eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Sistema visual de consulta y gestión de datos provenientes de máquinas extrusoras utilizando técnicas semánticas

Autora

Irati Galarza Burguete

2018

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Sistema visual de consulta y gestión de datos provenientes de máquinas extrusoras utilizando técnicas semánticas

Autora

Irati Galarza Burguete

Directoras

Arantza Illarramendi, Idoia Berges

Agradecimientos

Antes que nada, me gustaría agradecer a mis tutoras, Arantza Illarramendi e Idoia Berges, por haber confiado en mi para la realización de este proyecto y por lo bien que han sabido guiarme durante este largo proceso que finalmente ha llegado a su fin. Gracias a su ayuda y consejos he sido capaz de completar este TFG, aportándome conocimientos fundamentales que me servirán durante todo mi desarrollo profesional.

También quiero agradecer al grupo de investigación BDI por haberme acogido tan bien durante un tiempo en su lugar de trabajo y por haberme resuelto tan amablemente todas las dudas que me iban surgiendo. En especial, quiero agradecer a Borja Díez por toda la ayuda y orientación que me ha ofrecido durante el desarrollo de este proyecto, gran parte de lo conseguido ha sido también gracias a él.

Agradecer también a las empresas Urola Solutions y Savvy por haber hecho posible que este proyecto se llevase a cabo aportando los datos e información necesaria y atendiendo siempre a las cuestiones surgidas.

Por otro lado, me gustaría agradecer a mi familia por todo el apoyo que me han ofrecido, no solo durante la realización de este TFG, si no a lo largo de mis ya 22 años. En especial, quiero agradecer a mis padres y a mi hermano, quienes siempre han creído en mi y quienes han sabido aguantarme y animarme hasta en mis momentos más difíciles.

Gracias también a mis amigas, a mis ”*lagunas*”, por estar siempre ahí cuando las he necesitado, por ayudarme a desconectar, y por sacarme una sonrisa incluso en los días en los que el agobio estaba pudiendo conmigo.

Por último pero no menos importante, me gustaría también agradecer a mis compañeros y amigos de clase, quienes me han ayudado siempre que han podido y con quienes he tenido el placer de compartir estos cuatro años de trayecto universitario. Esta experiencia no hubiese sido la misma sin ellos. En especial, le quiero agradecer a Ania por todo el apoyo y los ánimos que siempre me ha dado, por haber creído en mi cuando yo no era capaz de hacerlo y, en definitiva, por haberse convertido en mi nueva hermana, en mi ”*siamesa*”, en tan solo cuatro años.

Resumen

Ese Trabajo de Fin de Grado se ha desarrollado dentro del marco de un proyecto real de mayor magnitud basado en conceptos de la Industria 4.0. Concretamente, en este TFG se presenta, a modo de prueba de concepto, un sistema visual de consulta y gestión de datos utilizando técnicas semánticas. Dichos datos provienen de una máquina extrusora de polímeros y han sido obtenidos a partir de las empresas colaboradoras del proyecto real en el que se enmarca.

En primer lugar se ha desarrollado una ontología que describe y modela los sensores de la máquina extrusora en cuestión. Después, en base a dicha ontología, se ha creado un módulo para el almacenamiento de los datos de los sensores en formato semántico y otro módulo para la consulta, análisis y visualización de los datos almacenados.

Por último, se ha analizado el sistema desarrollado y se han planteado posibles mejoras del mismo, tanto a corto como a largo plazo.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

El frenético y desmesurado desarrollo de las tecnologías de la información vivido en los últimos años ha tenido un impacto directo en el avance de los métodos de tratamiento y extracción de grandes cantidades de datos, también conocidos como *Big Data*, y los cuales no pueden ser tratadas con los sistemas de información tradicionales.

Uno de los sectores más beneficiados del desarrollo de estos métodos es el sector industrial, cuyo cambio ha sido tan significativo que se han acuñado nuevos términos como la *cuarta revolución industrial* o la *Industria 4.0* en relación con el mismo. El término Industria 4.0 hace referencia a un amplio rango de conceptos fundamentales que describen diferentes cambios, principalmente impulsados por las tecnologías de la información, en los sistemas de producción. Este desarrollo no tiene solamente implicaciones tecnológicas, sino que también tiene implicaciones organizativas, provocando un cambio de concepto en la orientación de las empresas. [Lasi et al., 2014]

Por otro lado, el contenido accesible en la *web* de hoy en día también aumenta de manera desmesurada. La web está mayormente formada por una gran cantidad de hipertexto entendible para los humanos pero que carece de significado para los diferentes sistemas computacionales que pudieran acceder a la misma. El término de la *Web Semántica* aparece como solución a esto último, y describe la evolución de una web compuesta por información para que los humanos la lean e interpreten a una web que incluya datos e información que los sistemas computacionales puedan entender y manipular, lo que además

favorece la integración e interoperabilidad de los datos. [Shadbolt et al., 2006]

En este Trabajo de Fin de Grado se busca una unión entre ambos conceptos, intentando determinar los beneficios que aporta la utilización de técnicas semánticas en un entorno de Industria 4.0, en el que el análisis y almacenamiento de grandes cantidades de datos es fundamental. Dotar de significado a estos datos podría impactar en una mejora notable en los procesos llevados a cabo por los sistemas computacionales.

CAPÍTULO 2

Antecedentes

Este Trabajo de Fin de Grado (TFG en adelante) se enmarca en un proyecto real de Industria 4.0 en el que participan distintas empresas colaborando con un objetivo final de mejora y optimización de un proceso concreto: la fabricación de botellas de plástico.

En este capítulo se presentan dichas empresas así como su rol en el proyecto comentado, definiendo así los antecedentes de este TFG concreto y mostrando una idea general de la situación en la que se basa el mismo.

Además, el grupo de investigación BDI[1] de la Universidad del País Vasco (UPV/EHU)[2] está desarrollando, en el mismo escenario industrial, una plataforma web actualmente enfocada a los ingenierios de datos proporcionando funcionalidades para el desarrollo de tareas de preprocesado de los datos.

Las funcionalidades desarrolladas en este TFG tienen también como objetivo formar parte de un módulo adicional de dicho sistema web, por lo que en este capítulo se presenta el estado actual de dicho sistema web y su relación con el trabajo a desarrollar.

Por último, como ya se ha mencionado en la introducción, este proyecto se plantea como una prueba de concepto, por lo que para identificar su interés es importante tener una visión clara de otros trabajos similares que hayan sido realizados con un propósito seme-

[1] Grupo de investigación BDI: http://bdi.si.ehu.es/bdi/
[2] UPV/EHU: https://www.ehu.eus/es/home

jante al de este TFG: el desarrollo de una solución visual para el análisis de datos en un entorno de Industria 4.0 basado en técnicas semánticas.

Por ello, en este capítulo también se presentan los trabajos similares encontrados y su relación con este TFG.

2.1. Empresas involucradas

El proyecto real definido cuenta con la colaboración de distintas empresas:

- Fábrica de Jabón La Corona y Bowler Plastics (Pty) Ltd, las empresas encargadas de producir las botellas de plástico.
- Urola Solutions, la empresa encargada de producir las máquinas industriales que son utilizadas para la creación de las botellas de plástico.
- Savvy Data Systems, la empresa que proporciona los servicios informáticos para la explotación de los datos recogidos en las máquinas industriales.

A continuación se detalla individualmente el papel que juega cada empresa en dicha colaboración así como el impacto que tienen las mismas en el TFG presentado.

2.1.1. Savvy Data Systems

Savvy Data Systems[3] tiene un área de negocio basado en el concepto de Industria 4.0, y los servicios que ofrece están dirigidos a la explotación de datos industriales mediante sistemas avanzados de información. Para conseguir esto es necesario un proceso de digitalización previo, después del cual la empresa realiza la explotación de los datos obtenidos mediante dos enfoques principales: *Business Intelligence* y *Machine Learning*.

