UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería en Tecnologías de la Información

DESARROLLO DE UN SISTEMA DE ETIQUETADO Y CONSULTA SEMÁNTICOS PARA USUARIOS DE LA UNED

MIGUEL EXPÓSITO MARTÍN

Dirigido por: JOSÉ LUIS FERNÁNDEZ VINDEL

Co-dirigido por: RAFAEL MARTÍNEZ TOMÁS

Curso: 2017-2018: 2ª Convocatoria

DESARROLLO DE UN SISTEMA DE ETIQUETADO Y CONSULTA SEMÁNTICOS PARA USUARIOS DE LA UNED

Proyecto de Fin de Grado de modalidad oferta específica

Realizado por: Miguel Expósito Martín

Dirigido por: José Luis Fernández Vindel

Co-dirigido por: Rafael Martínez Tomás

Tribunal calificador

Presidente: D/Dª.

Secretario: D/Dª.

Vocal: D/Dª.

Fecha de lectura y defensa:

Calificación:

Agradecimientos

A mi familia, por soportarme.

Resumen

El presente proyecto tiene como objeto el desarrollo de un sistema de *playground* para SPARQL y RDF que habrá de servir como herramienta educativa de introducción a las tecnologías de la Web Semántica. Su propósito es facilitar una interfaz de uso sencilla para que usuarios profanos puedan introducirse en este tipo de tecnologías a un nivel básico, permitiendo la realización de actividades académicas sobre manejo sencillo de grafos así como sobre consultas SPARQL al conjunto de datos de trabajo o a *endpoints* externos.

Si bien es cierto que la Web Semántica alcanza hoy en día un estado de madurez razonablemente avanzado, el ritmo de cambio de las tecnologías de *Frontend* en *Javascript* (en adelante, JS) es, cuanto menos, vertiginoso. Unido a ello, se tiene que la mayor parte de las implementaciones de componentes de la Web Semántica han sido desarrolladas en tecnologías de *backend* como *Java* (*Apache Jena*) o *Python* (*rdflib*), no existiendo aún estándares o implementaciones maduras puramente en JS.

Se abordan, por tanto, tres retos: la necesidad de conseguir un producto sencillo y fácilmente utilizable por usuarios no expertos, la dificultad para encontrar componentes maduros que aúnen Web Semántica con *frontend* y la conveniencia de apostar por un *framework* de desarrollo en JS estable y con una curva de aprendizaje suave.

Para dar solución a estos problemas, se ha optado por desarrollar una *Single Page Application* (SPA) con Vue.js (un *framework* JS que se jacta de aglutinar las mejores características de Angular y React, sus principales competidores) e integrarlo con las implementaciones recomendadas por el grupo de trabajo de bibliotecas JS *rdfjs* y con una plataforma de consultas para la web flexible y modular. El sistema está planteado para ejecutarse en un sencillo navegador contemporáneo, descargándose a través de un simple servidor web (siendo este la única infraestructura necesaria para su distribución).

Dado el alto nivel de incertidumbre existente en la implementación de las necesidades del proyecto, se ha optado por utilizar un enfoque metodológico incremental e iterativo basado en Extreme Programming y apoyado sobre tableros Kanban, lo que ha permitido una mejor organización y evolución del proyecto.

El resultado final ofrece un producto básico de introducción a las tecnologías de la Web Semántica y permite pensar en un desarrollo del mismo tan ambicioso como las propias necesidades de los equipos docentes, dado el estado de madurez actual del frontend web.

Abstract

<This project ...>

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

No cabe duda que uno de los retos a los que se enfrenta la Sociedad de la Información y del Conocimiento es la educación, una de las mejores inversiones en el futuro de Europa. Según la última Comunicación de la Comisión al Parlamento, al Consejo, al Comité Económico y Social Europeo y al Comité de las Regiones[2], el uso de las TIC para propósitos educativos se queda atrás, tanto en adopción como en competencias del profesorado. La innovación en sistemas educativos, entendida como la adopción de nuevos servicios y herramientas por parte de las organizaciones educativas, puede mejorar los resultados formativos y la eficiencia.

Siendo una de las prioridades del Plan de Acción propuesto por la Comisión hacer mejor uso de la tecnología digital para el aprendizaje y la enseñanza, parece apropiado que instituciones como la Universidad Nacional de Educación a Distancia (UNED) en España y sus departamentos, como el de Inteligencia Artificial, se propongan aplicar los últimos avances tecnológicos para mejorar la calidad del sistema educativo y facilitar a los alumnos la adquisición de nuevas competencias digitales complejas.

Dentro de estas competencias se pueden encuadrar términos como Web de Datos, Web Semántica o Datos Enlazados, todos definidos por el padre de la Web, Tim Berners-Lee[3, 4]. No son más que la evolución lógica de la Web a la que todos estamos acostumbrados, pero entendida en su crecimiento como una solución a los problemas de la ingente cantidad de datos no estructurados o inconexos que se genera al día en 2018 (según Domo[5], 2,5 quintillones de bytes).

En concreto, este trabajo pretende acercar las tecnologías de la Web Semántica al alumnado siguiendo las recomendaciones del W3C RDFJS Community Group[6] a través de una sencilla aplicación puramente desarrollada en Javascript y lista para usar, que permita la exploración de facilidades de modelado y consulta sobre un conjunto de datos predefinido o aportado por el propio alumno así como la carga y realización de actividades simples propuestas por el profesorado.

La memoria de este proyecto se estructura en los siguientes capítulos:

- **Capítulo 1. Introducción.** Esta introducción.
- **Capítulo 2. La Web Semántica.**
- **Capítulo 3. Motivación y objetivos.**
- **Capítulo 4. Trabajos previos, estado actual y recursos.**
- **Capítulo 5. Metodología y planificación.**
- **Capítulo 6. Análisis.**
- **Capítulo 7. Diseño.**
- **Capítulo 8. Implementación y pruebas.**
- **Capítulo 9. Resultados.**
- **Capítulo 10. Conclusiones y líneas futuras de trabajo.**

Capítulo 2

La Web Semántica

La Web Semántica es un término originalmente acuñado por Tim Berners-Lee en el año 2001[3]. Hasta la fecha, la *World Wide Web* había sido concebida como una idea de colaboración abierta en la que múltiples contribuciones de varios autores podían tener cabida y ser compartidas universalmente. Dichas contribuciones, realizadas en forma de documentos, estaban dirigidas a personas y no a máquinas o computadores. Berners-Lee vio más allá y propuso su extensión para lograr su manipulación automática; en resumidas cuentas, permitir que agentes inteligentes (programas de ordenador) fueran capaces de encontrar datos y su significado a través de hiperenlaces a definiciones de términos clave y reglas de razonamiento e inferencia lógica.

Para lograr tan ambicioso objetivo, los agentes inteligentes necesitarían tener acceso a contenido y conocimiento estructurado, así como a las reglas de inferencia necesarias. En este contexto, la Web Semántica podía apoyarse en las siguientes tecnologías existentes:

- **RDF** (*Resource Description Framework*), que permite expresar conocimiento en forma de tripletas (a modo de sujeto, verbo y objeto en una oración) utilizando URIs (*Uniform Resource Indicators*).
- **XML** (*eXtensible Markup Language*), que permite la creación de documentos convenientemente etiquetados y con estructura arbitraria.

Sin embargo, un tercer pilar era necesario para resolver la problemática de la existencia de distintos identificadores en distintas bases de datos relacionadas con el mismo significado conceptual. Gracias a las **ontologías**, documentos que definen formalmente las relaciones entre distintos términos, estas diferencias podían ser salvadas bien mediante el uso de términos estandarizados bien mediante la definición de relaciones conceptuales de igualdad o similitud entre dichos términos.

En palabras del propio Berners-Lee: «Con un diseño adecuado, la Web Semántica puede ayudar

en la evolución del conocimiento humano como un todo.» ([3])

2.1. Linked Data

Estrechamente relacionado con la Web Semántica se encuentra el concepto de datos enlazados o *Linked Data*, también propuesto por Berners-Lee en 2006[4] en lo que se considera como su introducción oficial y formal. Se trata de un movimiento respaldado por el propio W3C que se centra en conectar conjuntos de datos a lo largo de la Web y que puede verse como un subconjunto de la Web Semántica.

Se basa, por tanto, en dos aspectos clave:

- La publicación de conjuntos datos estructurados en línea.
- El establecimiento de enlaces entre dichos conjuntos de datos.

La Web Semántica es el fin y los datos enlazados proporcionan el medio para alcanzar dicho fin.

2.2. RDF

Las tecnologías de la Web Semántica permiten, por tanto, almacenar conocimiento en forma de conceptos y relaciones a través de ternas o tripletas, modelar dominios de conocimiento con vocabularios estándar y ofrecer potentes facilidades de consulta sobre dichos modelos. Dentro de estas tecnologías, RDF es, sin duda, el bloque de construcción fundamental en la Web Semántica (como HTML lo ha sido para la Web).

RDF fue propuesto por el W3C en 1999[7] como un estándar para crear y procesar metadatos con el objetivo de describir recursos independientemente de su dominio de aplicación, ayudando por tanto a promover la interoperabilidad entre aplicaciones.

Sin embargo, con la llegada de la Web Semántica el alcance de RDF creció, y ya no sólo se usa para codificar metadatos sobre recursos web, sino también **para describir cualquier recurso así como sus relaciones** en el mundo real.

El nuevo alcance de RDF quedó reflejado en las especificaciones publicadas en 2004 por el *RDF Core Working Group*[1], ahora actualizadas a fecha de 2014[2], de las que se pueden extraer las siguientes definiciones de RDF:

Recomendación	Definición
RDF Primer	Un framework para expresar información sobre recursos.
RDF Concepts & Syntax	Un framework para representar información en la Web.

Cuadro 2.1: Definiciones de RDF en las recomendaciones del W3C

2.2.1. Modelo Abstracto de RDF

RDF ofrece un modelo abstracto que permite descomponer el conocimiento en pequeñas piezas llamadas sentencias (*statements*), tripletas o ternas y que toman la forma:

Sujeto - Predicado - Objeto

En donde *Sujeto* y *Objeto* representan dos conceptos o "cosas" en el mundo (también denominados **recursos**) y *Predicado* la relación que los conecta.

Los nombres de estos recursos (así como de los predicados) han de ser globales y deberían identificarse por un *Uniform Resource Idenfifier* (URI).

Por tanto, un modelo RDF puede expresarse como una colección de tripletas o un grafo, dado que estas no son más que grafos dirigidos. Los sujetos y objetos serían los nodos del grafo y los predicados sus aristas.

Otra nomenclatura utilizada para representar una terna sería la siguiente:

[1] Recomendaciones del W3C, https://www.w3.org/2001/sw/RDFCore/

[2] Versiones actualizadas de las recomendaciones del W3C, https://www.w3.org/2011/rdf-wg/wiki/Main_Page

Recurso - Propiedad - Valor de la propiedad

En donde el valor de la propiedad u Objeto también puede ser un literal, que consiste simplemente en un dato textual bruto.

Cabe destacar que una terna RDF sólo puede modelar relaciones binarias. Para modelar una relación n-aria, suelen utilizarse recursos intermedios como nodos en blanco, que no son más que sujetos u objetos que no tienen un URI como identificador (nodos sin nombre o anónimos).

En resumen:

Sujeto	Puede ser un URI o un nodo en blanco.
Predicado o Propiedad	Debe ser un URI.
Objeto o Valor de la Propiedad	Puede ser un URI, un literal o un nodo en blanco.

Cuadro 2.2: Resumen conceptual de ternas

A continuación se muestran dos ejemplos de tripletas:

Sujeto	Predicado	Objeto
http://www.uned.es/ia/example#Analista	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
http://www.uned.es/ia/example#Analista	http://www.w3.org/2000/01/rdf-schema#label	"Analista Informático"

Cuadro 2.3: Ejemplos de tripletas reales

2.2.2. Aplicaciones prácticas de RDF

Gracias a RDF, es posible construir la Web Semántica y enlazar conjuntos de datos. A continuación se enumeran algunas de sus aplicaciones reales:

- Añadir información legible a los motores de búsqueda.
- Enriquecer un conjunto de datos enlazándolo con conjuntos de datos de terceros.
- Facilitar el descubrimiento de APIs[3] a través de su entrelazado.
- Construir agregaciones de datos sobre determinados temas.
- Proporcionar un estándar de intercambio de datos entre bases de datos.
- Enriquecer, describir y contextualizar los datos mediante un enfoque rico en metadatos.

Su principal ventaja frente a los modelos y bases de datos relacionales es una mayor facilidad para evolucionar los grafos, algo que en un sistema relacional es complejo (dado que requiere de modificaciones en las estructuras de almacenamiento de datos y sus consultas asociadas). En otras palabras, su **flexibilidad para modelar lo inesperado**.

Un ejemplo claro a nivel global es Wikidata[4], una base de datos de conocimiento abierta que puede ser leída y editada tanto por personas como por máquinas. Wikidata actúa como un almacén centralizado para datos estructurados de otros proyectos hermanos como Wikipedia, Wikisource, etc. En el momento de redacción de esta memoria, cuenta con 50,035,140 elementos de datos que cualquiera puede editar.

De una forma más concreta, es destacable el importante rol de RDF y las tecnologías de la Web Semántica en ámbitos de **búsqueda y clasificación bibliográfica o documental**, pudiendo citarse como ejemplo la publicación en RDF del *Diccionario de Lugares Geográficos* y el *Diccionario y Tesauro de Materias* del Patrimonio Cultural de España, gestionado en la actualidad por el Ministerio de Educación, Cultura y Deporte del Gobierno de España[5].

