. El sistema hace una llamada a la API de GitHub para cerrar la sesión.
3. El sistema informa al usuario de que la sesión ha sido cerrada modificando la interfaz gráfica: los cuadros de texto mencionados en UC1 aparecen de nuevo y el botón “Log out” pasa a ser “Log in”.

Caso de uso UC3: actualizar repositorio remoto en GitHub

Actor principal: Usuario que ha realizado cambios en la ontología.

Personal involucrado e intereses:

- Usuario que ha realizado cambios en la ontología: quiere que el resto de miembros de su equipo cuenten con una versión actualizada del repositorio. Además, para poder emplear el servicio OnToology, el repositorio debe estar en GitHub.
- Resto de miembros del equipo: quieren tener disponible, en la medida de lo posible, siempre la última versión.

Precondiciones: El usuario está autenticado desde el plugin en GitHub.

Garantías de éxito (postcondiciones): Se realiza el *commit* local y se sincroniza con el repositorio remoto de GitHub mediante un *push*.

Escenario principal de éxito:

1. El usuario realiza cambios en la ontología desde Protégé.
2. El usuario va a la pestaña OnToology.
3. En la lista de repositorios selecciona el que contiene la ontología editada y hace click en “commit”.
4. En la ventana que aparece escribe el mensaje asociado al commit y pincha en “commit and push”.

5. El sistema le informa por medio de una ventana emergente de que la acción se ha realizado correctamente.

6. El usuario hace click en “OK” y se cierra la ventana.

Caso de uso UC4: hacer *pull* de un repositorio remoto

Actor principal: Usuario que cuenta con una versión antigua del repositorio.

Personal involucrado e intereses:

- Usuario que cuenta con una versión antigua del repositorio: para poder seguir trabajando en una ontología del repositorio en cuestión es obvio que tiene que partir de la última versión de la misma.

Precondiciones: El usuario está autenticado desde el plugin en GitHub.

Garantías de éxito (postcondiciones): se realiza el *pull*. Por consiguiente, el usuario pasa a tener en su ordenador la última versión del repositorio.

Escenario principal de éxito:

1. En la lista de repositorios selecciona el deseado.

2. Hace click en “pull”.

3. Aparece una ventana y el usuario confirma el *pull* pinchando en “confirm”.

4. El sistema le informa de que el *pull* se ha realizado por medio de una ventana emergente.

5. El usuario hace click en “OK” y se cierra la ventana.

Caso de uso UC5: añadir repositorio remoto

Actor principal: Usuario.

Personal involucrado e intereses:

- Usuario: Se acaba de incorporar al equipo de desarrollo de una ontología. Está incluido en el repositorio de GitHub, pero aún no tiene una copia local. OnToology tampoco está supervisando dicho repositorio.

Precondiciones: El usuario está autenticado desde el plugin en GitHub.

Garantías de éxito (postcondiciones): El usuario pasa a tener una copia local del repositorio y OnToology comienza a supervisarlo.

Escenario principal de éxito:

1. El usuario pincha en la opción “add repository”.
2. En el cuadro de texto de la ventana que aparece, introduce la URL del repositorio que quiere añadir, selecciona la ubicación local donde quiere descargarlo y pincha en “add”.
3. El sistema realiza un *git clone* del repositorio en el directorio especificado y realiza una llamada a la API de OnToology para que supervise dicho repositorio.
4. El sistema informa de que el proceso ha finalizado correctamente.
5. El usuario hace click en “OK” y se cierra la ventana.

Caso de uso UC6: producir diagramas, validar las ontologías y/o generar la documentación de una ontología.

Nota: todos los posibles casos de uso en los que la acción principal del sistema es una llamada a la API de OnToology para realizar todas o alguna de las acciones indicadas son idénticos desde el punto de vista de la interacción persona-ordenador, por lo que se consideran como uno solo.

Actor principal: Usuario.

Personal involucrado e intereses:

- Usuario: una vez que la ontología en la que estaba trabajando se encuentra completa (aunque pueda ser ampliada en un futuro) quiere obtener material de soporte para la misma.
- Resto de miembros del equipo: si la ontología se desarrolla de manera colaborativa, tendrán también disponible dicho material.

Precondiciones: El usuario está autenticado desde el plugin en GitHub.

Garantía de éxito (postcondiciones): Se genera una *pull request* al repositorio con el nuevo material de soporte.