Respecto a este TFG, es especialmente relevante el enfoque de *Business Intelligence*. El objetivo de este enfoque es poder proporcionar recursos que permitan a los expertos de un dominio concreto analizar sus datos de una manera eficiente e inteligente, buscando que los resultados obtenidos puedan ser comprendidos y analizados por personas sin conocimientos informáticos. En este TFG se presenta un sistema web alternativo siguiendo el mismo enfoque pero basado en técnicas semánticas para su implementación, destacando las propiedades y el potencial de las mismas. Además también se pretende destacar las

[3] Savvy Data Systems: http://www.savvydatasystems.com/

posibilidades de personalización a la hora de formular las consultas de datos, ofreciendo una mayor libertad de análisis a los expertos de dominio.

2.1.2. Urola Solutions

Urola Solutions[4] es la empresa encargada de construir y comercializar la maquinaria industrial utilizada para el proceso de fabricación de botellas. Estas máquinas realizan un proceso denominado extrusión, por lo que son también conocidas como extrusoras.

En dichas máquinas, se incorporan sensores para la captura de diferentes fenómenos, señales o estados del proceso: las temperaturas en las zonas principales, la velocidad del motor, el estado de elementos mecánicos, etc.

En cuanto al TFG a desarrollar, los datos proporcionados por estos sensores serán el objeto a analizar, de manera que las máquinas extrusoras diseñadas por Urola deberán ser correctamente analizadas para el posterior almacenamiento y análisis de los datos.

Para simplificar el escenario de cara a la prueba de concepto, el análisis del dominio se basará principalmente en un tipo de máquina extrusora: la Extrusora de Cuatro Zonas (Fig. 2.1).

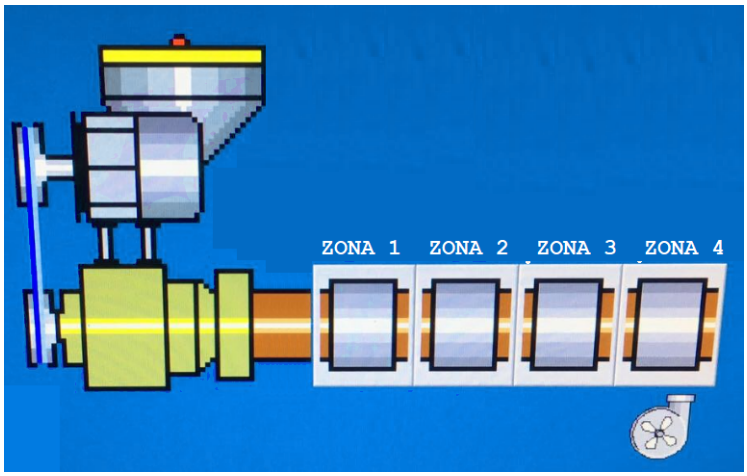

Figura 2.1: Imagen simplificada de la *Extrusora de Cuatro Zonas* de Urola Solutions

[4]Urola Solutions: http://www.urolasolutions.com/es/

2.1.3. Fábrica de Jabón La Corona y Bowler Plastics (Pty) Ltd.

La Fábrica de Jabón La Corona[5], situada en México, y Bowler Plastics (Pty) Ltd.[6], situada en Sudáfrica, serán las empresas encargadas de producir las botellas de plástico mediante la maquinaria proporcionada por Urola, por lo que serán las empresas que proporcionen los datos de las máquinas y sus sensores.

Éstos serán los datos con los que se trabajará en el TFG, de manera que se mostrarán situaciones y escenarios reales.

2.2. Sistema web en desarrollo: I4TSPS

En el mismo escenario y con la misma colaboración entre empresas, el grupo de investigación BDI está desarrollando la plataforma web *I4TSPS*[7] (Fig. 2.2), destinada a los ingenieros de datos y que proporciona distintas funcionalidades y herramientas en relación con el preprocesamiento de los datos.

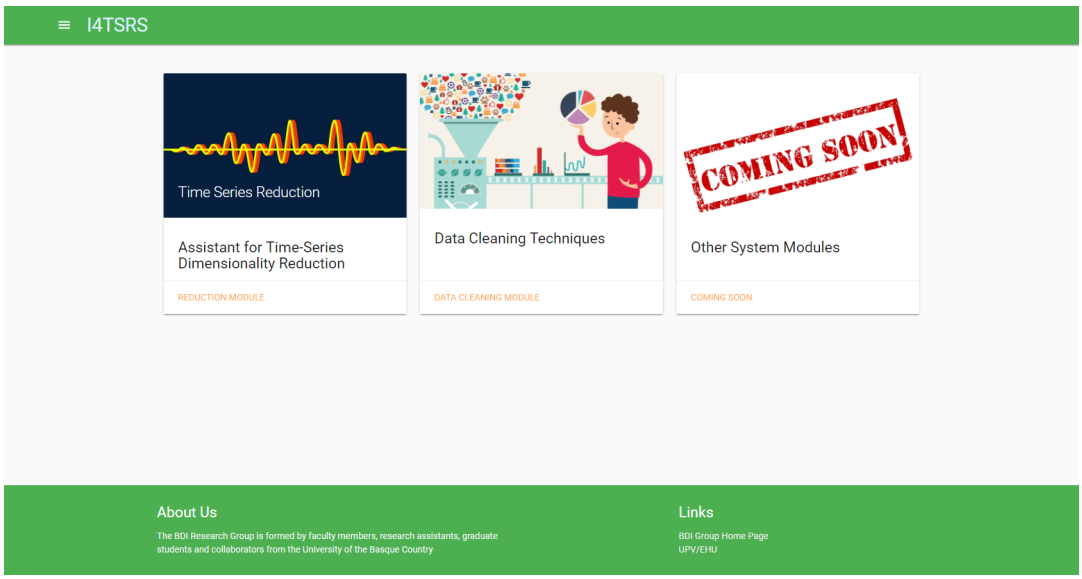

Figura 2.2: Captura de la página principal de I4TSRS

Por un lado, ofrece un asistente para la reducción de series de datos (Fig 2.3), recomendando distintas técnicas de reducción dependiendo de las características específicas de cada serie y permitiendo al ingeniero de datos aplicar dichas reducciones.

[5]Fábrica de Jabón La Corona: https://www.lacorona.com.mx/
[6]Bowler Plastics (Pty) Ltd.: http://www.bowler.co.za/
[7]I4TSPS: https://fdai-b5221.firebaseapp.com

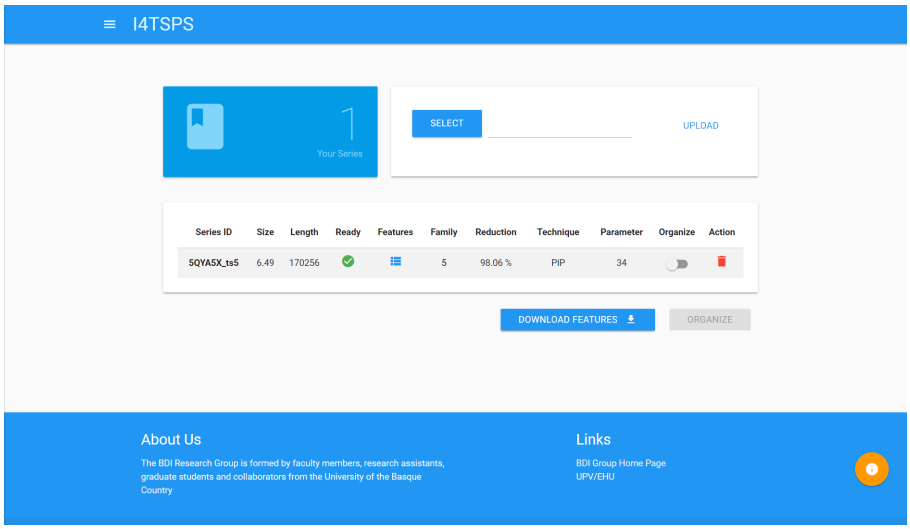

Figura 2.3: Captura de la página principal del asistente de reducción de series de I4TSRS

Por otro lado, también ofrece un catálogo de técnicas de limpieza de datos (Fig. 2.4) para aplicar a las series, de manera que el ingeniero de datos pueda probar las distintas técnicas y decidir la que más se ajuste a sus necesidades.