2.3. SPARQL

Tal como SQL provee (al menos, relativamente) un de un estándar de consulta para las bases de datos relacionales, SPARQL ofrece un lenguaje de consulta estandarizado para grafos RDF.

[3] *Aplication Programming Interface*, en este caso referido a puntos accesibles a través de la Web.
[4] https://www.wikidata.org/wiki/Wikidata:Main_Page
[5] https://www.mecd.gob.es/cultura-mecd/areas-cultura/museos/destacados/2015/tesauros.html

SPARQL es un acrónimo recursivo para SPARQL Protocol And RDF Query Language; es a la vez un lenguaje de consulta de grafos RDF y un protocolo, dado que permite exponer servicios web que acepten consultas en dicho lenguaje.

Entre sus características, cabe citar:

- Recuperación de valores de datos estructurados o semi-estructurados.
- Exploración de conjuntos de datos a través de consultas sobre relaciones desconocidas.
- Ejecución de uniones complejas sobre bases de datos dispersas en una única consulta.
- Transformación de datos RDF desde un vocabulario a otro.

La estructura básica de una consulta SPARQL es la siguiente:

```
# declaraciones de prefijos
PREFIX foo: <http://example.com/resources/>
...
# definición del conjunto de datos
FROM ...
# cláusula de resultado
SELECT ...
# patrón de la consulta
WHERE {
        ...
}
# modificadores de la consulta
ORDER BY ...
```

Listado de código 2.1: Estructura básica de consulta SPARQL.

No es objeto de este proyecto profundizar en la sintaxis del lenguaje, pero sí se incluye a continuación un ejemplo de consulta:

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
        ?person foaf:name ?name .
}
```

Listado de código 2.2: Consulta que devuelve todos los nombres de personas mencionadas en un grafo.

Capítulo 3

Motivación y objetivos

3.1. Motivación

En el ámbito académico, la Web Semántica se contextualiza e integra en materias como la Gestión Avanzada de la Información y el conocimiento, donde las Tecnologías de la Información se insertan con la Inteligencia Artificial con el objeto de modelar conocimiento humano para un posterior procesamiento y aprendizaje automático.

Las tecnologías de la Web Semántica se caracterizan por un cambio de paradigma bastante brusco con respecto a otras tecnologías de manipulación de datos con las que el alumno muy probablemente pueda estar familiarizado a la hora de introducirse en este nuevo mundo. Es cierto que pueden establecerse ciertas comparativas conceptuales para facilitar su comprensión; como ejemplo, el cuadro 3.1 equipara la Web Semántica con un sistema basado en bases de datos relacionales.

Mundo Relacional	Mundo Semántico
Registro de tabla	Nodo RDF
Columna de tabla	RDF propertyType
Celda de tabla	Valor
Consulta SQL	Consulta SPARQL
Modelo de datosL	Ontología

Cuadro 3.1: Comparativa paradigma relacional vs semántico

Sin embargo, determinados indicadores pueden dar lugar a entender que la interiorización de los conceptos asociados a la Web Semántica no es sencilla. Por ejemplo: su uso casi exclusivamente académico con poca penetración en el mundo empresarial, su lenta velocidad de propagación, su soporte limitado en determinadas plataformas tecnológicas o incluso su tímido interés en el ámbito público (excluyendo honrosas excepciones como el académico o universitario, las iniciativas de datos abiertos o lo relativo a la biblioteconomía). Un ejemplo muy concreto, real y tangible de esta escasa penetración es el reducido número de artículos (cuatro) presentado en *workshops* como SemStats[1][8], de cuyo *Program Comittee* este autor forma parte:

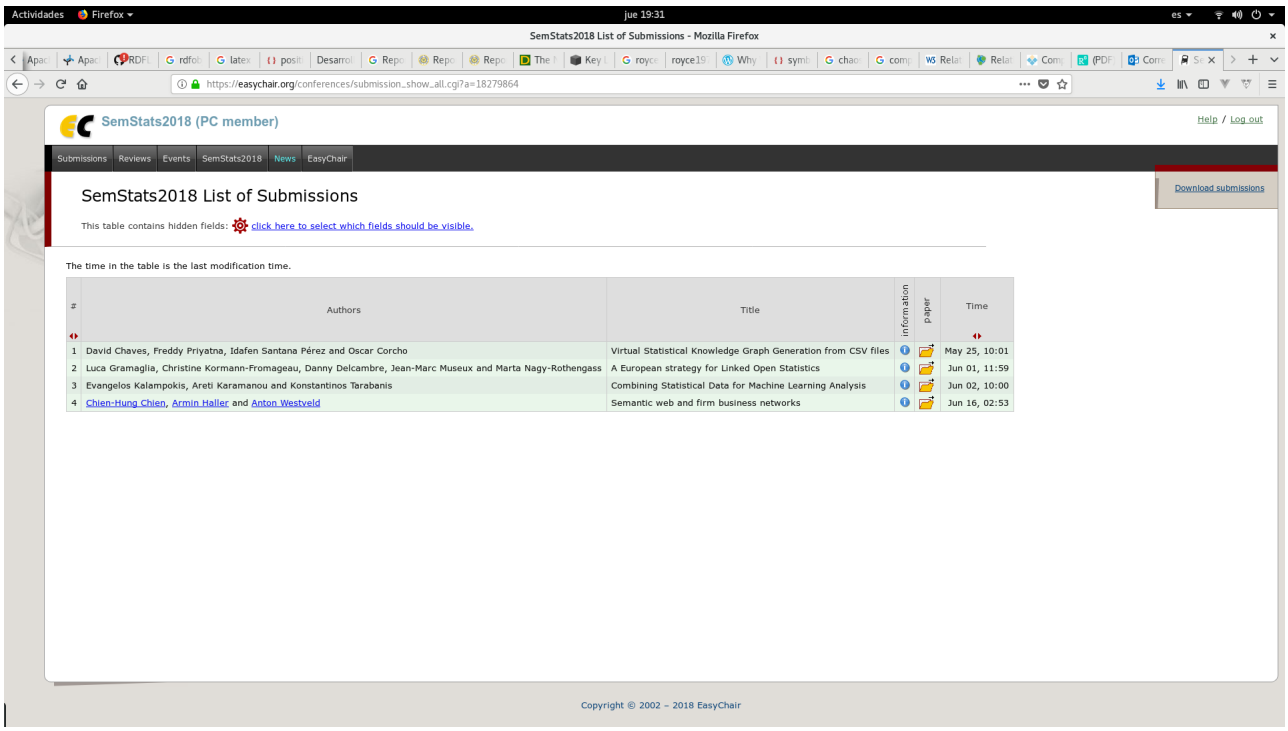

Figura 3.1: Trabajos enviados a Semstats 2018

En este ejemplo concreto, resulta cuando menos sorprendente que las Oficinas Estadísticas Públicas, encargadas de generar y publicar datos estadísticos sobre economía, población y sociedad en general, no estén mostrando mayor interés a la hora no ya de publicar sus datos en formatos enlazables, sino de construir aplicaciones semánticas para explotar y relacionar mejor estos datos.

Es esta **situación de dificultad en el aprendizaje y en la interiorización de la utilidad de las tecnologías de la Web Semántica la que motiva**, en última instancia, la idea de desarrollar herramientas que acerquen este tipo de tecnologías al público en general y faciliten una toma de contacto gradual con las mismas, ayudando así a su divulgación.

[1]SemStats es un *workshop* de celebración anual centrado en el estado actual de las tecnologías de la Web Semántica aplicadas a la estadística pública

3.2. Objetivos

Si bien es cierto que existen, por una parte, varias aproximaciones gratuitas en la red a modo de *playground* que permiten el lanzamiento de consultas SPARQL a determinados *endpoints* y por otra, editores y herramientas de modelado del calibre de Protégé[2] (analizados con detalle en la sección 4.1), no parece haber en el mercado referencias de aplicaciones que ofrezcan simultáneamente subconjuntos reducidos de ambas funcionalidades. Llenar este hueco y **conseguir una herramienta formativa única de modelado y consulta en tecnologías de la Web Semántica** que permita a los Equipos Docentes de las asignaturas relacionadas con tecnologías de la Web Semántica ofrecer a los alumnos (o a cualquier no iniciado en estas tecnologías) un *playground* o entorno de pruebas que les ayude a consolidar su aprendizaje teórico con una base práctica adaptada al mundo real **es el objetivo principal de este proyecto**.

De forma más específica, es posible enumerar los siguientes objetivos detallados:

- Facilitar el aprendizaje y la **familiarización con tecnologías de la Web Semántica** a usuarios inexpertos.
- Reducir al máximo los requisitos de uso de la herramienta para alcanzar el mayor público objetivo posible.
- Ofrecer a los Equipos Docentes la posibilidad de **distribuir actividades formativas** autocontenidas para que los alumnos trabajen sobre ellas en la herramienta.
- Agrupar en un solo producto **funcionalidades básicas de edición y consulta sobre grafos RDF**, de cara a cubrir las expectativas educativas en este ámbito.
- Permitir a los alumnos comprobar *in situ* los efectos de sus acciones sobre un grafo RDF y ofrecerles la **flexibilidad suficiente** como para permitirles dar rienda suelta a su creatividad e inquietudes.

  Y por último, y no menos importante,

- **Obtener un MVP** (*Minimum Viable Product*)[3] que pueda ser evolucionado y poner su código a disposición de la comunidad educativa para revertir sobre ella su valor aportado.

[2] https://protege.stanford.edu/

[3] Un producto con las suficientes características como para satisfacer a sus usuarios tempranos.

Capítulo 4

Trabajos previos, estado actual y recursos

4.1. Trabajos previos

Para enfocar el presente proyecto se llevó a cabo un estudio previo de los trabajos existentes que pudieran estar relacionado con los objetivos del mismo. En las siguientes subsecciones se analizan las familias de herramientas analizadas.

4.1.1. Herramientas de consulta

La herramienta de consulta web que se asemeja más a los resultados obtenidos con este proyecto es el SPARQL playground desarrollado por el *Swiss Institute of Bioinformatics*[1].

Se trata de una aplicación web *standalone* y multiplataforma cuyo propósito fundamental es el aprendizaje de SPARQL. Sus autores proporcionan una demo *online*[2] además de documentación y consultas de ejemplo para que cualquiera pueda familiarizarse con este lenguaje de consulta.

Si bien sus funcionalidades de consulta son muy completas, carece de facilidades de edición de grafos propios o de consulta a *endpoints* externos, por ejemplo.

Existan también bibliotecas de consulta para el *frontend*, como Jassa (*JAvascript Suite for Sparql Access*)[3]. Se trata de una biblioteca modular que comprende desde una API RDF sobre una capa de abstracción de servicios hasta una capa de navegación por facetas. Sin embargo, desarrolla un

[1] https://www.isb-sib.ch
[2] http://sparql-playground.sib.swiss/
[3] http://aksw.org/Projects/Jassa.html

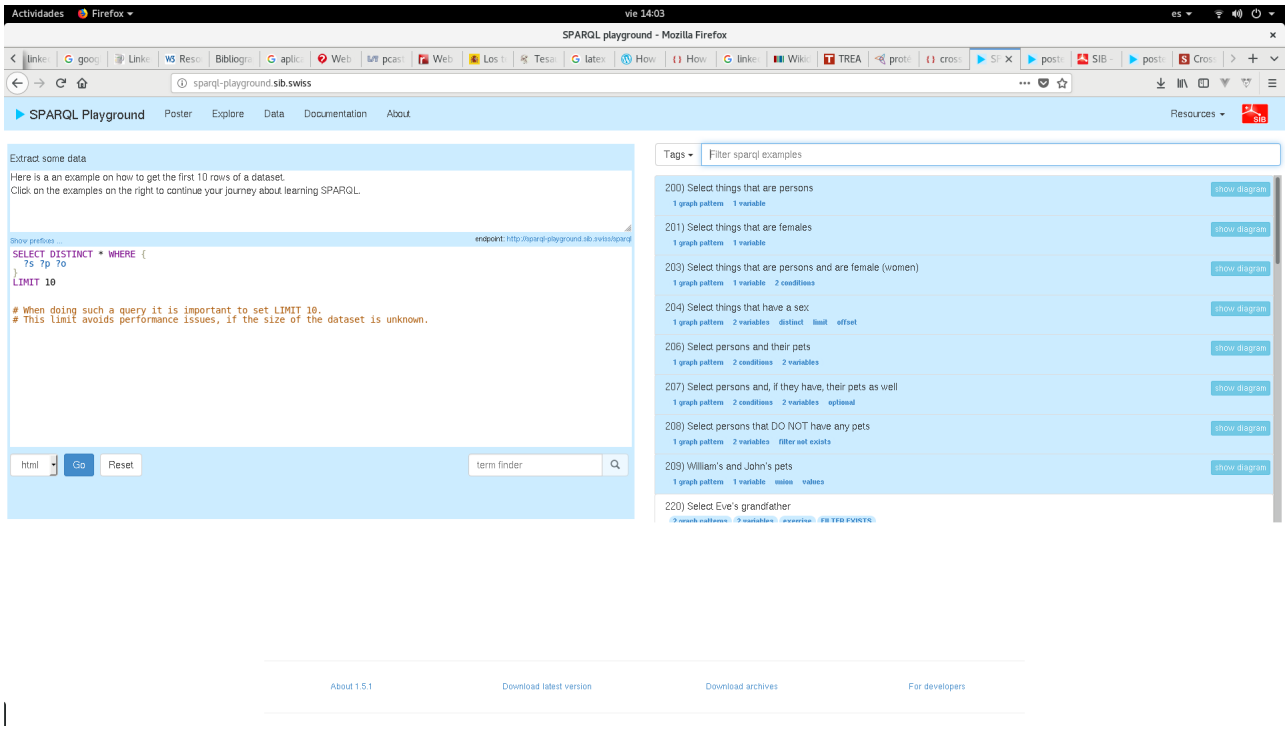

Figura 4.1: Interfaz de SPARQL *playground*

modelo propio del grafo RDF no basado en estándares y se integra con versiones muy antiguas de Angular, lo que hace que pierda interés para su uso en producción.

4.1.2. Herramientas de modelado

4.1.2.1. Herramientas web

Existen pocas herramientas con funcionalidades de edición de grafos RDF en la Web. Al margen de la versión web de Protégé, cuya versión de escritorio (más representativa) se comenta en el punto 4.1.2.2, el resto de trabajos encontrados son fundamentalmente bibliotecas o paquetes de edición de formularios, como rdfforms[4]. Se trata de una biblioteca Javascript cuyo propósito es facilitar la construcción de editores RDF basados en formularios en un entorno web. Para ello, se basa en un mecanismo de plantillas que describe cómo generar un formulario HTML y cómo mapear expresiones específicas en un grafo RDF a sus correspondientes campos.

Unas pruebas iniciales con la biblioteca sugirieron falta de madurez por fallos en su funcionamiento

[4]https://rdforms.org/

en un navegador. Además, la biblioteca implementa su propio modelo de grafo RDF, que no está relacionado con los estándares actuales (ver sección 4.2).

Figura 4.2: Ejemplo de formulario desarrollado con rdfforms

4.1.2.2. Herramientas de escritorio

La herramienta de modelado de ontologías en RDF por antonomasia es Protégé, un editor de ontologías *opensource* gratuito y un marco de trabajo para construir sistemas inteligentes soportados por una fuerte comunidad de usuarios académicos, gubernamentales y corporativos. Se utiliza en áreas tan diversas como la biomedicina, el comercio electrónico, la predicción metereológica o el modelado organizacional.

Protégé se instala como una aplicación de escritorio multiplataforma y tiene una arquitectura basada en *plugins* o complementos que permite extender fácilmente su funcionalidad. Tiene soporte para razonadores sobre las distintas ontologías que se modelan y facilidades de representación y edición de grafos RDF.

De cara a un primer encuentro con las tecnologías de la Web Semántica, puede parecer abrumador por su gran flexibilidad y potencia. Además, no cuenta con facilidades de consulta sobre los grafos modelados.

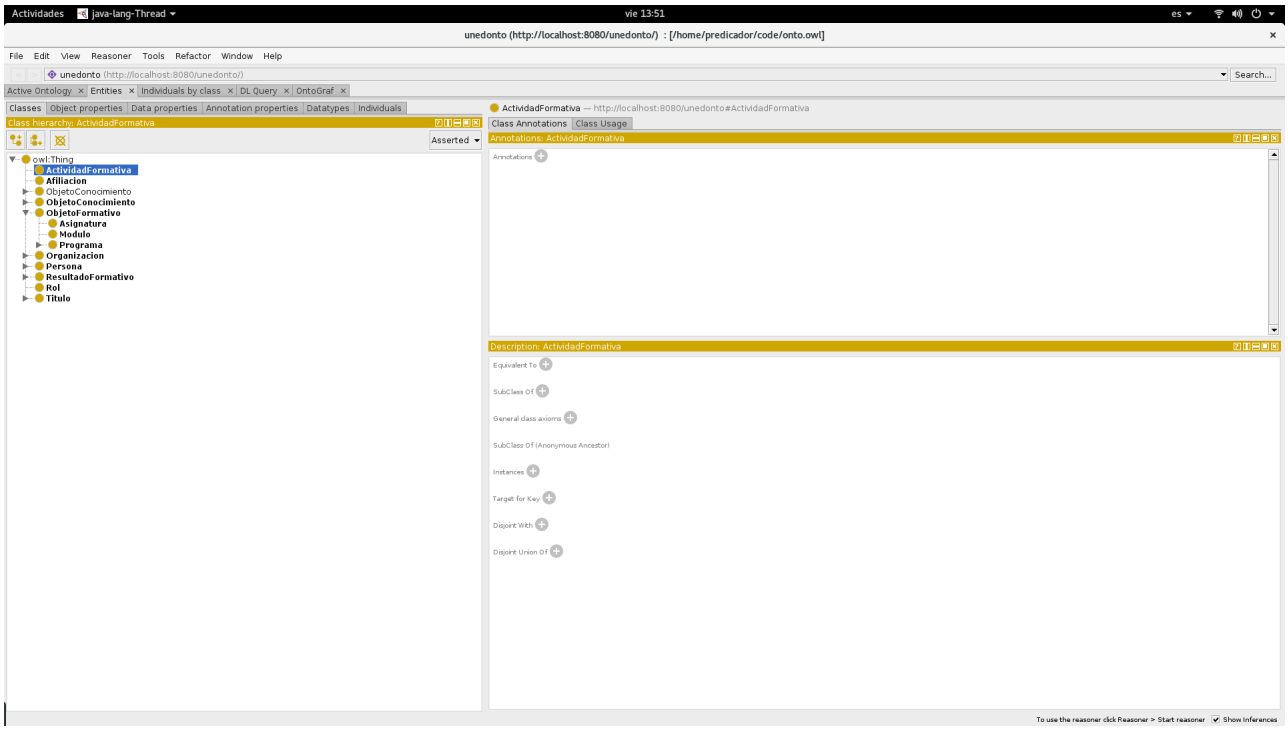

Figura 4.3: Interfaz de Protégé

4.2. Estado actual

El estado actual (o *state of the art*, del inglés) de las tecnologías de la Web Semántica alcanza distintos niveles de madurez en sus implementaciones. Complementando el listado de herramientas ya presentado en el estudio de trabajos previos (4.1), se analizan a continuación las plataformas o bibliotecas más representativas dentro de las tecnologías más utilizadas.

4.2.1. RDF en Java

Así, para la plataforma Java se cuenta con Apache Jena[9], un marco de trabajo o *framework opensource* para construir aplicaciones para la Web Semántica o sobre datos enlazados. De entre sus componentes y funcionalidades, cabe destacar:

- **TDB**: una base de datos de tripletas nativa y de alto rendimiento, con excelente integración con el resto de APIs de Jena.
- **ARQ**: un motor SPARQL compatible con su versión 1.1 que soporta consultas federadas y búsqueda por texto libre.

- **API RDF**: una API principal que permite interactuar con RDF para crear y leer grafos y tripletas, así como serializarlas a los formatos más comunes (Turtle, XML, etc.).
- **Fuseki**: un *endpoint* SPARQL que permite exponer el grafo de trabajo y ofrece interacción tipo REST con tripletas RDF.
- **Otras APIs**: como las de ontologías e inferencias, que permiten añadir más semántica al modelo y razonar sobre reglas por defecto o personalizadas.

Apache Jena es una solución robusta y muy utilizada en entornos académicos y de producción empresarial con más de quince años de vida.

4.2.2. RDF en Python

RDFLib[5] es una biblioteca *opensource* ligera pero funcionalmente completa para trabajar con RDF desde plataformas Python. Permite a las aplicaciones acceder a estructuras RDF a través de construcciones idiomáticas en Python, lo que facilita un acercamiento de la tecnología al programador experimentado; por ejemplo, un grafo no es más que una colección de tripletas *<sujeto, predicado, objeto>*. Entre el resto de sus características, destacan:

- Contiene procesadores y serializadores para XML, N3, Turtle, RDFa, etc.
- Presenta una interfaz para un grafo que puede soportarse sobre multitud de implementaciones de almacenes.