Escenario principal de éxito:

1. En la lista de repositorios, el usuario selecciona la ontología deseada y marca las casillas correspondientes a las acciones que desee realizar.
2. Hace click en el botón “Go”, situado a la derecha de dichas casillas.
3. Aparece una ventana de confirmación y hace click en “Generate pull request”.
4. El sistema hace una llamada a la API de OnToology para que se realicen las acciones especificadas y se genere la *pull request* correspondiente.
5. El sistema informa de que el proceso ha finalizado correctamente.
6. El usuario hace click en “OK” para cerrar la pestaña.

Caso de uso UC7: producir diagramas, validar las ontologías y/o generar la documentación de una ontología. Además, aceptar automáticamente la *pull request.*

Actor principal: Usuario.

Personal involucrado e intereses: Los mismos que en UC4

Precondiciones: El usuario está autenticado desde el plugin en GitHub.

Garantía de éxito (postcondiciones): Se genera una *pull request* al repositorio con el nuevo material de soporte, la cual se acepta inmediata y automáticamente.

Escenario principal de éxito:

1. En la lista de repositorios, el usuario selecciona la ontología deseada y marca las casillas correspondientes a las acciones que desee realizar.
2. Hace click en el botón “Go”, situado a la derecha de dichas casillas.
3. Aparece una ventana de confirmación y hace click en “Generate and merge pull request”.
4. El sistema hace una llamada a la API de OnToology para que se realicen las acciones especificadas y se genere la *pull request* correspondiente.
5. El sistema hace una llamada a la API de GitHub para que se acepte la *pull request* recién creada.
6. El sistema informa de que el proceso ha finalizado correctamente.
7. El usuario hace click en “OK” para cerrar la pestaña.

4.3. IMPLEMENTACIÓN

El primer paso para desarrollar un plugin para Protégé es configurar el entorno de desarrollo (en este caso Eclipse). Para ello se han seguido los pasos que indica la documentación de Protégé [11]. Una vez configurado, antes de empezar a escribir código Java, se deben definir diferentes ficheros XML:

- **pom.xml**: En él se definen las diferentes bibliotecas usadas en el proyecto para que éste pueda ser adecuadamente compilado.
- **plugin.xml**: En él se indica el nombre de plugin, así como la ubicación de la clase principal donde se encuentra la implementación del mismo.
- **viewconfig.xml**: En él se describe la apariencia básica de, en este caso, la pestaña OnToology que aparecerá en Protegé al ejecutar el plugin. Esto es, las secciones en las que se divide y la clase en la que se implementa cada una.

Una vez completado este paso sí se puede pasar a escribir código Java como tal. Dada la ausencia de una API para conectarse a OnToology, únicamente se han podido desarrollar funcionalidades completamente independientes de dicho servicio. En cualquier caso, como se ha dicho anteriormente, esta sección no es ni mucho menos el grueso de este TFG.

La interfaz gráfica de usuario se compone de una clase Java Swing por cada componente. Además de lo que está integrado en la pestaña de Protégé, se abrirá una pequeña ventana (clase JPanel[8]) para cada cuadro de confirmación o de información que el sistema muestre al usuario.

En cuanto a la interacción con GitHub, toda ella se realiza por medio de la biblioteca GitHub Java API, en lugar de usar directamente la API REST de GitHub. Así, se tiene una capa de abstracción adicional que hace mucho más fácil la programación. Para detectar repositorios locales se emplea otra biblioteca: JGit.

5. TRABAJOS FUTUROS

La principal línea futura se plantea para cuando la API REST de OnToology esté disponible. Con ella se podrá terminar de construir el plugin. Hay dos opciones para usarla:

[8] JPanel: https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html

- Directamente: Llamando directamente a los verbos HTTP con los parámetros adecuados desde el código Java. Esta opción es la más rápida e inmediata; sin embargo, es también la más difícil de mantener. Esto se debe a que, en caso de un cambio en la API, es muy probable que se deban reescribir todas las llamadas al método afectado. Sin lugar a dudas, esto es muy proclive a provocar errores.