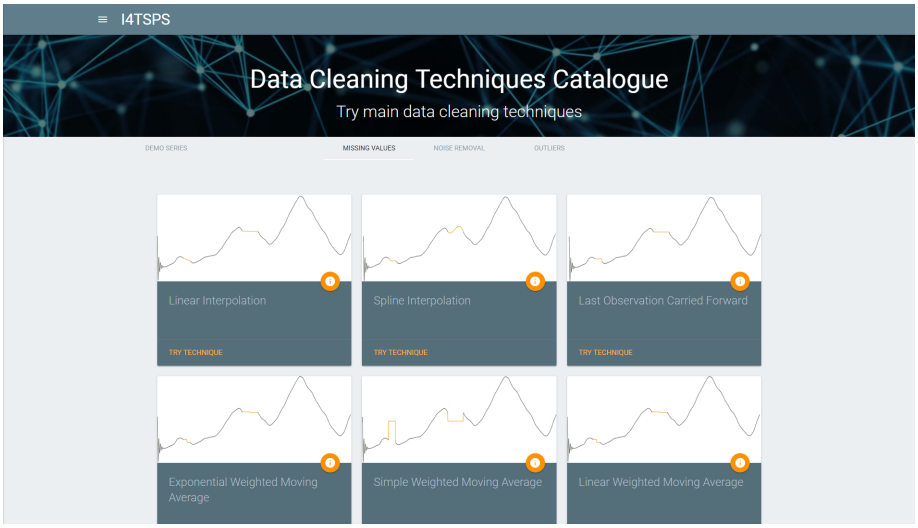

Figura 2.4: Captura del catálogo de técnicas de limpieza de datos de I4TSRS

Con la intención de ampliar el rango de funcionalidades y soluciones ofrecidas por este sistema web, se pretende añadir al mismo el sistema web resultante del TFG actual, proporcionando funciones de análisis e inserción de las series de datos provenientes de las extrusoras.

2.3. Trabajos relacionados

En cuanto a trabajos similares a lo presentado en este TFG, no ha sido encontrada ninguna aplicación que ofrezca al usuario un sistema visual de consulta y análisis de datos basado en técnicas semánticas para un ámbito de Industria 4.0.

Sin embargo, sí que se pueden encontrar diferentes trabajos y artículos relacionados con algunos de los diferentes conceptos abordados en este TFG. Dichos trabajos realizan distintos acercamientos sobre la Industria 4.0, las técnicas semánticas y/o los sistemas de consulta visuales.

2.3.1. Trabajos sobre ontologías relacionadas con la industria

En relación al ámbito de la Web Semántica, se pueden encontrar diferentes ontologías ya definidas y de interés en el mundo industrial.

Entre éstas destaca la ontología *Semantic Sensor Network (SSN)*[8][Compton et al., 2012]. La SSN es una ontología desarrollada por el *WorldWide Web Consortium (W3C)*[9] y que permite, entre otras, describir los diferentes sensores que pueden tener las máquinas así como las observaciones y resultados que se obtienen de los mismos.

Por otro lado, en el artículo *Converting the Industry Foundation Classes to the Web Ontology Language* [Schevers and Drogemuller, 2005] se presenta el desarrollo de una ontología que describe el modelo *Industry Foundation Classes*. Éste es un modelo de datos estándar utilizado en la industria de la construcción, el cual define las características de los datos relacionados con el diseño, construcción, mantenimiento y operación de obras civiles. En el artículo mencionado se abarca la representación semántica de dicho modelo, permitiendo así el uso del mismo en conjunto con diferentes técnicas semánticas.

Con otro enfoque distinto, en el artículo *An organisation ontology for enterprise modeling: Preliminary concepts for linking structure and behaviour* [Fox et al., 1995] se presenta una ontología que describe una organización industrial, permitiendo definir la estructura de la misma. A diferencia de la mayoría de ontologías relacionadas con la industria, las cuales se centran en los productos desarrollados o procesos llevados a cabo en las diferentes empresas, la ontología desarrollada en este artículo se centra en la organización interna de la propia empresa.

[8] SSN: https://www.w3.org/TR/vocab-ssn/
[9] W3C: https://www.w3c.es/

2.3.2. Trabajos sobre técnicas semánticas en la Industria 4.0

Se han desarrollado distintos trabajos en los que se pretende diseñar una ontología eficiente para su uso en el ámbito de la Industria 4.0, proponiendo diferentes soluciones a problemas aparecidos en el mismo y adecuando las ontologías a escenarios concretos.

Un ejemplo de esto es el artículo *Big Data Semantics in Industry 4.0* [Obitko and Jirkovský, 2015], donde sus autores proponen una solución semántica para poder manejar el *Big Data* que comúnmente nos encontramos en los escenarios de Industria 4.0, dando pie a una futura explotación y análisis de los mismos.

Como otro ejemplo relacionado se puede mencionar el artículo *Towards a Semantic Administrative Shell for Industry 4.0 Components* [Grangel-González et al., 2016], en el cual se ha diseñado una ontología para la representación de un panel de administración de componentes relacionados con la Industria 4.0. De esta manera, gracias al uso de las tecnologías semánticas, pretenden mejorar la comunicación y comprensión entre los diferentes componentes.

De la misma manera, en el artículo *Manufacturing Ontology Development based on Industry 4.0 Demonstration Production Line* [Cheng et al., 2016] se desarrolla una ontología para la representación de la línea de producción en un ámbito de Industria 4.0. Dicha ontología engloba todos los aspectos de los productos de manera genérica, desde la personalización de un pedido hasta la producción resultante del mismo.

2.3.3. Trabajos sobre sistemas de consulta visuales (*VQS*) en un entorno semántico

Por último, también se han hecho diferentes acercamientos sobre sistemas de consulta visuales o *Visual Query Systems (VQS)* basados en técnicas semánticas, en los que se propone una solución visual a la realización de consultas *SPARQL*[10] para la extracción de datos concretos.

Un ejemplo relevante relacionado es la herramienta *OptiqueVQS* [Soylu et al., 2015]. Esta herramienta propone un sistema visual de consultas para un sistema basado en ontologías y hace una valoración sobre los efectos que el *Big Data* o las grandes cantidades de datos tendrían sobre el mismo, siendo éste un concepto cercano a los ámbitos de la Industria 4.0 como ya se ha comentado anteriormente.

[10] *SPARQL* es el lenguaje de consulta estándar para la información representada mediante *Resource Description Framework (RDF)*, modelo base y estandarizado de la Web Semántica, ambos creados por el W3C.

Otro ejemplo de esto es la herramienta presentada en el artículo *PepeSearch: Semantic Data for the Masses* [Vega-Gorgojo et al., 2016a]. Dicha herramienta también ofrece una solución visual para la consulta de datos semánticos, de manera que los usuarios no necesitan tener conocimientos sobre SPARQL ni RDF para formular dichas consultas. En el mismo artículo también se hace mención de las distintas evaluaciones que han sido realizadas sobre la herramienta con usuarios inexpertos, cuyos resultados fueron altamente satisfactorios.

Relacionado con las herramientas presentadas en ambos artículos, en el artículo *Visual query interfaces for semantic datasets: An evaluation study* [Vega-Gorgojo et al., 2016b] se hace un estudio sobre las interfaces de consulta visuales para datos semánticos. Para ello, en el artículo se realiza una comparación entre las herramientas *OptiqueVQS* y *PepeSearch*, las cuales están desarrolladas siguiendo dos enfoques distintos para la realización de las consultas.

Por otro lado, en el artículo *How Useful Are Natural Language Interfaces to the Semantic Web for Casual End-Users?* [Kaufmann and Bernstein, 2007] se evalúa el grado de utilidad de las interfaces basadas en el lenguaje natural para la consulta de datos semánticos. Para ello, comparan cuatro herramientas de consulta desarrolladas con distintos enfoques y se analizan las pruebas realizadas sobre las mismas con un amplio número de usuarios inexpertos.

En este TFG concretamente, creando una unión entre los diferentes conceptos y enfoques comentados, se exploran los sistemas visuales de consulta en un entorno semántico y dentro de un marco de Industria 4.0.

CAPÍTULO 3

Planteamiento Inicial

En este capítulo se presenta el planteamiento inicial pensado para el correcto desarrollo del proyecto, describiendo y analizando los objetivos del mismo, definiendo un alcance adecuado y explicando la planificación resultante.