- Incluye una implementación de SPARQL v1.1.
- Presenta una arquitectura modular basada en *plugins* o complementos.

Su desarrollo es estable y ha sido utilizada en referencias bibliográficas como *Programming the Semantic Web*[10], de *O'Reilly*.

[5] https://github.com/RDFLib/rdflib

4.2.3. RDF en Javascript

Dado que uno de los objetivos del proyecto es reducir al máximo los requisitos de uso de la herramienta para alcanzar el mayor público objetivo posible (ver 3.2), **Javascript se presenta como una tecnología de sumo interés para la implementación del mismo**, ya que permite producir aplicaciones ejecutables al cien por cien en un navegador web estándar.

Sin embargo, el **estado actual de implementación de RDF en Javascript puede calificarse de inmaduro y e inestable**. Se han llevado a cabo intentos aislados de construir APIs RDF y herramientas para gestión de grafos y consultas SPARQL, como las citadas en la sección 4.1, pero no existe un estándar único consolidado (aún).

4.2.3.1. Manejo de RDF

En el W3C puede consultarse la mejor comparativa existente de bibliotecas desarrolladas en Javascript para trabajar con RDF[11].

Sin duda, el esfuerzo más destacable hasta ahora para conseguir un estándar de RDF en Javascript es el proyecto **rdflib.js**[6], iniciado por el mismísimo Tim Berners-Lee. Se trata de una biblioteca que proporciona una API local para lanzar consultas a un almacén RDF, soporta parte de la especificación SPARQL y permite importar y escribir formatos como RDF/XML, Turtle y N3.

En el propio repositorio de rdflib.js se cita al *RDFJS Representation Task Force*[7], un grupo dentro del *RDFJS Community Group*[8] encargado de diseñar las especificaciones de interfaz con el objetivo de conseguir que las distintas implementaciones de conceptos RDF en Javascript sean interoperables. La especificación de la interfaz existe en forma de borrador a fecha de 14 de Agosto de 2017[9]. En ella se definen interfaces para términos, cuaternas, factorías de datos, nodos en blanco, literales, etc. así como para *streams* de Javascript.

Consultando el sitio web del *Community Group* se puede encontrar una noticia del 23 de Abril de 2018 (publicada, por tanto, en pleno desarrollo de este proyecto) anunciando la creación de RDF.js, que tratará de unir los esfuerzos de distintas entidades (como la Universidad de Gante o el MIT) y

[6] https://github.com/linkeddata/rdflib.js
[7] https://github.com/rdfjs/representation-task-force
[8] https://www.w3.org/community/rdfjs/
[9] http://rdf.js.org/

alinear y publicar bajo un único paraguas proyectos como:

- El citado rdflib.js
- **RDF-Ext**[10], una biblioteca Javascript para trabajar con RDF y datos enlazados que cumple con la especificación del *RDFJS Representation Task Force*.
- **N3.js**[11], una biblioteca que implementa la especificación de bajo nivel RDF.js y permite leer, procesar, almacenar y exportar ternas RDF de forma asíncrona en Javascript.

Ante esta situación confusa y repleta de distintas implementaciones, se optó por utilizar canales de comunicación directos con sus autores. Existen varios grupos de discusión en Gitter, tales como:

- rdfjs/public[12]
- rdf-ext/discussions[13]

En ellos, el autor del proyecto tuvo la oportunidad de recibir indicaciones directamente de Thomas Bergwinkl y Michael Luggen, importantes miembros del grupo, quienes confirmaron que RDF-Ext es el camino a seguir en caso de tener que desarrollar un proyecto en Javascript que trabaje con RDF. Luggen también indicó que rdflib.js es la biblioteca más antigua de las citadas, iniciada por el propio Tim Berners-Lee (que, como se puede comprobar en la figura 4.4, aparece también en el grupo de conversación.)

4.2.3.2. Manejo de SPARQL

Como se ha visto en la subsección anterior, la situación actual de la implementación de RDF en Javascript es compleja. Con SPARQL, esta línea se mantiene.

El primer trabajo desarrollado en esta línea es **rdfstore-js**, de Antonio Garrote[14], que consiste en una implementación pura en Javascript de un almacén de grafos RDF con soporte para el lenguaje de consulta y manipulación de datos SPARQL.

[10] https://github.com/rdf-ext/rdf-ext
[11] https://github.com/rdfjs/N3.js
[12] https://gitter.im/rdfjs/public
[13] https://gitter.im/rdf-ext/discussions
[14] https://github.com/antoniogarrote/rdfstore-js

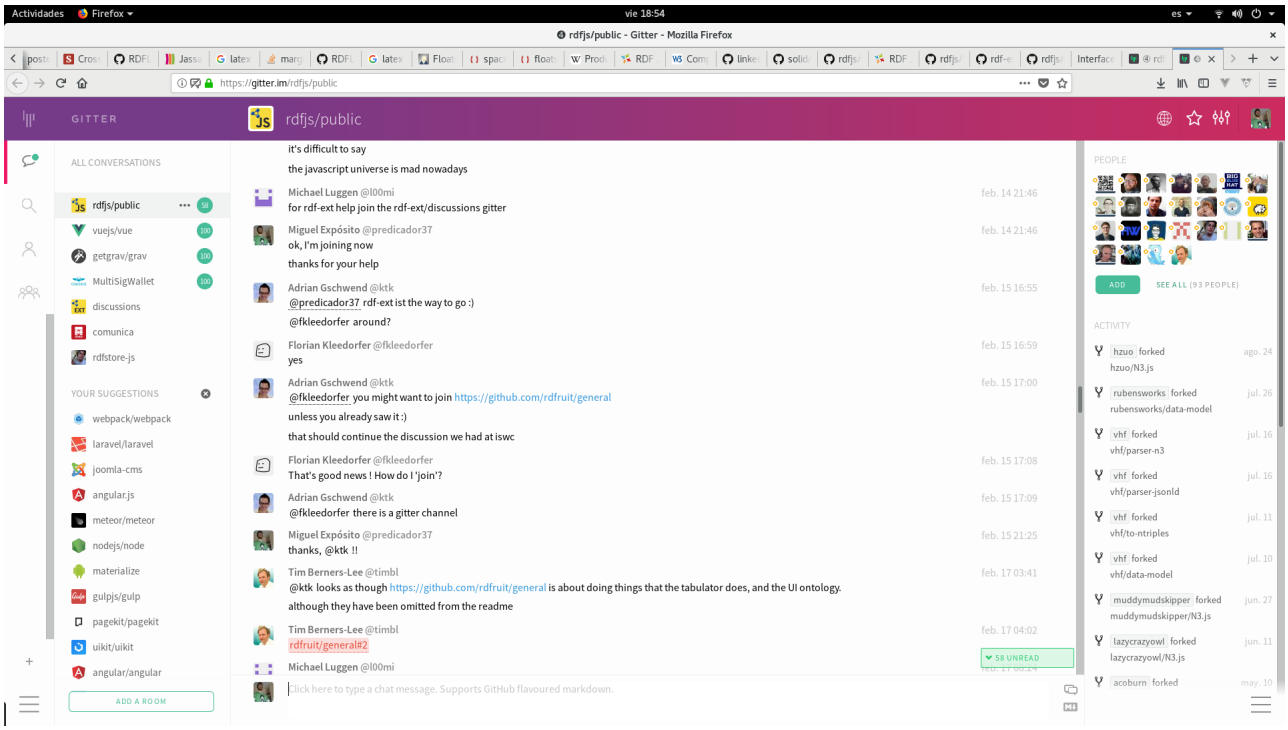

Figura 4.4: Charla con miembros del W3C RDFJS Community Group

Lamentablemente, tal y como indica el propio autor, la implementación actual está lejos de estar completa. Por otra parte, revisando el repositorio, se puede comprobar que la última actualización del código fuente es de hace dos años. Existe un grupo de gitter sobre el proyecto, pero carente de actividad. Por tanto, no es descabellado asegurar que la biblioteca está obsoleta y ha sido discontinuada, lo que hace inviable su inclusión en el proyecto.

Por otro lado, Ruben Taelman, miembro del grupo de discusión en Gitter sobre RDF-Ext, lidera otro proyecto, Comunica[15]: una plataforma de motores de consulta altamente modular y flexible para la web. Esta plataforma permite enviar consultas a distintos motores, entre ellos SPARQL. Entre sus características se encuentra la capacidad para lanzar consultas SPARQL 1.1 a endpoints externos y también a un almacén local en Javascript, lo que lo hace ideal para un proyecto como el que se trata en esta memoria.

No obstante, es un producto en pleno desarrollo en el que muchos de sus módulos aún no han sido publicados en una versión estable y, por tanto, **inestable y con defectos potenciales**.

[15]https://github.com/comunica/comunica

4.2.4. Tecnologías de frontend web

La interfaz de usuario será una parte muy importante del proyecto, que pretende facilitar el aprendizaje de RDF y SPARQL a alumnos sin experiencia. Ello, unido a la necesidad de facilitar lo máximo posible la puesta en marcha de la herramienta, hace que las tecnologías de *frontend* web cobren un papel especialmente relevante en este análisis.

El mundo del *frontend* web ha cambiado mucho en los últimos años, y de hecho, la comunidad de desarrolladores aún se encuentra inmersa en una vorágine de cambios cuya velocidad es, en la mayor parte de los casos, infinitamente mayor que la capacidad de los proyectos y los propios desarrolladores para asimilarla.

Atrás quedaron los días de servir interfaces de usuario en el servidor a través de JSP[16] o JSF[17] y de esas tímidas incursiones de Javascript en la capa de presentación a través de bibliotecas como JQuery[18] o Ext Js[19].

En estos momentos, son tres los frameworks que aglutinan la mayor parte del desarrollo de frontend web:

- **Angular**
- **React**
- **Vue**

En las siguientes subsecciones se analizan someramente.

4.2.4.1. Angular

Angular es un framework basado en Typescript y creado por Google en 2009, pensado para la construcción de aplicaciones web altamente interactivas. Entre sus características se encuentran las siguientes:

[16] Java Server Pages, tecnología para crear páginas web dinámicas en servidor en Java
[17] Java Server Faces, tecnología que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE
[18] https://jquery.com/
[19] https://www.sencha.com/products/extjs/#overview

- Orientado a componentes[20].
- Uso de Typescript.
- Sistema de plantillas en HTML con directivas.
- Basado en el patrón MVC[21].
- Incluye inyección de dependencias[22].
- Varias opciones para la gestión de estado.
- Curva de aprendizaje pronunciada.
- Tamaño pesado.
- Publicado bajo licencia MIT.

4.2.4.2. React

React es una biblioteca Javascript para construir interfaces de usuario desarrollada y mantenida por Facebook desde 2013.

- Orientado a componentes.
- Uso de Javascript ES6.
- Sistema de plantillas en JSX.
- Basado en el patrón MVC.
- Virtual DOM: permite organizar documentos HTML en un árbol más sencillo de procesar por un navegador web.
- Gestión de estado con Redux.
- Curva de aprendizaje pronunciada.
- Tamaño medio.
- Publicado bajo licencia MIT.

[20]Un componente es una unidad de encapsulación de código y responsabilidades que facilita la reutilización
[21]Modelo, Vista, Controlador.
[22]Patrón en el cual un objeto (servicio) proporciona las dependencias necesarias a otro objeto (cliente)

4.2.4.3. Vue

Vue es uno de los frameworks que más rápidamente está creciendo en popularidad y uso desde 2014. Su primera versión fue publicada por un antiguo trabajador de Google, Evan You.

- Orientado a componentes.
- Uso de Javascript ES5/ES6.
- Sistema de plantillas en HTML con directivas.
- *Single File Components*, con separación de responsabilidades.
- Basado en el patrón MVVM[23].
- Virtual DOM.
- Gestión de estado con Vuex.
- Curva de aprendizaje suave.
- Tamaño ligero.
- Publicado bajo licencia MIT.

4.2.4.4. Comparativa de frameworks

Tras un análisis de las características de los tres frameworks, se puede comprobar que técnicamente no hay grandes diferencias en cuanto a funcionalidad provista o patrones empleados, pero sí en cuanto a diseño y facilidad de uso. Angular es pesado y complejo, React es el más demandado laboralmente (ver 4.6) pero su curva de aprendizaje es pronunciada y Vue es el framework más ligero y sencillo de usar y aprender, siendo recomendado para proyectos con equipos pequeños o unipersonales.

En cuanto a popularidad según el número de estrellas en Github (tabla 4.1), se puede concluir que React y Vue son los más populares, quedando Angular bastante por detrás.

[23]Modelo, Vista, Vista-Modelo, que facilita la separación del desarrollo de la interfaz de usuario de la lógica de negocio o *backend*

Framework	Nº de estrellas en Github
Angular	40 133
React	110 129
Vue	112 650

Cuadro 4.1: Estrellas en Github de los principales frameworks Javascript en Septiembre de 2018

Por otra parte, a la vista una encuesta de stateofjs.com en 2017 (ver figura 4.5) sobre una muestra de más de 28.000 desarrolladores de todo el mundo, si bien React es el jugador dominante, Vue está haciendo grandes progresos en detrimento de Angular, lo que podría suponer un desafío para el liderazgo de React en 2018 y años posteriores.

Figura 4.5: Encuesta sobre el estado de Javascript en 2017 por stateofjs.com[]

Cabe destacar en esta memoria que Vue fue propuesto como alternativa para el desarrollo del proyecto por los miembros del *W3C RDFJS Community Group*, como puede comprobarse en la figura 4.7.

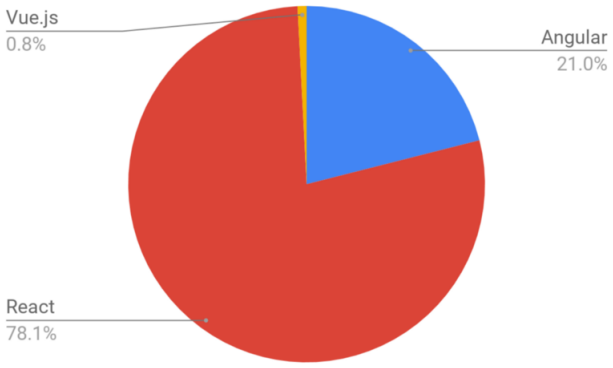

Figura 4.6: Ofertas de trabajo por *framework* JS según Medium.com a través de Indeed.com[1]

4.3. Recursos

Una vez analizados los trabajos previos existentes en 4.1 y la situación actual en 4.2, se discute en esta sección qué recursos que serán necesarios para el desarrollo del proyecto.