- A través de una biblioteca que envuelva la API: al igual que no se emplea directamente la API oficial de GitHub, sino la GitHub Java API, es posible escribir una biblioteca similar. De esta manera, en el código Java del plugin no aparecerán directamente llamadas a verbos HTTP, sino llamadas a métodos Java que a su vez llamarán a la API de OnToology. Aunque esta aproximación es más costosa en tiempo, merece la pena, pues hace que el código sea más flexible y fácil de mantener. En caso de un cambio en la API, tan solo habría que reescribir los métodos afectados de la biblioteca, quedando el código del plugin intacto. A su vez, esta aproximación cuenta con dos opciones:

    - Escribir la biblioteca dentro del proyecto Java del plugin: la opción más inmediata. Habría un paquete (*package*) donde todas las clases del mismo envolviesen la API. A su vez este paquete podría contener subpaquetes, la organización del código no es algo que se deba decidir ahora.

    - Crear un proyecto aparte: considero que es lo más adecuado por varias razones. En primer lugar, la biblioteca no es una parte del plugin, sino un recurso que el plugin utiliza, por lo que no resulta muy lógico incluirla dentro del mismo proyecto. Por otra parte, si la biblioteca es un proyecto completamente independiente, otros desarrolladores podrían hacer uso de ella. Para ello, es imprescindible documentar adecuadamente cada clase y cada método de la misma. Sería además buena idea publicarla con alguna licencia de código abierto, como la General Public License (GPL) de GNU. De esta manera, podría ser modificada por terceros.

Una vez finalizado el plugin, se podría plantear el desarrollo de proyectos similares para otros editores de ontologías. Sería verdaderamente interesante el poder integrar OnToology dentro de WebProtégé.

6. CONCLUSIONES

Esta ha sido la primera vez que me he enfrentado a un proyecto de estas características. Si bien había usado Protégé para la asignatura Sistemas Inteligentes, jamás había pasado del perfil de usuario, que poco tiene que ver con el perfil de desarrollador.

En un principio este proyecto se planteó para que la escritura de código supusiera la mayor carga de trabajo. Sin embargo, ante la imposibilidad de escribir toda la funcionalidad, se decidió dejar plasmadas las decisiones de diseño de una manera más formal. Por ejemplo, al iniciar el TFG no me planteaba escribir casos de uso, o al menos no de manera formal y; sin embargo, han terminado convirtiéndose en uno de los componente con mayor peso en el mismo.

6.1. A NIVEL PERSONAL

La entrega de este TFG supone para mí el cierre de un ciclo: cinco años desarrollando estudios de Grado en la UPM. Durante este tiempo me he formado como ingeniero informático y puedo decir que las expectativas que tenía cuando empecé la carrera se han cumplido con creces. No ha sido un camino ni mucho menos fácil, pero en ningún momento pensé que lo fuera a ser.

7. BIBLIOGRAFÍA

[1] *Dle.rae.es*, 2017. [Online]. Available: http://dle.rae.es/?id=R5B0YYh.

[2] "ontology - definition of ontology in English | Oxford Dictionaries", *Oxford* Dictionaries | *English*, 2017. [Online]. Available: https://en.oxforddictionaries.com/definition/ontology.

[3] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator and W.R. Swartout, *Enabling Technology for Knowledge Sharing*. AI Magazine. Winter 1991. 36-56.

[4] T. Gruber, *A translation Approach to portable ontology specifications*. Knowledge Acquisition. Vol. 5. 1993. 199-220

[5] B. Swartout; R. Patil; K. Knight; T. Russ, *Toward Distributed Use of Large-Scale Ontologies* Ontological Engineering. AAAI-97 Spring Symposium Series. 1997. 138-148.

[6] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Pérez, A. Fernández and Ó. Corcho, *Supporting Ontology Engineering with OnToology*.

[7]"OnToology", *Ontoology.linkeddata.es*, 2017. [Online]. Available: http://ontology.linkeddata.es/about.

[8]"Protege5DevDocs - Protege Wiki", *Protegewiki.stanford.edu*, 2017. [Online]. Available: https://protegewiki.stanford.edu/wiki/Protege5DevDocs.

[9]"GitHub API v3 | GitHub Developer Guide", Developer.github.com, 2017. [Online]. Available: https://developer.github.com/v3/.

[10]"PluginTypes - Protege Wiki", *Protegewiki.stanford.edu*, 2017. [Online]. Available: https://protegewiki.stanford.edu/wiki/PluginTypes.

[11]"protegeproject/protege", GitHub, 2017. [Online]. Available: https://github.com/protegeproject/protege/wiki/Setup-in-Eclipse.

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Thu Jun 08 16:55:26 CEST 2017
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)