También se exponen las herramientas necesarias para su desarrollo y se hace una evaluación de los posibles riesgos que se prevén en la realización del proyecto, detallando además el plan de mitigación desarrollado para los mismos.

3.1. Descripción del producto

El proyecto a realizar tiene como fin el desarrollo de una prueba de concepto que demuestre el potencial de la utilización de técnicas semánticas en un entorno de Industria 4.0. Con este fin, se desarrollará un sistema web que permita a un responsable de planta de fabricación consultar, a nivel semántico, los datos generados por sensores industriales.

Para ello, el sistema web ofrecerá una manera visual e intuitiva para la formulación de la consulta que se desee realizar, evitando de esta manera la necesidad de tener conocimientos informáticos para la extracción de los datos. La información resultante de la consulta se mostrará de forma gráfica para una más fácil y directa comprensión de los resultados.

El sistema también ofrecerá la posibilidad de inserción de nuevos datos a partir de archi-

vos que el responsable de planta podrá proporcionar, de manera que desde una misma aplicación serán posibles tanto el análisis como el almacenamiento de los datos.

3.2. Objetivos

El objetivo principal de este proyecto consiste en la **creación de un Sistema Web** que permita a un responsable de planta de fabricación la consulta de datos generados por sensores industriales. Para ello, se analizarán distintas tecnologías semánticas de anotación de datos y la información de los datos se representará mediante distintas gráficas.

Entre los objetivos secundarios, destaca el aprendizaje de la **utilización de tecnologías semánticas en un entorno de Big Data**, comprobando además el potencial de este tipo de técnicas para el ámbito de análisis de datos. El campo de la llamada *Web Semántica* tiene cada vez más relevancia en el mundo de la informática, especialmente en entornos en los que obtener información de los datos almacenados es uno de los principales objetivos. Dotando a los datos de una semántica se facilitan la identificación de datos con un mayor significado y la inferencia de las relaciones entre los mismos, consiguiendo una extracción de información más relevante. Por lo que, en un momento en el que la demanda de este tipo de tecnologías es cada vez mayor, tener un conocimiento sobre el funcionamiento de las mismas puede ser un punto muy relevante en la formación de un Ingeniero Informático.

Como objetivo secundario también se encuentra el aprendizaje del desarrollo de Sistemas Web mediante una herramienta como **React.js**, que basa su funcionamiento en las **SPA (*Single-Page Applications*** o Aplicaciones de una sola página). Este tipo de aplicaciones interactúan con el usuario reescribiendo dinámicamente el contenido de la página actual, evitando cargar la página entera desde un servidor. Este enfoque mejora la experiencia de los usuarios en la aplicación, evitando las interrupciones en la carga de sucesivas páginas. Estas técnicas están siendo cada vez más utilizadas, teniendo así mismo cada vez más relevancia las librerías y frameworks de JavaScript que las implementan, por lo que trabajar con una de ellas aportará un conocimiento muy relevante de cara al futuro.

Otro de los objetivos secundarios es **conocer y utilizar un lenguaje de diseño**, los cuales están enfocados a mejorar la usabilidad de las aplicaciones de cara a los usuarios finales. En este caso, se utilizará el llamado ***Material Design***, un lenguaje de diseño desarrollado por Google y que intenta describir y unificar ciertos conceptos de diseño de interfaces para mejorar la experiencia del usuario a través de todo tipo de dispositivos, plataformas y métodos de entrada. Asegurar una buena experiencia de usuario es un punto cada vez

más importante en el desarrollo de sistemas web, por lo que conocer y entender los conceptos de un lenguaje de diseño que se centra en esto puede ser de gran beneficio para la formación personal.

Por último, pero no menos importante, otro de los objetivos secundarios es demostrar la capacidad de **realizar y gestionar un proyecto de este nivel de principio a fin** de manera mayormente individual. Planificar, organizar y desarrollar un proyecto de estas características demuestra la autosuficiencia y capacidad de aprendizaje adquirida durante los cuatro años de carrera, características más que críticas para el paso al mundo laboral.

3.3. Alcance

Para el desarrollo de una buena planificación, es importante definir correctamente el alcance del proyecto a realizar, identificando los procesos necesarios para que el proyecto incluya todo el trabajo requerido para completarlo con éxito.

3.3.1. Exclusiones

Con motivo de la ya comentada futura unión del TFG con el sistema web I4TSPS, se excluyen del alcance del proyecto la implementación de las funcionalidades correspondientes al control de accesos de la aplicación. Con esto se hace referencia al registro, inicio y cierre de sesión de los diferentes usuarios de la aplicación final, y para ello se utilizarán las funcionalidades ya desarrolladas por el grupo de investigación BDI.

Sin embargo, el sistema web sí que será desarrollado con una posible distinción de roles de usuarios en la misma, solo que no proporcionará la funcionalidad de gestión de los mismos.

3.3.2. Ciclo de vida

Para el desarrollo de este proyecto no se ha optado por adoptar una metodología ágil concreta, pero sí que se ha acordado sin embargo organizar el trabajo de manera que se pueda ir observando el resultado de manera incremental y continua.

Por ello, se ha adoptado un ciclo de vida incremental (Fig.3.1), creando distintas versiones del producto final y de manera que las nuevas versiones aporten nuevas funcionalidades y mejoras con respecto a las versiones anteriores.

Este ciclo de vida permite así ir alcanzando determinados objetivos concretos listos para probar y mostrar antes de finalizar el proyecto completo. De esta manera, se podrán discutir y analizar las distintas versiones generadas para decidir los cambios que sean necesarios, facilitando así el proceso de desarrollo del producto y permitiendo además poder mostrar los avances del mismo de una manera clara y concreta.

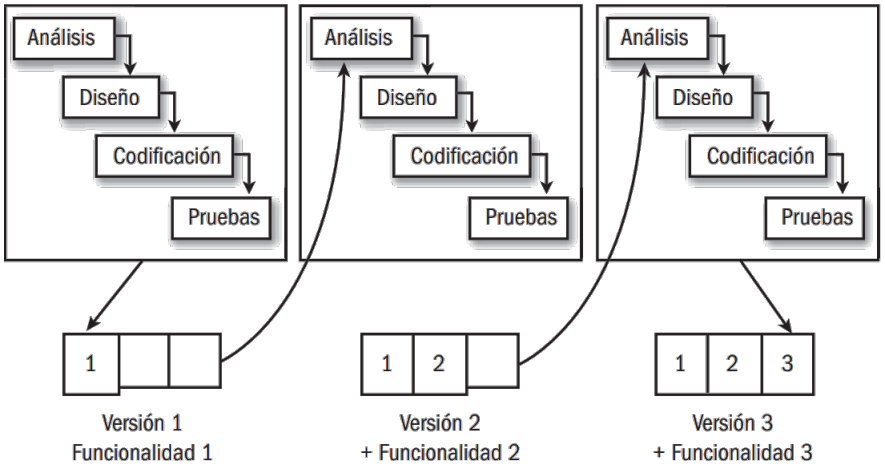

Figura 3.1: Ciclo de vida Incremental

3.3.3. Estructura de Descomposición del Trabajo

Definir una Estructura de Descomposición del Trabajo (E.D.T., Fig. 3.2) es importante en toda definición del alcance de un proyecto. Mediante el E.D.T. se pretende subdividir el trabajo del proyecto en componentes más pequeños y más fáciles de manejar.

En este caso, en el primer nivel se han identificado las distintas *fases* que deberá tener el proyecto, creando así 7 grupos diferentes. En el segundo nivel, se han identificado los paquetes de trabajo específicos necesarios para completar el proyecto, guiados por los diferentes módulos que compondrán el resultado final.

Estos paquetes de trabajo a su vez, estarán compuestos por un número de tareas específicas necesarias para completarlos.