- En primer lugar, será necesario utilizar un sistema de control de versiones para el código desarrollado. Actualmente, el más utilizado es *git*, y la plataforma con mayor número de desarrolladores, *Github* (28 millones, según la propia Github[24]). Una cuenta gratuita bastará para alojar el código fuente y la documentación del proyecto, si bien *Github* ofrece cuentas privadas con duración limitada para estudiantes universitarios.
- Para llevar a cabo la planificación y evolución diaria de las tareas, se utilizará un plan gratuito de *Trello* (lo cual se detalla en profundidad en la subsección 5.2.2)
- A la vista de los objetivos del proyecto, Javascript ES6 parece la tecnología más adecuada para su desarrollo. Dada la inexperiencia del autor en esta tecnología, se proponen las siguientes referencias bibliográficas y de formación *online*:
  - *Get Programming With Javascript Next*, JD Isaacks[12].
  - Curso en la plataforma Udemy[25]: *ES6 Javascript: The Complete Developer's Guide*, Stephen Grider[13].
  - Curso en la plataforma Udemy: Working with Javascript Streams[14].

[24] https://github.com/search?q=type:user&type=Users
[25] https://www.udemy.com

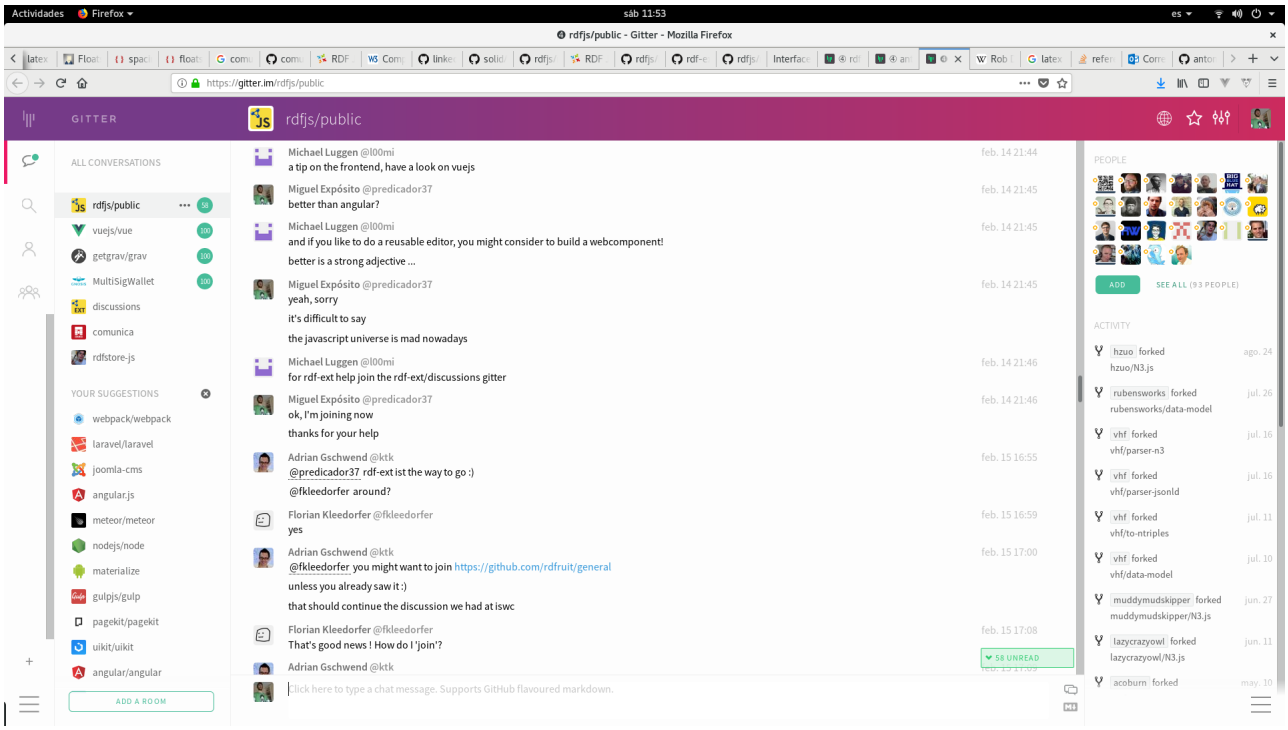

Figura 4.7: Recomendación de Vue en la charla con miembros del W3C RDFJS Community Group

- Analizadas las distintas implementaciones de RDF y SPARQL en Javascript y, siguiendo las recomendaciones del *W3C RDFJS Community Group*, se propone el uso inicial de los siguientes recursos:
  - RDF-Ext como biblioteca para trabajar con la API de RDF, siguiendo los estándares existentes.
  - Comunica para las consultas SPARQL, dado que aún con los riesgos inherentes que conlleva la utilización de un producto aún en desarrollo, es la única plataforma que puede responder a las necesidades del proyecto.
- Tras el estudio de la situación actual de los framerworks de front-end Javascript y, teniendo en cuenta asimismo las recomendaciones del *W3C RDFJS Community Group*, se opta por Vue para el proyecto dada su gran proyección y su curva de aprendizaje suave. En concreto, se propone utilizar los siguientes recursos:
  - Curso en la plataforma Udemy: *Vue JS 2 - The Complete Guide*, Maximilian Schwarzmüller[15], un referente en la comunidad de desarrolladores de Vue.
  - Curso en la plataforma Udemy: *The Ultimate Vue JS 2 Developers Course*, Anthony Gore[16]. Otro referente importante.
  - *Vue.js Design Patterns and Best Practices*, Paul Halliday[17].

- Como entorno de desarrollo, el autor de este proyecto ya contaba con una licencia de estudiante de WebStorm de Jetbrains[26], uno de los entornos integrados de desarrollo (IDE) más completos y modernos para Javascript existentes en la actualidad. Soporta Javascript de forma nativa, depuración con el navegador y frameworks como Vue, React o Angular, además de facilidades de pruebas unitarias, análisis estático de código, análisis de rendimiento, etc.

[26] https://www.jetbrains.com/webstorm/

Capítulo 5

Metodología y Planificación

5.1. Metodología

Para acometer este proyecto, se han valorado dos familias de metodologías de desarrollo de *software*:

1. Metodologías en cascada (*Waterfall*)
2. Metodologías ágiles (incrementales e iterativas)

La metodología de desarrollo en cascada surgió como idea en un artículo de Winston W. Royce en 1970[18]. Históricamente, este modelo se ha extendido tanto en ámbitos académicos como profesionales, siendo estas sus principales características:

- Gestión predictiva de proyectos llevada al software.
- Toma como modelo la forma de proceder en el resto de ingenierías.
- Intenta llenar el vacío del *code & fix*.
- Cada fase se realiza, en principio, una única vez.
- Cada fase produce un entregable que será entrada de la siguiente.
- Los entregables no son, en principio, modificables.

Es decir, se basa en la separación entre diseño y construcción (o entre creatividad y repetición).

La propuesta de Royce,tal y como se desprende de la lectura del artículo original, describía el modelo en cascada como la «descripción más simple» ([18]) que solo funcionaría para los proyectos más sencillos. Irónicamente, este mensaje malentendido ha sido el origen de la popularidad de la metodología en cascada, que hoy en día se sigue promoviendo en muchos casos por inercia, desconocimiento, comodidad o ilusión de control sobre el proyecto.

Lamentablemente, en la *Ingeniería de Software* los pesos de diseño y construcción están invertidos con respecto a otras ingenierías, siendo el *software* un dominio de cambio y alta inestabilidad. El desarrollo de *software* es, intrínsecamente, una labor creativa; y la creatividad no es fácilmente predecible. Esto ha dado lugar a que los desarrollos tradicionales adolezcan de ciertos problemas[19]:

- Existencia de muchos requisitos vagos o especulativos y diseño detallado por adelantado.
- Están fuertemente asociados con las tasas de fallo más altas en proyectos.
- Se encuentran promovidos históricamente por creencia más que por evidencia estadística significativa.
- Su rigidez incrementa el riesgo de fracaso, pospuesto hasta las fases finales del proyecto.
- Asume que las especificaciones son predecibles, estables y completas.
- Pospone integración y pruebas hasta fases tardías.
- Se basa en estimaciones y planificación "fiables".

Entre los estudios que ratifican las afirmaciones anteriores, cabe citar los siguientes:

- Informe Chaos 2015[20].
- *Dr. Dobb's Journal article The Non'Existent Software Crisis: Debunking the Chaos Report*[21].
- Encuesta de Gartner[22].
- TODO

Por tanto, atendiendo a esta exposición y considerando que el proyecto en cuestión presentaba un alto nivel de incertidumbre debido a su alto componente en investigación del estado tecnológico actual, se ha optado por utilizar un enfoque metodológico incremental e iterativo, puesto que:

- Facilita llevar a cabo proyectos pequeños.
- Fomenta la interacción entre el desarrollador y el usuario.
- Fuerza a que los inevitables cambios en requisitos sucedan en fases tempranas del proyecto.

Entre sus características es posible citar:

- Se trabaja sobre subconjuntos de funcionalidad (*features*).
- Los incrementos permiten añadir funcionalidad al producto (mejora del proceso).
- Las iteraciones permiten rediseñar, revisar y refactorizar el producto (mejora del producto).
- Se basa en entregas frecuentes y ciclos prueba/error.
- Ofrece flexibilidad a la hora de gestionar el cambio.

La referencia más clara en este ámbito es *Extreme Programming*, de Kent Beck[23]. *Extreme Programming* es «un estilo de desarrollo de software centrado en la aplicación excelente de técnicas de programación, comunicación clara y trabajo en equipo que permite conseguir objetivos antes impensables» ([18]). Se trata de una metodología basada en valores como la comunicación, realimentación, simplicidad, valentía y respeto, soportada sobre un cuerpo de prácticas útiles y con un conjunto de principios complementarios, además de contar con una comunidad de usuarios que comparte todo lo anterior.

Su aplicación al desarrollo de este proyecto no ha sido estricta; por ejemplo, no se han definido ciclos estrictos por la propia naturaleza inestable de la dedicación al desarrollo del mismo, pero sí que ha realizado un diseño evolutivo soportado sobre un número suficiente de pruebas unitarias así como una planificación incremental y adaptativa a las problemáticas que iban surgiendo.

Prueba del acierto a la hora de elegir esta metodología es el cambio de necesidades y requisitos funcionales que tuvo lugar en la reunión presencial mantenida con el tutor, donde **la metodología aportó que no fuera necesario descartar ningún desarrollo o trabajo realizado hasta la fecha**. Permitió realizar una gestión del cambio efectiva, re-orientando el trabajo a tiempo sin impactar en el diseño ni en la implementación existente. Finalmente, tanto la organización como la planificación de las tareas han sido lo suficientemente flexibles para conseguir un ritmo adecuado de desarrollo, reduciendo los puntos de bloqueo.

5.2. Planificación

5.2.1. Planificación global

Para realizar la planificación del proyecto, y dada la metodología de desarrollo incremental e iterativa elegida, no se ha seguido un modelo típico en fases de *Análisis, Diseño, Implementación, etc.* sino que se ha optado por un enfoque orientado a tareas relacionadas con la funcionalidad del producto final esperado.

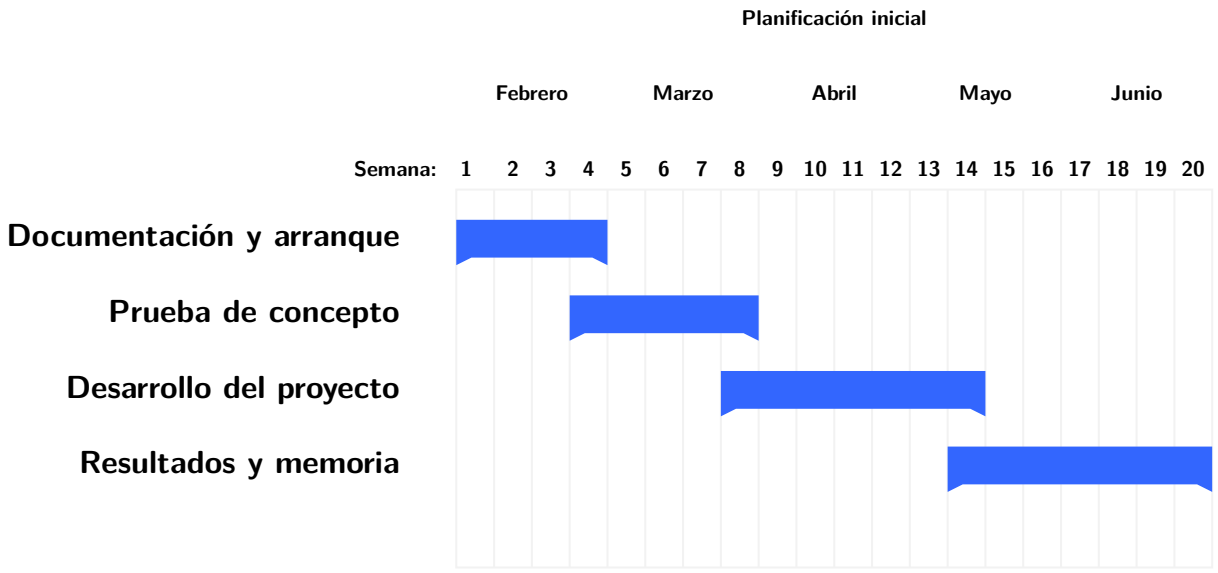

Figura 5.1: Planificación inicial

La figura anterior (5.1) muestra la planificación propuesta para la elaboración del anteproyecto.

En esta propuesta era clave la revisión de la prueba de concepto con el tutor para comprobar que efectivamente se habían comprendido las necesidades desprendidas del análisis y para poder continuar con la especificación funcional de forma más refinada y precisa.

En la práctica, la evolución del proyecto ha sido bien distinta. En la figura 5.2 se puede comprobar cuál ha sido la planificación efectiva, producto esta de cambios sobre la inicial para resolver los distintos imprevistos que han ido surgiendo durante la ejecución del mismo.

Con respecto a la planificación inicial, se tiene que:

- El proyecto se ha retrasado un total de ocho semanas.
- El período de formación llevó al menos dos semanas más de lo esperado (en realidad, el aprendizaje del *framework Vue* ha estado presente a lo largo de prácticamente todo el desarrollo del proyecto).
- El desarrollo del producto ha consumido seis semanas más de las previstas inicialmente.
- La memoria se ha redactado en dos semanas menos con respecto a la estimación inicial.

Las desviaciones surgidas con respecto a la planificación inicial tienen su explicación en los siguientes motivos:

- El alto grado de incertidumbre a la hora de estimar la planificación inicial, dado que se desconocía cuál era la situación actual del ecosistema tecnológico de *front-end web* y su integración con las tecnologías de la Web Semántica.
- La curva de aprendizaje de *Vue*, si bien es considerada más suave que la de sus competidores (*React, Angular*) fue mayor de lo esperado. El bajo grado de familiarización del autor del proyecto con las tecnologías de *front-end* y, especialmente, *Javascript ES6+*, no facilitó el aprendizaje.
- El **bajo nivel de madurez de las bibliotecas existentes** para la manipulación de RDF en Javascript, así como la heterogeneidad y poca estabilidad de estas implementaciones, ha suscitado muchas dudas sobre cuáles utilizar y cómo enfocar su integración con *frameworks* más maduros como *Vue*.

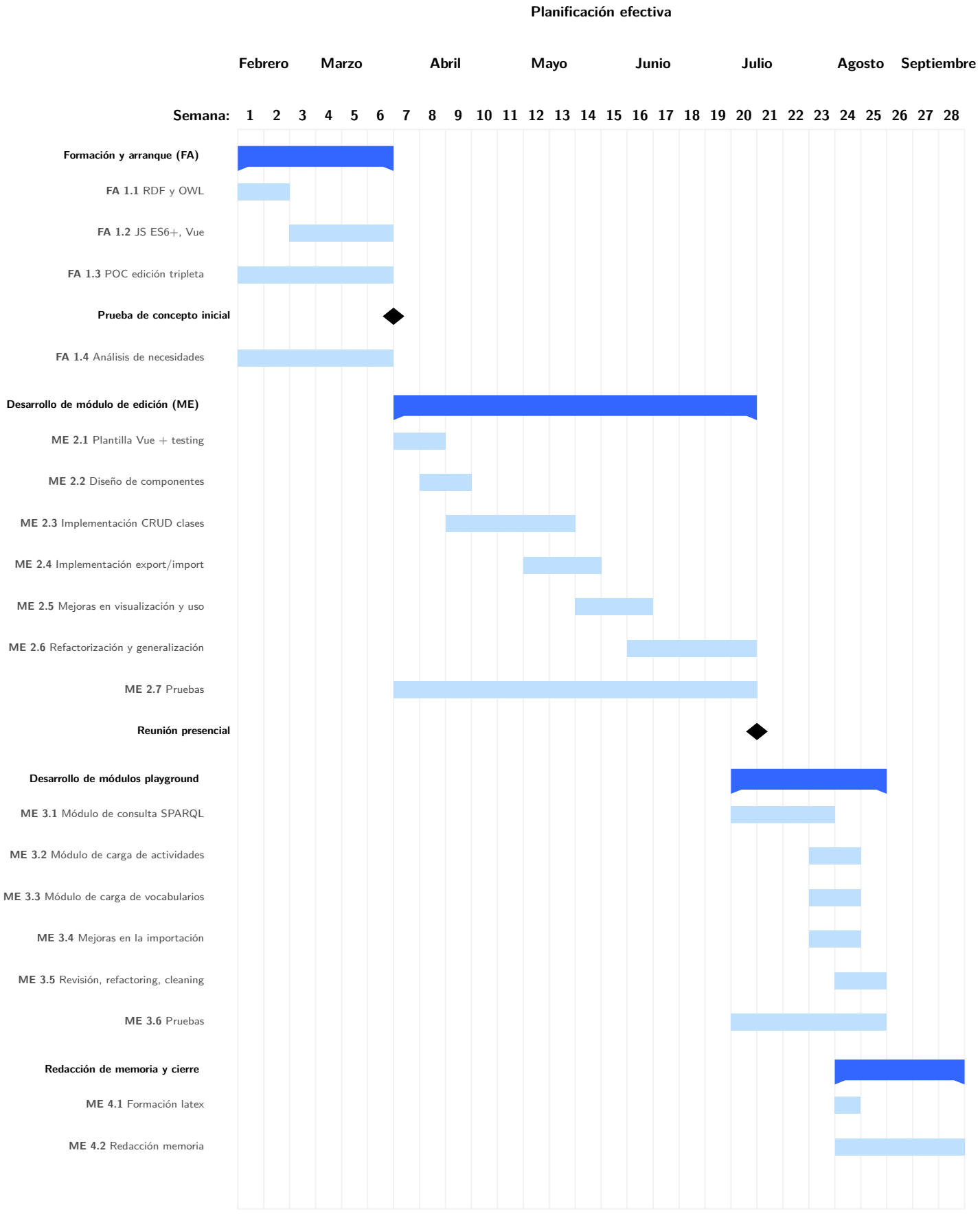

Figura 5.2: Planificación efectiva

- La **práctica ausencia de bibliotecas** o componentes de interacción con SPARQL **conformes a la especificación estándar de la interfaz** *rdf.js*[24] (que por otra parte, es un borrador de 2017) ha puesto en peligro la viabilidad del proyecto. La plataforma finalmente utilizada, *Comunica*[25], aún no tiene una versión estable muchos de sus módulos (concretamente, el endpoint SPARQL utilizado para el proyecto no la tiene), con lo que ha sido necesario estar en contacto directo con sus autores y colaborar con ellos en la revisión de defectos o *bugs* y dependiendo por tanto de sus tiempos de respuesta (hay que tener en cuenta que la plataforma es *opensource* y por tanto no existe acuerdo de nivel de servicio alguno.)