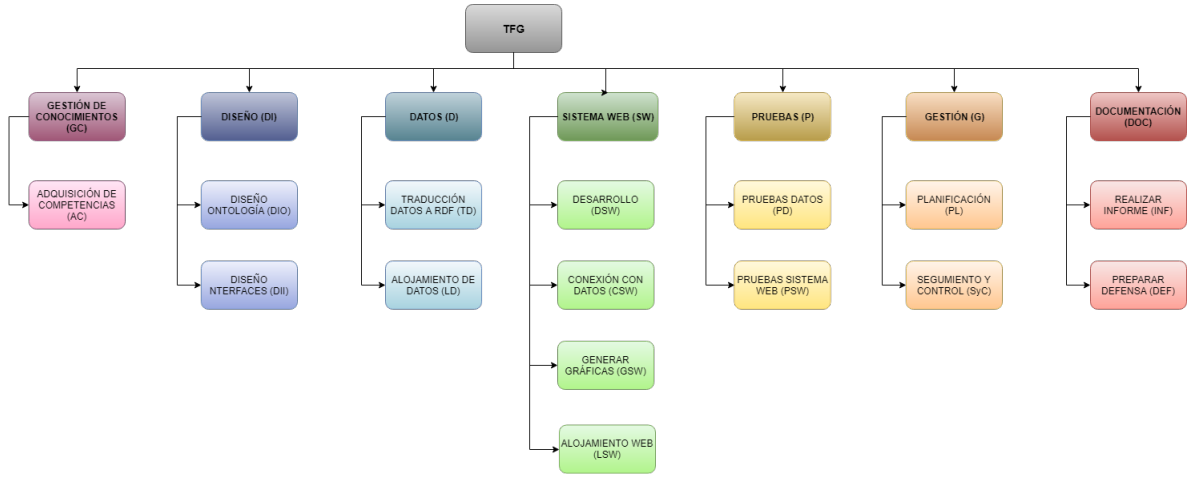

Figura 3.2: E.D.T. Inicial del Proyecto

A continuación se describen todos los paquetes de trabajo y se detallan sus tareas explicando brevemente la finalidad de cada una.

Fase del Gestión del Conocimiento (GC):

El paquete de trabajo ***Adquisición de Competencias (AC)*** agrupará las tareas necesarias para la adquisición de las competencias requeridas para el correcto desarrollo del proyecto. Dichas tareas son:

- AC.1: Conocer y alcanzar un conocimiento intermedio sobre las ontologías y la web semántica, entendiendo el formato RDF para la representación de la información y el lenguaje SPARQL para la consulta de datos.
- AC.2: Conocer y alcanzar un conocimiento intermedio sobre la utilización del programa Protégé, herramienta que se utilizará para la creación de la ontología propia del proyecto.
- AC.3: Conocer y alcanzar un conocimiento alto sobre las herramientas React.js y MaterializeCss, imprescindibles para el desarrollo principal del Sistema Web.
- AC.4: Conocer y alcanzar un conocimiento intermedio sobre Firebase, la plataforma que se encargará del alojamiento el Sistema Web final.
- AC.5: Conocer y alcanzar un conocimiento intermedio sobre OpenLink Virtuoso, herramienta que permitirá el alojamiento de los datos en formato RDF.

- AC.6: Conocer y alcanzar un conocimiento intermedio sobre el servicio Gráficas de Google, que se encargará de proporcionar la visualización de los datos que el usuario consulte.
- AC.7: Conocer y alcanzar conocimientos básicos sobre las herramientas secundarias necesarias para desarrollar el proyecto, ya sean librerías o programas concretos para utilizaciones puntuales.

Fase de Diseño:

El paquete de trabajo ***Diseño Ontología (DIO)*** agrupará las tareas necesarias para el desarrollo de una ontología que permita describir los datos capturados por los sensores de la *Extrusora de Cuatro Zonas* de la empresa Urola Solutions. Dichas tareas son:

- DIO.1: Desarrollar una primera versión de la ontología con la información inicial básica de los sensores que se tienen en la extrusora, siguiendo la metodología NeOn para ello.
- DIO.2: Mejorar y completar la ontología básica con conocimiento relevante que se vaya obteniendo sobre los sensores.

El paquete de trabajo ***Diseño Interfaces (DII)*** agrupará las tareas necesarias para definir el diseño que tendrá el Sistema Web resultante en el proyecto. Dichas tareas son:

- DII.1: Crear una versión inicial de las interfaces del Sistema Web, pensando en las funcionalidades básicas que tendrá.
- DII.2: Mejorar y completar la versión inicial de las interfaces, solucionando posibles errores de diseño o incluyendo en las mismas cambios acordados con las tutoras.

Fase de Datos:

El paquete de trabajo ***Traducción Datos a RDF (TD)*** agrupará las tareas necesarias para desarrollar el módulo encargado de traducir los datos de los sensores, proporcionados en formato CSV, a un formato RDF de acuerdo con la ontología desarrollada previamente. Dichas tareas son:

- TD.1: Entender cómo representar los datos en formato RDF, concretamente, en formato Turtle (extensión .ttl), y definir la estructura que deberán tener los mismos.
- TD.2: Crear un módulo para la traducción de los datos recibidos en formato CSV al formato Turtle siguiendo el esquema de la estructura en RDF desarrollado en TD.1.

El paquete de trabajo ***Alojamiento de Datos (LD)*** agrupará las tareas necesarias para desarrollar el módulo encargado de alojar los datos en formato RDF en el repositorio de Virtuoso adecuado. Dichas tareas son:

- LD.1: Preparar y configurar el entorno, Virtuoso más concretamente, para el alojamiento de los datos necesarios.
- LD.2: Implementar la automatización del proceso de alojamiento de los datos.

Fase de Sistema Web:

El paquete de trabajo ***Desarrollo del Sistema Web (DSW)*** agrupará las tareas necesarias para el desarrollo de la estructura principal del sistema web, desarrollado con React.js y MaterializeCss. Dichas tareas son:

- DSW.1: Desarrollo inicial de las interfaces del sistema con React.js y MaterializeCss.
- DSW.2: Implementar los cambios y mejoras necesarias en las distintas interfaces del Sistema Web, ya sea para solucionar ciertos errores o por cambios en el diseño de las mismas.

El paquete de trabajo ***Conexión con Datos (CSW)*** agrupará las tareas necesarias para llevar a cabo la conexión con los datos alojados previamente. Para ello, será necesario generar preguntas SPARQL, acceder a los datos y traducir la respuesta para poder manejarla. Dichas tareas son:

- CSW.1: Implementar la generación de distintas preguntas SPARQL a partir de cierta información recibida a través de formularios en el Sistema Web.
- CSW.2: Implementar la conexión con Virtuoso para enviar la pregunta SPARQL generada.

- CSW.3: Implementar la recogida y tratamiento de los datos recibidos.

El paquete de trabajo ***Generar Gráficas (GSW)*** agrupará las tareas necesarias para analizar los datos recogidos y generar gráficas a partir de ellos con la ayuda de los gráficos de Google. Dichas tareas son:

- GSW.1: Añadir los datos recibidos desde Virtuoso a las Gráficas de Google para visualizar las consultas.
- GSW.2: Implementar el uso de distintas opciones y características de las gráficas dependiendo del tipo de preguntas realizadas.

El paquete de trabajo ***Alojamiento Web (LSW)*** agrupará las tareas necesarias para alojar el sistema web desarrollado en Firebase. Dichas tareas son:

- LSW.1: Configurar Firebase y el proyecto para el correcto alojamiento del Sistema Web.
- LSW.2: Alojamiento final y continuo del proyecto, solucionando posibles errores que aparezcan una vez esté en línea.

Fase de Pruebas:

El paquete de trabajo ***Pruebas Datos (PD)*** agrupará todas las tareas necesarias para llevar a cabo las pruebas necesarias sobre los módulos relacionados con los datos. Se probará su correcta traducción a RDF y su correcto alojamiento. Dichas tareas son:

- PD.1: Verificación de que la traducción de los datos al formato Turtle es correcta.
- PD.2: Verificación de que el alojamiento automático de los datos funciona correctamente.

El paquete de trabajo ***Pruebas Sistema Web (PSW)*** agrupará todas las tareas necesarias para llevar a cabo las pruebas sobre el sistema web desarrollado y su correcto funcionamiento. Dichas tareas son:

- PSW.1: Verificación de que las interfaces creadas para el Sistema Web funcionan correctamente.

- PSW.2: Verificación del correcto funcionamiento de la generación de las consultas SPARQL a partir de los formularios.
- PSW.3: Verificación de que la conexión con Virtuoso es correcta.
- PSW.4: Verificación de que las gráficas se muestran correctamente, correspondiendo con los datos iniciales.
- PSW.5: Verificación de que el Sistema Web funciona correctamente después de su alojamiento en Firebase.

Fase de Gestión:

El paquete de trabajo de ***Planificación (PL)*** agrupará las tareas de planificación inicial, así como aquellas que, en su caso, fuera necesario realizar para mantener una planificación adecuada. Dichas tareas son:

- PL.1: Tareas relacionadas con la identificación de requisitos, toma de decisiones iniciales, análisis de información y resolución de dudas.
- PL.2: Planificación inicial orientada a la preparación del entorno de desarrollo del proyecto.
- PL.3: Actualización, si fuera necesaria, de la planificación inicial.

El paquete de trabajo ***Seguimiento y Control (SyC)*** agrupará las tareas necesarias para garantizar el adecuado desarrollo del proyecto, comparando el plan previsto con lo finalmente llevado a cabo para poder gestionar las modificaciones. Dichas tareas son:

- SyC.1: Reuniones con las tutoras a lo largo de todo el proyecto.
- SyC.2: Elaboración de un documento a modo de diario para anotar la dedicación diaria y las tareas realizadas.
- SyC.3: Elaboración de una hoja de cálculo para representar estas dedicaciones.
- SyC.4: Recopilación de información relevante sobre el desarrollo del proyecto.
- SyC.5: Contraste de la información de seguimiento con los planes iniciales, identificándose las desviaciones más significativas y los riesgos emergentes.

Fase de Documentación:

El paquete de trabajo ***Realizar Informe (INF)*** agrupará todas las tareas necesarias para llevar a cabo el desarrollo del informe del trabajo de fin de grado. Dichas tareas son:

- INF.1: Preparar y configurar el entorno para el desarrollo de la memoria con LaTeX[1].
- INF.2: Desarrollo de la memoria.

El paquete de trabajo ***Preparar Defensa (DEF)*** agrupará todas las tareas necesarias para preparar la defensa del trabajo de fin de grado. Dichas tareas son:

- DEF.1: Crear la presentación que servirá de apoyo visual en la defensa del proyecto.
- DEF.2: Preparar la defensa repasando y mejorando, en caso de que sea necesario, la presentación creada.

3.4. Planificación Temporal

Una vez definido el alcance, es importante desarrollar una buena planificación para determinar las fechas y plazos previstos para la realización del proyecto, procurando que ésta sea lo más realista posible.

Para ello, a continuación se especifican las dedicaciones estimadas para cada una de las tareas de cada paquete de trabajo y las dependencias existentes entre las mismas.

Con dichos datos, finalmente, se muestra un Diagrama Gantt que representa los periodos de realización previstos para cada tarea y los hitos determinados durante el proyecto.

3.4.1. Estimación de dedicación

En la tabla 8.2 se expresan las horas de dedicación estimadas para cada una de las tareas detalladas en el punto anterior, agrupando también las horas totales resultantes para cada paquete de trabajo así como para el proyecto entero.

[1]LaTeX, un sistema de composición de textos: https://www.latex-project.org/

Tabla 3.1: Dedicaciones estimadas en horas para cada tarea

Tareas	Estimación en horas
Proyecto Fin de Grado	**356**
Gestión de Conocimientos	**38**
Adquisición de Competencias (AC)	38
AC.1: RDF y SPARQL	5
AC.2: Protégé	5
AC.3: React.js y MaterializeCss	8
AC.4: Firebsae	5
AC.5: OpenLink Virtuoso	8
AC.6: Gráficas de Google	2
AC.7: Herramientas secundarias	5
Diseño (DI)	**20**
Diseño Ontología (DIO)	13
DIO.1: Primera versión	8
DIO.2: Mejorar y completar primera versión	5
Diseño Interfaces (DII)	7
DII.1: Primera versión	2
DII.2: Mejorar y completar primera versión	5
Datos (D)	**45**
Traducción Datos a RDF (TD)	15
TD.1: Estructura datos en RDF	5
TD.2: Módulo de traducción de datos	10
Alojamiento de Datos (LD)	30
LD.1: Configurar entorno	10
LD.2: Implementar alojamiento de datos	20
Sistema Web (SW)	**97**
Desarrollo Interfaces (DSW)	40
DSW.1: Desarrollo inicial	20
DSW.2: Cambios y mejoras versión inicial	20
Consexión con Datos (CSW)	30
CSW.1: Preguntas SPARQL	10
CSW.2: Conexión con Virtuoso	10
CSW.3: Recogida y tratamiento de datos	10

Continuación de la tabla 3.1

Tareas	Estimación en horas
Generar Gráficas (GSW)	12
GSW.1: Tratamiento datos	6
GSW.2: Opciones y características gráficas	6
Alojamiento Sistema Web (LSW)	15
LSW.1: Configuración inicial Firebase	5
LSW.2: Alojamiento final proyecto	10
Pruebas (P)	**30**
Pruebas Datos (PD)	7
PD.1: Verificación traducción datos	2
PD.2: Verificación alojamiento datos	5
Pruebas Sistema Web (PSW)	23
PSW.1: Verificación interfaces	5
PSW.2: Verificación preguntas	4
PSW.3: Verificación tratamiento datos	4
PSW.4: Verificación gráficas	8
PSW.5: Verificación alojamiento proyecto	2
Gestión	**44**
Planificación (PL)	14
PL.1: Toma decisiones inicial	2
PL.2: Planificación inicial	10
PL.3: Actualización planificación inicial	2
Seguimiento y Control (SyC)	30
SyC.1: Reunión con tutoras	13
SyC.2: Diario dedicaciones	5
SyC.3: Hoja de cálculo dedicaciones	2
SyC.4: Recopilación información	5
SyC.5: Contraste información	5
Documentación	**82**
Realizar Informe (INF)	72
INF.1: Configurar entorno LaTeX	2
INF.2: Desarrollo informe	70
Preparar Defensa (DEF)	10
DEF.1: Crear presentación	4

Continuación de la tabla 3.1

Tareas	Estimación en horas
DEF.2: Repasar y mejorar defensa	6

3.4.2. Dependencias entre tareas

Para poder realizar una buena planificación es importante identificar todas las posibles dependencias existentes entre las distintas tareas a abordar, con el fin de determinar los periodos de realización de cada una de una manera correcta.

A continuación se detallan dichas dependencias y finalmente, en la figura 3.3, se muestra el diagrama asociado a las mismas.

PL.1 será la tarea inicial, sin ella no se puede comenzar el proyecto ya que abarca tareas tan básicas como la identificación de requisitos, toma de decisiones iniciales o el análisis de la información.

Los paquetes de trabajo relacionados con los *Datos*, el *Diseño*, el *Sistema Web* y la *Documentación* tienen dependencias internas de realización entre las tareas de su mismo paquete. Estas dependencias internas se han representado ordenando dichas tareas dentro del paquete, de manera que el número de cada identificador de tarea representa la prioridad de realización de la misma. Por ejemplo, en el paquete de trabajo de *Conexión con datos (CSW)*, la realización de la tarea **CSW.1** debe preceder a la de la tarea **CSW.2** y, a su vez, la tarea **CSW.2** debe completarse antes del inicio de la tarea **CSW.3**.

Aparte de estas dependencias internas, también existen dependencias entre tareas de distintos paquetes de trabajo.

El primer ejemplo de ello son las tareas del paquete de trabajo *Adquisición de Conocimientos*, que tienen dependencias directas con las tareas que ponen en práctica éstos conocimientos a adquirir. De manera que:

- El comienzo de las tareas **AC.1** y **AC.2** deben preceder a la tarea **DIO.1**.
- El comienzo de la tarea **AC.3** debe preceder a la tarea **DSW.1**.
- El comienzo de la tarea **AC.4** debe preceder a la tarea **LSW.1**.
- El comienzo de la tarea **AC.5** debe preceder a la tarea **LD.1**.
- El comienzo de la tarea **AC.6** debe preceder a la tarea **GSW.1**.