- La dificultad existente en llevar a cabo un análisis de requisitos a través de una plataforma online de colaboración como puede ser *Skype*: para constatar este hecho no hay más que verificar el cambio de rumbo del proyecto una vez mantenida la reunión presencial, que sirvió para definir objetivos más claros y comprender las necesidades del Departamento.

A pesar de todo ello, el autor de este proyecto está satisfecho con el nivel de conocimiento adquirido en el ámbito de todas las tecnologías empleadas y el tiempo consumido para obtener como resultado un producto desplegado en producción y listo para utilizar.

5.2.2. Planificación ágil

Si bien para la planificación global del proyecto se han utilizado herramientas tales como el *diagrama de Gantt*, para gestionar el trabajo del día a día otro enfoque ha sido necesario. En el contexto de metodologías ágiles de desarrollo de software de tipo incremental e iterativo como Extreme Programming, **las tareas de planificación adquieren un carácter adaptativo** muy distinto del que presenta una planificación tradicional con una metodología de desarrollo en cascada, por ejemplo.

La planificación ágil es un proceso en continua evolución, también iterativo e incremental como las metodologías a las que pertenece, basada en ciclos del tipo:

1. Añadir tareas
2. Estimar tareas
3. Priorizar tareas

En este caso, la estimación de tareas se llevó a cabo de forma heurística, utilizando la experiencia del autor y el conocimiento del contexto existente en el momento de la estimación. En base a ello, la priorización (ordenación) de las tareas tenía lugar inmediatamente, relegando a la estimación a un segundo plano.

Para gestionar estas tareas, se ha optado por utilizar como herramienta Kanban. Un tablero Kanban puede definirse como un dispositivo de señalización que introduce el flujo de trabajo de un proceso a un ritmo manejable. Presenta las siguientes características:

- Solo envía trabajo cuando lo ordena el cliente o usuario (en este caso, el propio autor del proyecto)
- Indica específicamente qué trabajo debe hacerse.
- Controla la cantidad de trabajo en progreso.
- Regula las interrupciones y orquesta el ritmo de trabajo.

Básicamente, consiste en utilizar una tabla con varias columnas para visualizar el estado de una tarea a lo largo de las distintas fases que se consideren. Para el caso de la realización de este proyecto, se crearon las siguientes columnas:

Columna	Descripción
To-Do	Tareas por realizar en orden de prioridad descendente
Work in progress	Tareas realizándose en un momento dado. No más de dos o tres.
Stand-by	Tareas a la espera por motivos ajenos al autor del proyecto.
Done	Tareas finalizadas.
Discarded	Tareas descartadas debido a cambios de diseño, requisitos, etc.
Doubts	Dudas planteadas a lo largo del desarrollo del proyecto.
Resources	Recursos documentales en la web útiles para del desarrollo del proyecto.

Cuadro 5.1: Diseño del tablero Kanban

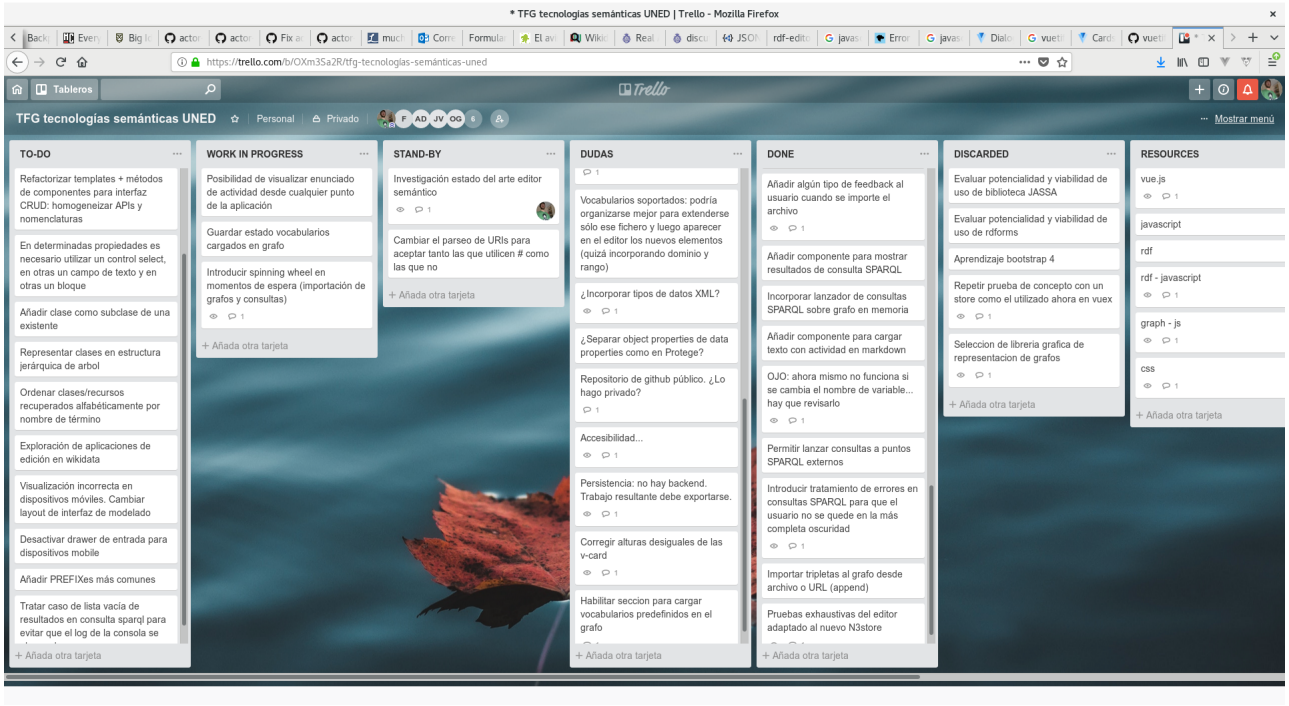

Figura 5.3: Ejemplo de tablero Trello

Capítulo 6

Análisis

6.1. Captura y documentación de requisitos

6.1.1. Captura de requisitos

La técnica de captura de requisitos utilizada para este proyecto ha sido, fundamentalmente, **la entrevista** con el tutor. Se eligió esta técnica por los siguientes motivos:

- Las entrevistas, bien a distancia o presenciales (y especialmente estas últimas), permiten una mayor implicación del usuario en la captura de requisitos.
- Combinada con una maqueta o prueba de concepto, una entrevista presencial puede dar lugar a la aparición de nuevos requisitos de producto, cambios en las especificaciones e incluso en el enfoque y objetivos del mismo.
- Permite la práctica de la escucha activa y la sugerencia de ideas por parte del analista, aportando un valor añadido que enriquece la simple captura de requisitos.
- Es claramente la técnica más obvia, directa y accesible en el contexto de la realización del proyecto.

Concretamente, se han llevado a cabo varias entrevistas utilizando la plataforma colaborativa Skype y una presencial, en la que el autor de este proyecto se ha desplazado a la sede del departamento en Madrid con objeto de conseguir una comunicación más fluida y un mayor entendimiento a la hora de consensuar las necesidades y funcionalidades requeridas del producto.

La primera entrevista a distancia propició un intercambio de documentos e ideas que desembocó en la elaboración del documento del anteproyecto. El resto de entrevistas a través de Skype sirvieron para concretar en mayor medida las tareas a realizar y el enfoque del proyecto. Sin embargo, no fue hasta que no tuvo lugar la reunión presencial cuando realmente se le dio al proyecto su orientación final, con unos objetivos claramente definidos y una posibilidad para cumplir los hitos propuestos.

A continuación se resumen todas las sesiones de captura de requisitos:

id	Fecha	Resumen de la entrevista
1	18/10/17	Primer contacto y comunicación de ideas iniciales para la confección del anteproyecto
2	21/02/17	Consolidación de ideas y aportación de más documentación (vídeos sobre prototipos de cuaderno, documentos de texto con descripciones, etc.)
3	28/03/18	Primera demo a modo de POC con un entorno capaz de añadir tripletas.
4	10/07/18	Reunión presencial con demostración *in-situ* de los módulos de modelado e importación/exportación. Tiene lugar una tormenta de ideas y se enfoca el proyecto de otro modo, modificando sus objetivos hacia una herramienta formativa..
5	4/08/18	Revisión de los últimos avances con la integración de un endpoint SPARQL en el frontend y planificación del resto de funcionalidade requeridas.

Cuadro 6.1: Sesiones de entrevistas

6.1.2. Documentación

Para documentar la captura de requisitos, se utilizará la técnica de casos de uso. Se descarta la incorporación de diagramas UML de casos de uso, dado que dichos diagramas carecen de información esencial sobre los mismos (como qué actor lleva a cabo cada paso, o notas sobre el orden de ejecución de los pasos). Si bien pueden ser útiles como resumen o índice de contenidos, se decide prescindir de ellos dado que el número de casos de uso contemplados en el proyecto es manejable.

Se utilizará una plantilla propuesta por Cockburn[26]: el estilo RUP (*Rational Unified Pro-*

cess)[27], atractivo y fácil de seguir pese al elevado número de apartados, modificado para plasmar los aspectos más relevantes del proyecto (por ejemplo, no se incluirá un campo *ámbito* porque siempre va a estar referido al mismo sistema o aplicación). El motivo de no utilizar una tabla es meramente subjetivo, ya que el autor de esta memoria opina que puede oscurecer el contenido.

La plantilla sigue la siguiente estructura:

1. Nombre del caso de uso
    a) Descripción breve
    b) Actores, entre los que estará el actor principal. Presentan comportamiento.
    c) Disparadores: acciones sobre el sistema que inician los casos de uso.
2. Flujo de eventos
    a) Flujo básico: escenario principal de éxito.
    b) Flujos alternativos: qué puede pasar que no sea el flujo principal.
        1) Condicion 1
        2) Condición 2
        3) ...
3. Requisitos especiales (si se dieran): plataforma, etc.
4. Precondiciones: qué debe ser cierto antes de ejecutar el caso de uso.
5. Postcondiciones: qué debe ser cierto después de ejecutar el caso de uso.

Los requisitos fueron capturados inicialmente como notas manuscritas y convertidos en necesidades de alto nivel en la plataforma Trello. A partir de ahí, dichas necesidades se refinaron para dar lugar a la batería de casos de uso incluida en 6.3.

6.2. Necesidades

La reunión presencial marcó un punto de inflexión en cuanto a objetivos del proyecto, lo que se traduce en un cambio de necesidades. Para reflejar la evolución completa, se dividirá su captura en dos fases detalladas a continuación:

6.2.1. Captura inicial

A continuación se resumen, en lenguaje natural, las necesidades identificadas durante la primera fase de desarrollo del proyecto:

1. Permitir a usuarios de la UNED de distintos colectivos etiquetar y generar sus propios cuadernos con información y metainformación semántica.
2. Permitir a dichos usuarios realizar consultas sobre sus cuadernos.
3. Desarrollar una interfaz web que permita al usuario gestionar tripletas RDF.
4. Desarrollar un módulo de generación de consultas SPARQL a partir de consultas de lectura y escritura en formato JSON.
5. Desarrollar los correspondientes productos de interés para el usuario: consultas exportadas en forma diversa (CSV, JSON) o embebidas en una plantilla HTML significativa para el usuario.
6. Ofrecer la posibilidad de variar interfaces de entrada y exportadores en función de los distintos colectivos de usuario utilizando los metadatos sobre cuadernos RDF previamente almacenados en una base de datos relacional.
7. Permitir mostrar en pantalla una serie de términos como punto de partida que el usuario pueda utilizar para construir ternas RDF y relaciones entre ellas.
8. Permitir mostrar en pantalla una serie de términos como punto de partida que el usuario pueda utilizar para construir ternas RDF y relaciones entre ellas.
9. Ofrecer al usuario una visualización sencilla y correcta de su modelo que proporciona una perspectiva adecuada sobre la que trabajar.
10. Presentar una interfaz de mantenimiento del grafo: vocabulario e instancias (conceptualización y poblamiento de una ontología).
11. Importar y exportar información estructurada en formatos semánticos estándar.
12. Extender con vocabularios tales como SKOS y OWL.

6.2.2. Captura final

Una vez celebrada la reunión presencial, se decidió darle otro enfoque al proyecto. Si bien la idea inicial era desarrollar un sistema que permitiese generar cuadernos a través de la manipulación de grafos, una vez presentada una maqueta o prueba de concepto con funcionalidades básicas de modelado el tutor propuso convertir la herramienta en un *playground* o sistema de realización de actividades académicas con un enfoque docente orientado a facilitar el aprendizaje de las tecnologías de la Web Semántica (básicamente RDF y SPARQL) a personas con poco o ningún contacto con estas materias (por ejemplo, alumnos de los Grados de Ingeniería Informática o en Tecnologías de la Información de la UNED).

Este nuevo enfoque se tradujo en la siguiente instantánea de necesidades de alto nivel:

1. Permitir el modelado semántico con edición CRUD de clases, subclases y propiedades de clases (anotaciones básicas).
2. Permitir la edición CRUD de relaciones o propiedades, subpropiedades, etc.
3. Ofrecer mecanismos de poblamiento del grafo.
4. Permitir el lanzamiento de consultas SPARQL sobre el grafo local y visualización de resultados.
5. Permitir el lanzamiento de consultas SPARQL sobre endpoints remotos y visualización de resultados.
6. Ofrecer funcionalidad de carga de consultas SPARQL predefinidas desde archivo de texto.
7. Incorporar módulo para añadir un texto con la definición de la actividad a realizar en formato Markdown.
8. Permitir la importación de tripletas desde archivos o URL.
9. Permitir la incorporación (append) de tripletas al grafo local desde archivos o URL.
10. Ofrecer un panel para cargar en el grafo vocabularios comunes predefinidos.
11. Presentar una interfaz fácil de usar y responsiva para el usuario, con una gestión de errores adecuada y suficiente.

6.3. Casos de uso

6.3.1. Dar de alta una nueva clase

6.3.1.1. Descripción breve

Este caso de uso permite a un usuario añadir una nueva clase a su grafo a través de una tripleta que tendrá como sujeto el nombre de la clase, como objeto el recurso owl:Class y como predicado el recurso rdf:type.

6.3.1.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.1.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el botón flotante + asociado a la tarjeta del listado de clases.

6.3.1.4. Flujo de eventos

1. Flujo básico
    a) El usuario pulsa en el botón flotante + asociado a la tarjeta del listado de clases.
    b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre de la clase.
    c) El usuario introduce el URI del recurso que quiere dar de alta como clase y pulsa en guardar.
    d) El nuevo recurso de tipo clase se crea y aparece en el listado de clases.
2. Flujo alternativo 1

a) El usuario pulsa en el botón flotante + asociado a la tarjeta del listado de clases.

b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre de la clase.

c) El usuario introduce el URI del recurso que quiere dar de alta como clase y pulsa en guardar.

d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.1.5. Requisitos especiales

Ninguno.

6.3.1.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien a través de la barra de navegación lateral.

6.3.1.7. Postcondiciones

Ninguna.

6.3.1.8. Extensiones

Ninguna.

6.3.2. Editar una clase existente

6.3.2.1. Descripción breve

Este caso de uso permite a un usuario editar el recurso asociado a una clase existente en su grafo y cambiar su URI.

6.3.2.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.2.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción ."Editar".

6.3.2.4. Flujo de eventos

1. Flujo básico
    a) El usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción ."Editar".
    b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
    c) El usuario modifica (o no) el URI del recurso y pulsa en guardar.
    d) El recurso ha sido modificado y su cambio se ve reflejado en el listado.
2. Flujo alternativo 1
    a) El usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción ."Editar".
    b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
    c) El usuario modifica (o no) el URI del recurso y pulsa en guardar.
    d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.2.5. Requisitos especiales

Ninguno.

6.3.2.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien a través de la barra de navegación lateral.

6.3.2.7. Postcondiciones

Ninguna.

6.3.2.8. Extensiones

Ninguna.

6.3.3. Eliminar una clase existente

6.3.3.1. Descripción breve

Este caso de uso permite a un usuario eliminar el recurso asociado a una clase existente en su grafo.

6.3.3.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.3.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción "Eliminar".

6.3.3.4. Flujo de eventos

1. Flujo básico
    a) El usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción "Eliminar".
    b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.
    c) El usuario elige confirmar.
    d) El recurso y su detalle asociado desaparecen del listado.
2. Flujo alternativo 1
    a) El usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción "Eliminar".
    b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.
    c) El usuario elige cancelar.
    d) El recurso y su detalle siguen apareciendo y no son eliminados.
3. Flujo alternativo 2
    a) El usuario pulsa en el icono de menú asociado a la clase que quiere editar y selecciona la acción "Eliminar".
    b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.
    c) El usuario elige confirmar.
    d) Ocurre un error en el borrado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.3.5. Requisitos especiales

Ninguno.

6.3.3.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien a través de la barra de navegación lateral.

6.3.3.7. Postcondiciones

Ninguna.

6.3.3.8. Extensiones

Ninguna.

Capítulo 7

Diseño

7.1. Arquitectura de una aplicación semántica

Independientemente de la tecnología elegida para su implementación, una aplicación semántica genérica se caracteriza por una arquitectura concreta que se va a proceder a detallar. Dicha arquitectura está basada en componentes que pueden ser proporcionados por uno o varios productos de mercado. Es común encontrar los siguientes:

- **Almacén de ternas** (*store*): no es más que una base de datos que permite el almacenamiento y extracción de ternas RDF, así como consolidar información procedente de distintas fuentes de datos.
- **Procesador/Serializador** (*parser/serializer*): un procesador RDF lee ficheros de texto y los interpreta como ternas RDF; un serializador realiza el proceso inverso.
- **Motor de consultas** (*query engine*): provee a la aplicación de la funcionalidad de recuperación de información en base a consultas estructuradas.

A continuación se estudian con más detalle, ofreciendo ejemplos de los más utilizados y el caso concreto del proyecto que atañe a esta memoria.

7.1.1. Almacén RDF (*store*)

El almacén de ternas es un componente esencial para salvaguardar conjuntos de datos en ternas independientemente de su origen (un fichero serializado, el resultado de una consulta, etc.). Conceptualmente, el equivalente en una base de datos relacional sería una tabla con tres columnas (Sujeto

- Predicado - Objeto). Sin embargo, parece evidente que un almacén RDF esté optimizado para el almacenamiento y recuperación de ternas RDF.

Comparados con una base de datos relacional, sin embargo, son más flexibles y requieren de menor coste de uso y mantenimiento. Más concretamente:

- **Flexibilidad:** su esquema flexible permite realizar cambios sin paradas ni rediseños, cosa que no ocurre con una base de datos relacional.
- **Estandarización:** el nivel de estandarización de RDF y SPARQL es mucho mayor que el de SQL. Es prácticamente inmediato sustituir un almacén de tripletas por otro, mientras que en el caso de las bases de datos relacionales es necesario tener en cuenta los distintos dialectos e implementaciones de SQL según el fabricante.
- **Expresividad:** es mucho más sencillo modelar datos complejos y con elevado número de relaciones entre ellos en RDF que en SQL. Ocurre al contrario si los datos son de naturaleza tabular, naturalmente.

No todo son ventajas en la comparación:

- **Madurez:** las bases de datos relacionales son mucho más maduras y presentan más funcionalidades que los almacenes de ternas.
- **Coste de almacenamiento:** El coste por unidad de información almacenada en un almacén de ternas es mucho mayor que el de una base de datos relacional, lo cual puede ser crítico si se están tratando grandes volúmenes de datos.

Entre los almacenes de ternas más utilizados, se pueden citar OpenLink Virtuoso[1], GraphDB de Ontotext[2], Apache Jena TDB[3] o AllegroGraph[4].

7.1.1.1. Almacén en el proyecto

En el caso del presente proyecto, se valoraron dos alternativas:

[1] https://virtuoso.openlinksw.com/
[2] https://ontotext.com/products/graphdb/
[3] https://jena.apache.org/documentation/tdb/
[4] https://franz.com/agraph/allegrograph/

- **RDF-Ext Dataset**[5]**:** un simple conjunto de datos cargado en memoria y conforme con la especificación estándar. Este almacén presentaba limitaciones en cuanto a rendimiento y número de tripletas a almacenar, además de estar pensado para operaciones síncronas.
- **N3.js:**[6] La bilioteca N3 ofrece, además de almacenamiento de tripletas en memoria en Javascript nativo, facilidades de serialización y procesado con formatos estándar como Turtle, N3, N-Triples y TriG. Si bien su conformidad con la especificación de interfaces RDF.js no es completa, sí está presente en gran parte de sus módulos (*DataFactory, StreamParser, StreamWriter y Store*).

Confirmadas las limitaciones de RDF-Ext Dataset y consultado con el *W3C RDFJS Community Group*, se opta finalmente por utilizar **N3.js** (lo que supuso adaptar ligeramente la implementación del proyecto al nuevo *storage*).

7.1.2. Procesador y Serializador RDF (*parser/serializer*)

En muchos casos, los módulos para importación y exportación de datos en RDF son proporcionados por el propio almacén. Sin entrar a valorar su sintaxis dada su complejidad específica y por considerarse fuera del interés de este proyecto, sí se va a proceder a identificar y caracterizar los formatos de serialización más comunes.

7.1.2.1. RDF/XML

Se trata de una representación en XML, comúnmente criticada por su dificultad de lectura por parte de personas. Entre sus otras críticas están las limitaciones impuestas por las reglas de nomenclatura de XML o los problemas encontrados para procesar este formato con herramientas populares para XML[28]. Para poder codificar un grafo RDF en XML, tanto los nodos como los predicados han de ser representados en términos de XML (elementos, atributos, contenido de elementos y valores de atributos).

Conceptualmente, se construye a partir de un conjunto de descripciones más pequeñas, cada una de las cuales traza un camino a lo largo de un grafo RDF. Estos caminos se describen en términos

[5] https://github.com/rdf-ext/rdf-store-dataset
[6] https://github.com/rdfjs/N3.js/

de los nodos (sujetos) y enlaces (predicados), que los conectan con otros nodos (objetos).

A continuación se muestra un ejemplo de serialización en RDF/XML[7]:

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
<ex:editor>
<rdf:Description>
<ex:homePage>
<rdf:Description rdf:about="http://purl.org/net/dajobe/">
</rdf:Description>
</ex:homePage>
</rdf:Description>
</ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
<ex:editor>
<rdf:Description>
<ex:fullName>Dave Beckett</ex:fullName>
</rdf:Description>
</ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
<dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

Listado de código 7.1: Ejemplo de RDF/XML del W3C.

7.1.2.2. N-Triples

Se trata de una notación muy simple pero verbosa, donde cada línea representa una única declaración conteniendo sujeto, predicado y objeto seguidos por un punto.

Ejemplo[8]:

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin> .
```