- El comienzo de la tarea **Ac.7** debe preceder a la tarea **DSW.1**.

La tarea **DIO.1** también deberá preceder a la realización de la tarea **TD.1**, ya que es necesario tener una ontología diseñada para poder determinar cómo representar la información. De la misma manera, la tarea **DSW.1** encargada de implementar una primera versión del Sistema Web, deberá empezar después de que la tarea **DII.2** haya empezado, ya que tiene que haber un diseño pensado para poder implementarlo. Además, la tarea DII.2 deberá acabar antes de que lo haga la tarea DSW.2, el diseño final tiene que estar pensado para poder acabar la implementación del mismo.

La tarea **TD.2** deberá finalizar antes de comenzar la tarea **LD.2**, ya que para poder alojar los datos, éstos tienen que estar disponibles.

La tarea **DSW.2** debe haber comenzado antes del inicio de la tarea **CSW.2**, ya que para poder desarrollar funcionalidades en el sistema web, la interfaz de la misma debe estar mínimamente disponible.

De la misma manera, la tarea **LSW.2** no podrá comenzar hasta que la tarea **DSW.2** haya sido iniciada, ya que para poder alojar el sistema web éste debe estar disponible.

Por otro lado, la tarea **LD.2** deberá preceder a la tarea **CSW.1**, ya que se pretende conectar el Sistema Web con los datos, por lo que para ello deben de estar los datos ya alojados en Virtuoso.

La tarea **GSW.1** deberá realizarse después de haber terminado la tarea **CSW.3**, ya que para poder añadir datos a las gráficas es necesario recibirlos.

Por último, las tareas relacionadas con las Pruebas no podrán ser abordadas si las tareas objetivo de su verificación no han comenzado. De manera que:

- El inicio de la tarea **TD.2** debe preceder a la tarea **PD.1**.
- El inicio de la tarea **LD.2** debe preceder a la tarea **PD.2**.
- El inicio de la tarea **DSW.1** debe preceder a las tarea **PSW.1**.
- El inicio de la tarea **CSW.1** debe preceder a la tarea **PSW.2**.
- El inicio de la tarea **CSW.2** debe preceder a la tarea **PSW.3**.
- El inicio de la tarea **GSW.1** debe preceder a la tarea **PSW.4**.
- El inicio de la tarea **LSW.2** debe preceder a la tarea **PSW.5**.

Estas tareas de pruebas además, deberán acabar siempre más tarde que las tareas que verifican, ya que se presupone que las últimas pruebas serán las de verificación de que todo funciona correctamente y no hay más cambios que realizar.

3.4.3. Periodos de desarrollo e Hitos

A partir de las estimaciones de dedicaciones y las dependencias identificadas entre tareas, en este apartado se presenta un Diagrama Gantt que representa los distintos periodos de realización de cada tarea así como los hitos significativos del proyecto.

Para poder determinar los periodos de realización es importante también tener en cuenta la situación en la que el proyecto va a ser desarrollado, ya que no se trata de un proyecto aislado al que se le va a dedicar el 100 % del tiempo disponible.

Hasta el día 27 de Abril el proyecto se desarrollará a la vez que las asignaturas del segundo cuatrimestre de cuarto curso, por lo que la dedicación semanal se verá afectada en consecuencia. A partir de dicho día, la dedicación del proyecto aumentará en gran medida, permitiendo así que los periodos de realización de las tareas disminuyan. Además, se trabajará en el proyecto de lunes a domingo, días festivos incluídos, por lo que la dedicación del proyecto no se verá afectada por periodos vacacionales determinados.

En las figuras 3.4 y 3.5 se muestra el Diagrama Gantt agrupado por meses, mostrando los periodos de desarrollo de cada tarea, las dependencias especificadas y los hitos del proyecto.

Para una visión más clara de los periodos y las dependencias entre las tareas, en las figuras 3.6 y 3.7 se muestra el mismo diagrama agrupado por semanas.

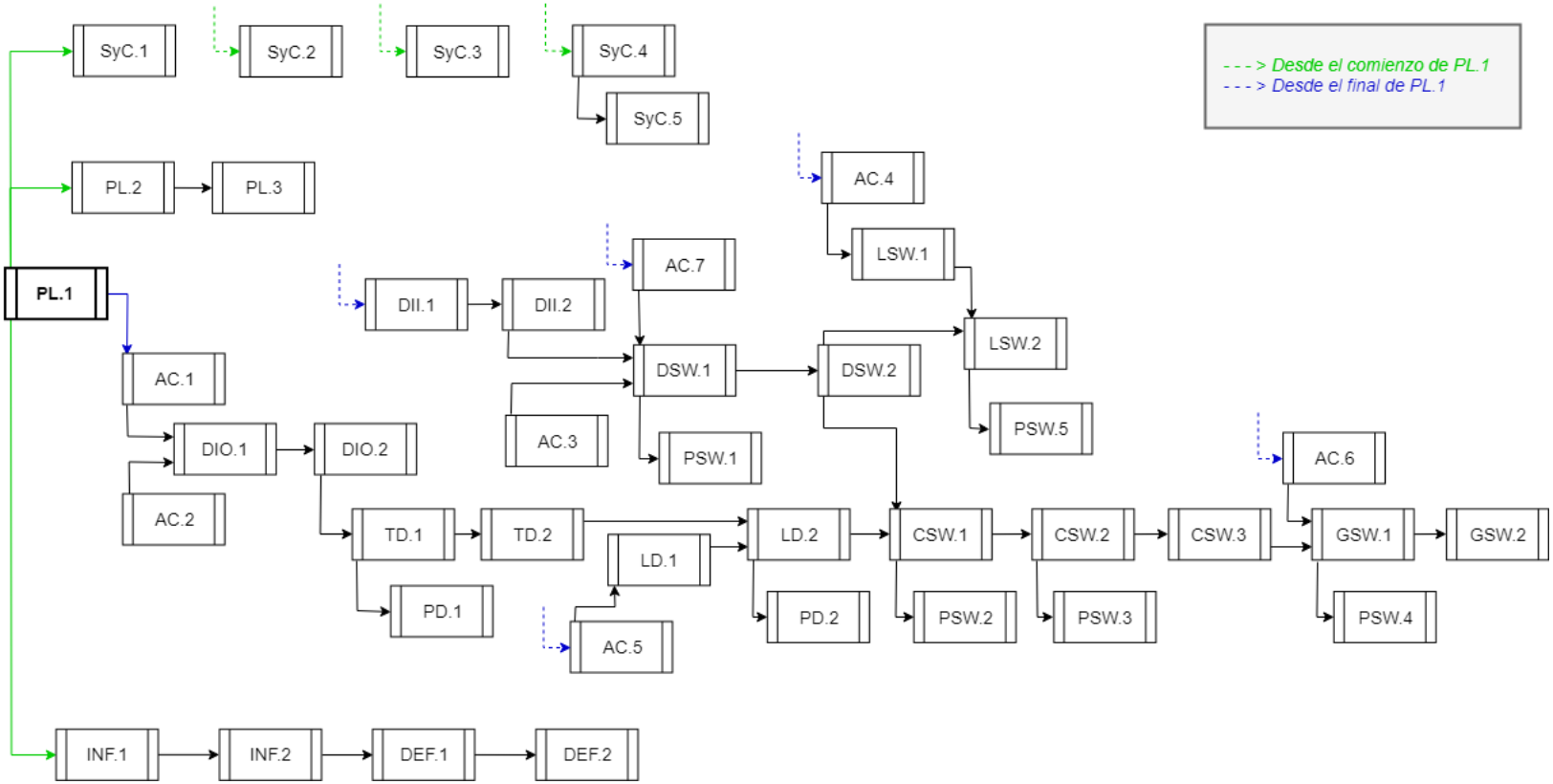

Figura 3.3: Diagrama de las dependencias entre las tareas definidas en el E.D.T.