Listado de código 7.2: Ejemplo de N-Triples del W3C.

[7] https://www.w3.org/TR/rdf-syntax-grammar/#example3
[8] https://www.w3.org/TR/n-triples/

7.1.2.3. N3

N3, abreviatura de *Notation 3*, fue un proyecto personal de Tim Berners-Lee[28]. Es muy similar a N-Triples, pero añadiendo características adicionales como atajos, un formato más claro, o la condensación de muchas de las repeticiones de este formato.

A pesar de todo, N3 nunca se convirtió en un estándar y sus características adicionales frente a N-Triples no tuvieron mucha aceptación.

7.1.2.4. Turtle

El formato Turtle permite escribir un grafo RDF en texto de una forma más compacta que RDF/XML y más sencilla de leer que N-Triples. Su gramática es un subconjunto de la especificación del lenguaje de consulta SPARQL 1.1, compartiendo ambas nombres de terminales y producciones en la medida de lo posible.

En estos momentos, es el formato de serialización más popular entre las comunidades de desarrolladores[29].

A continuación se muestra un ejemplo de serialización en Turtle[9]:

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
rel:enemyOf <#spiderman> ;
a foaf:Person ;    # in the context of the Marvel universe
foaf:name "Green Goblin" .

<#spiderman>
rel:enemyOf <#green-goblin> ;
a foaf:Person ;
foaf:name "Spiderman" .
```

Listado de código 7.3: Ejemplo de Turtle del W3C.

[9] https://www.w3.org/TR/turtle/

7.1.2.5. Procesadores y serializadores en el proyecto

El almacén seleccionado, **N3.js, proporciona procesadores y serializadores** para los principales formatos citados: N3, N-Triples, Turtle y TriG (una extensión de este último).

No se ha considerado necesario diseñar ningún serializador o procesador personalizado, al estar ya incluidos los formatos más comunes utilizados por desarrolladores (nótese la ausencia deliberada de RDF/XML).

7.1.3. Motor de consulta RDF (*query engine*)

El motor de consulta es un componente íntimamente ligado al almacén de ternas. El W3C fijó un estándar para consultas: SPARQL (presentado en **??**), cuya especificación está en su versión 1.1 desde 2013.

Desde el punto de vista de la arquitectura de una aplicación semántica, es deseable que el motor de consultas, además de cumplir con la especificación, esté íntimamente integrado con el almacén de ternas para ofrecer un rendimiento aceptable.

Los principales proveedores de almacenes de ternas ofrecen también motores de consulta integrados, como Apache Jena, Virtuoso, AllegroGraph...

7.1.3.1. Motor de consultas en el proyecto

En el caso del presente proyecto, integrar un motor de consultas no fue sencillo. El componente N3.js no ofrece dichas funcionalidades, con lo que fue necesario estudiar la viabilidad de incorporar un motor local totalmente desarrollado en Javascript.

Tras el estudio de los trabajos previos desarrollados (**??**) y la situación actual (**??**), se llegó a la conclusión de que la plataforma Comunica se posicionaba como el producto abierto más prometedor para satisfacer las necesidades de consulta SPARQL.

Comunica proporciona las herramientas necesarias para crear una aplicación combinando múltiples bloques de construcción independientes. Su propósito es ofrecer una implementación modular de un

cliente *Triple Pattern Fragment* o *Linked Data Fragment*, un tipo de fragmento que consiste en:

- **Datos** que corresponden a un patrón de tripleta.
- **Metadatos** que consisten en el total aproximado de tripletas.
- **Controles** que llevan al resto de fragmentos del mismo conjunto de datos.

De forma resumida, un cliente que soporta este tipo de fragmentos puede resolver múltiples consultas SPARQL de forma eficiente.

Comunica consta de los siguientes componentes:

- **Actores:** definen el formato de la entrada que aceptan y la salida correspondiente que producen.
- **Buses:** combinan los actores que soportan el mismo formato de entrada y salida y permite enviar mensajes a todos los actores registrados en un bus dado.
- **Mediadores:** envuelven los buses y se aseguran de que cada petición recibe tan sólo una respuesta.

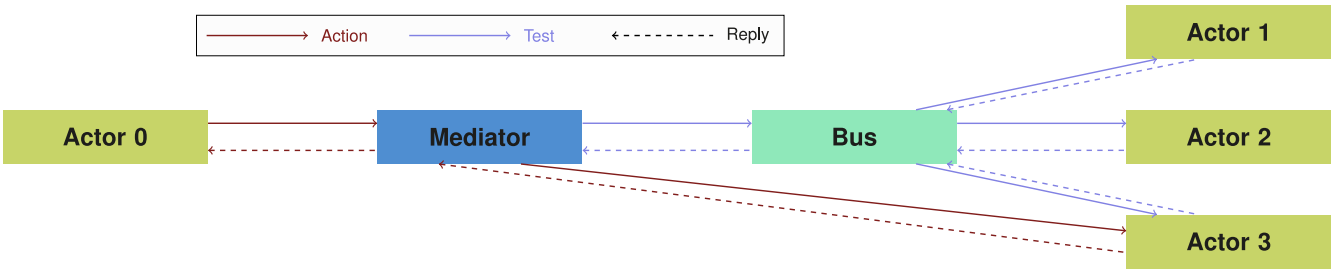

Figura 7.1: Arquitectura de Comunica

Para el presente proyecto, han sido necesarios dos actores:

- **actor-init-sparql:** se trata de un cliente que permite resolver consultas sobre interfaces heterogéneos. En concreto, este módulo inicializa un motor Comunica con actores que evalúan consultas SPARQL.

- **actor-init-sparql-rdfjs:** permite lanzar consultas SPARQL a fuentes que implementan la interfaz *Source*[10]. Una de esas fuentes es N3.js, el *store* utilizado para almacenar ternas en el proyecto.

Es decir, se utiliza un módulo para lanzar consultas a *endpoints* SPARQL remotos y otro para la fuente o almacén local. Estos módulos **aún están en desarrollo y no han alcanzado una versión estable** a fecha de redacción de la presente memoria, con lo que se van adquiriendo nuevas funcionalidades y corrigiendo defectos a lo largo del tiempo de desarrollo de este proyecto (por ejemplo, en Julio de 2018 el motor de consulta no soportaba la palabra clave SERVICE, característica implementada en Agosto).

7.2. Arquitectura de la aplicación Vue

Al margen de su naturaleza semántica, la herramienta desarrollada se implementa en Javascript ES6+ y Vue.js. Es necesario, por tanto, proponer un diseño de alto nivel de sus componentes.

7.2.1. Almacenamiento de estado

Para almacenar el estado de la aplicación se ha optado por usar Vuex[11].

Vuex es una biblioteca que implementa un patrón de gestión de estado (ver figura 7.2[30]) basado en Redux[12], Flux[13] y la arquitectura Elm[14]. Se utiliza como un almacén centralizado para todos los componentes de la aplicación, y permite asegurar:

- Que todos los componentes comparten ese estado.
- Que el estado sólo podrá ser modificado de forma controlada, bien síncrona o asíncrona.

Sin entrar en detalles, el funcionamiento de Vuex se ciñe a las siguientes reglas:

[10] http://rdf.js.org/#source-interface
[11] https://vuex.vuejs.org/
[12] Redux es un contenedor predecible del estado de aplicaciones Javascript. Ver https://es.redux.js.org/
[13] Una arquitectura de aplicación para React que utiliza un flujo de datos unidireccional. Ver https://github.com/facebook/flux
[14] Un patrón sencillo de arquitectura de aplicaciones web. Ver https://guide.elm-lang.org/architecture/

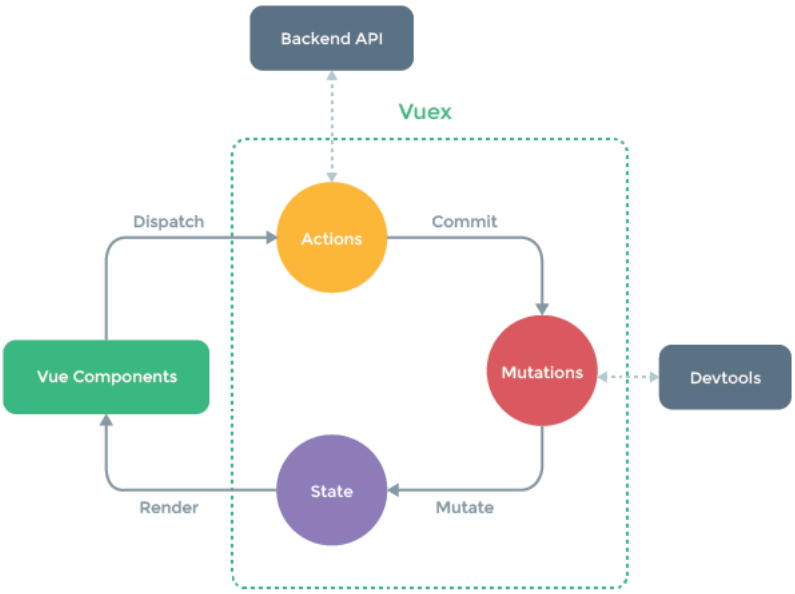

Figura 7.2: Vuex: gestión de estado centralizado.

- Existe un único estado compartido por todos los componentes de la aplicación, independientemente de que cada componente tenga su propio estado individual.
- El acceso al estado centralizado se realiza a través de **getters**.
- El estado sólo puede ser modificado a través de **mutaciones síncronas**.
- Las mutaciones pueden ser ejecutadas de manera asíncrona a través de **acciones**.

7.2.2. Arquitectura de componentes

Para poder dar cumplimiento a los requisitos de la aplicación, se definen en el cuadro 7.1 los componentes necesarios con sus ámbitos de responsabilidad.

En la figura 7.3 se muestra un diagrama UML 2.0 con la jerarquía de componentes de la aplicación, que muestra convenientemente las relaciones entre los mismos distinguiéndose tres niveles:

- Componente raíz de la aplicación, del que dependen todos los demás (actúa de enrutador).
- Componentes intermedios o contenedores, no reutilizables.

ID	Componente	Responsabilidad
1	Contenedor de aplicación	Estructura básica de la aplicación (cabecera, pie, barra lateral).
2	Bienvenida	Entrada a la aplicación con ayuda y descripción general.
3	Contenedor de modelo	Contiene los componentes de edición del modelo.
4	Contenedor de importación/exportación	Contiene los componentes de importación y exportación de datos.
5	Contenedor de consultas SPARQL	Contiene los componentes de ejecución y visualización de resultados de consultas SPARQL.
6	Contenedor de actividades	Contiene los componentes de carga y visualización de actividades.
7	Listado de recursos	Componente reutilizable para listar recursos en una tabla de datos.
8	Detalle de recursos	Componente reutilizable para visualizar detalles de recursos.
9	Lanzador de consultas SPARQL	Componente reutilizable para lanzar consultas SPARQL
10	Listado de resultados de consultas SPARQL	Componente reutilizable para mostrar resultados de una consulta SPARQL.
11	Cargador de archivos	Componente reutilizable para importar datos desde archivo.
12	Cargador de URLs	Componente reutilizable para importar datos desde URL.
13	Cargador de Vocabularios	Componente reutilizable para habilitar o deshabilitar vocabularios.
14	Visor de markdown	Componente reutilizable para visualizar markdown renderizado en HTML.

Cuadro 7.1: Listado de componentes de la aplicación.

- Componentes finales reutilizables.

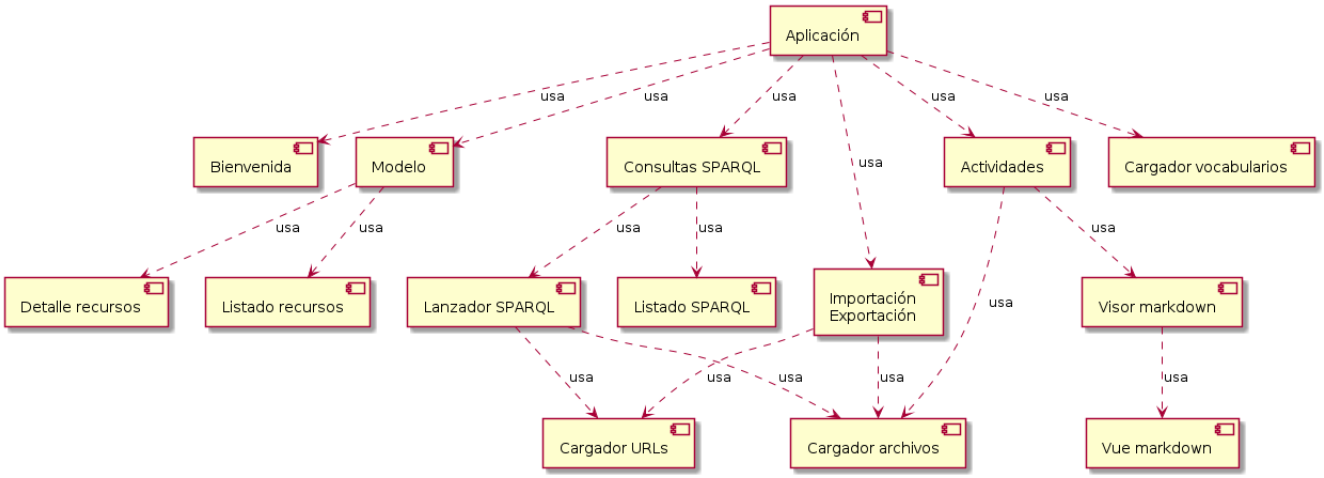

Figura 7.3: Jerarquía de componentes de la aplicación.

Capítulo 8

Implementación y pruebas

<....>

Capítulo 9

Resultados

<....>

Capítulo 10

Conclusiones y trabajos futuros

10.1. Conclusiones

<....>

10.2. Trabajos futuros

<....>

Bibliografía

[1] TechMagic. Reactjs vs angular4 vs vue.js, what to choose in 2018, 2018.

[2] Communication from the comission to the european parliament, the council, the european economic and social committee and the committee of the regions., 2018.

[3] Tim Berners-Lee. The semantic web. *Scientific American, Inc.*, 2001.

[4] Tim Berners-Lee. Linked data. *W3C*, 2006.

[5] Data never sleeps 5.0, 2017.

[6] Ruben Verborgh and Thomas Bergwinkl. Rdf javascript libraries community group, 2018.

[7] Ora Lassila. Resource description framework (RDF) model and syntax specification. W3C recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[8] SemStats. 2018.

[9] Apache Software Foundation. Apache jena.

[10] Colin Evans and Jamie Taylor. *Programming the Semantic Web*. 2009.

[11] Varios. Comparison of rdfjs libraries.

[12] JD Isaacks. *Get Programming With Javascript Next*. 2018.

[13] Stephen Grider. Es6 javascript: The complete developer's guide.

[14] Stone River eLearning. Working with javascript streams, 2018.

[15] Maximilian SchwarzmÃŒller. Vue js 2 - the complete guide, 2018.

[16] Anthony Gore. The ultimate vue js 2 developers course, 2018.

[17] Paul. Halliday. *Vue.js 2 Design Patterns and Best Practices*. 2018.

[18] Winston W. Royce. Managing the development of large software systems. *IEEE WESCON*, 1970.

[19] Craig Larman. *Applying UML and patterns*. 2005.

[20] The Standish Group. Chaos report, 2015.

[21] Scott W. Ambler. The non-existent software crisis: Debunking the chaos report. *Dr. Dobb's*, 2014.

[22] Lars Mierizt. Gartnet survey shows why projects fail, 2012.

[23] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. 2005.

[24] Thomas Bergwinkl, Michael Luggen, elf Pavlik, Blake Regalia, Piero Savastano, and Ruben Verborgh. Interface specification: Rdf representation, 2017.

[25] Ruben Taelman and Joachim Van Herwegen. Comunica query engine platform, 2018.

[26] Alistair Cockburn. *Writing Effective Use Cases*. 2001.

[27] Rational. Rational unified process: Best practices for software development teams. 1998.

[28] Bob DuCharme. *Learning SPARQL*. 2013.

[29] Liyang Yu. *A Develope's Guidfe to the Semantic Web*. 2015.

[30] What is vuex?, 2018.

Anexo A

<Título Anexo A>

<...>

A.1. <Primera sección anexo>

A.1.1. <Primera subsección anexo>