	Nombre de la tarea	Fecha de Ini	Fecha final
1	**Proyecto Fin de Grado**		
2	*Hito.1: Primera reunión con ambas tutoras*	*19/01/18*	*19/01/18*
3	*Hito.2: Finalizar implementación del proyecto*	*01/06/18*	*01/06/18*
4	**Gestión de Conocimientos (GC)**	**19/01/18**	**18/05/18**
5	Adquisición de Competencias (AC)	19/01/18	18/05/18
6	AC.1: RDF y SPARQL	19/01/18	24/01/18
7	AC.2: Protégé	25/01/18	04/02/18
8	AC.3: React.js y MaterializeCss	05/03/18	29/03/18
9	AC.4: Firebase	27/03/18	29/03/18
10	AC.5: OpenLink Virtuoso	13/04/18	28/04/18
11	AC.6: Gráficas de Google	14/05/18	16/05/18
12	AC.7: Herramientas secundarias	12/03/18	18/05/18
13	**Diseño (DI)**	**05/02/18**	**18/05/18**
14	Diseño Ontología (DIO)	05/02/18	13/04/18
15	DIO.1: Primera versión ontología	05/02/18	09/02/18
16	DIO.2: Mejorar y completar primera versión ontología	10/02/18	13/04/18
17	Diseño Interfaces (DII)	25/02/18	18/05/18
18	DII.1: Primera versión interfaces	25/02/18	02/03/18
19	DII.2: Mejorar y completar primera versión interfaces	03/03/18	18/05/18
20	**Datos (D)**	**14/04/18**	**03/05/18**
21	Traduccion Datos a RDF (TD)	14/04/18	20/04/18
22	TD.1: Estructura de los datos en RDF	14/04/18	15/04/18
23	TD.2: Módulo de traducción de datos	16/04/18	20/04/18
24	*Hito.3: Completar traducción de datos*	*19/04/18*	*19/04/18*
25	Alojamiento de Datos (LD)	19/04/18	03/05/18
26	LD.1: Configurar Virtuoso	19/04/18	23/04/18
27	LD.2: Implementar alojamiento de datos	24/04/18	03/05/18
28	*Hito.4: Completar alojamiento de datos*	*02/05/18*	*02/05/18*
29	**Sistema Web (SW)**	**30/03/18**	**08/06/18**
30	Desarrollo Interfaces (DSW)	30/03/18	25/05/18
31	DSW.1: Desarrollo inicial de las interfaces	30/03/18	10/04/18
32	DSW.2: Desarrollo de cambios y mejoras a la versión inicial	11/04/18	25/05/18
33	Conexión con Datos (CSW)	04/05/18	15/05/18
34	CSW.1: Preguntas SPARQL	04/05/18	05/05/18
35	CSW.2: Conexión con Virtuoso	06/05/18	11/05/18
36	CSW.3: Recogida y tratamiento datos recogidos	12/05/18	15/05/18
37	*Hito.4: Completar conexión con Virtuoso*	*14/05/18*	*14/05/18*
38	Generar Gráficas (GSW)	16/05/18	23/05/18
39	GSW.1: Añadir datos a gráficas	16/05/18	19/05/18
40	GSW.2: Opciones y características gráficas	20/05/18	23/05/18
41	*Hito.5: Completar generación de gráficas*	*22/05/18*	*22/05/18*
42	Alojamiento Sistema Web (LSW)	02/04/18	08/06/18
43	LSW.1: Configuración inicial Firebase	02/04/18	05/04/18
44	LSW.2: Alojamiento final proyecto	15/05/18	08/06/18
45	*Hito.6: Completar alojamiento proyecto*	*07/06/18*	*07/06/18*
46	**Pruebas (P)**	**01/04/18**	**13/06/18**
47	Pruebas Datos (PD)	18/04/18	30/04/18
48	PD.1: Verificación traducción datos	18/04/18	27/04/18
49	PD.2: Verificación alojamiento datos	26/04/18	30/04/18
50	Pruebas Sistema Web (PSW)	01/04/18	13/06/18
51	PSW.1: Verificación interfaces	01/04/18	07/05/18
52	PSW.2: Verificación preguntas SPARQL	05/05/18	06/05/18
53	PSW.3: Verificación conexión con Virtuoso	08/05/18	13/05/18
54	PSW.4: Verificación gráficas	18/05/18	25/05/18
55	PSW.5: Verificación alojamiento proyecto	17/05/18	13/06/18

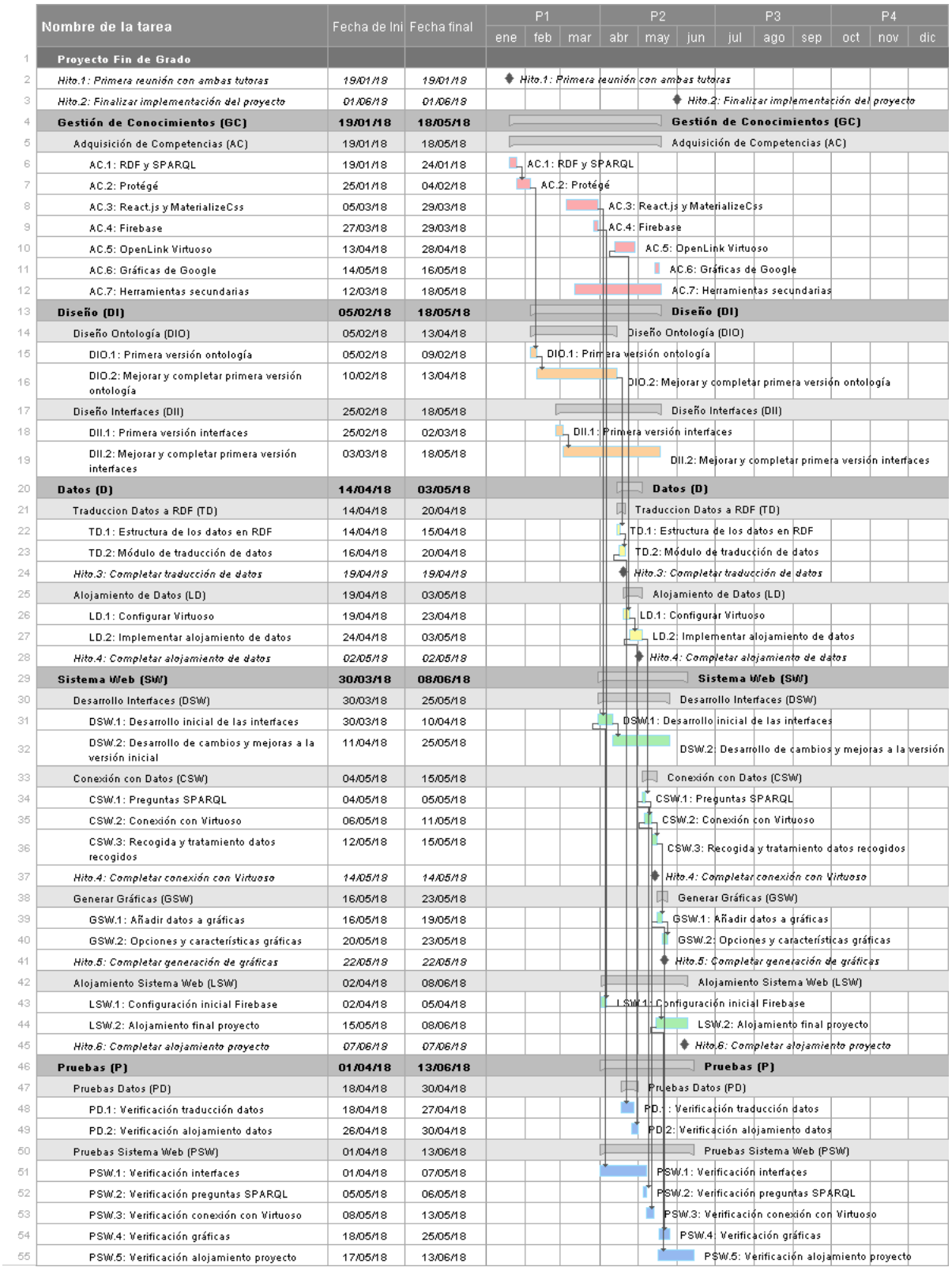

Figura 3.4: Diagrama Gantt por meses - Primera parte

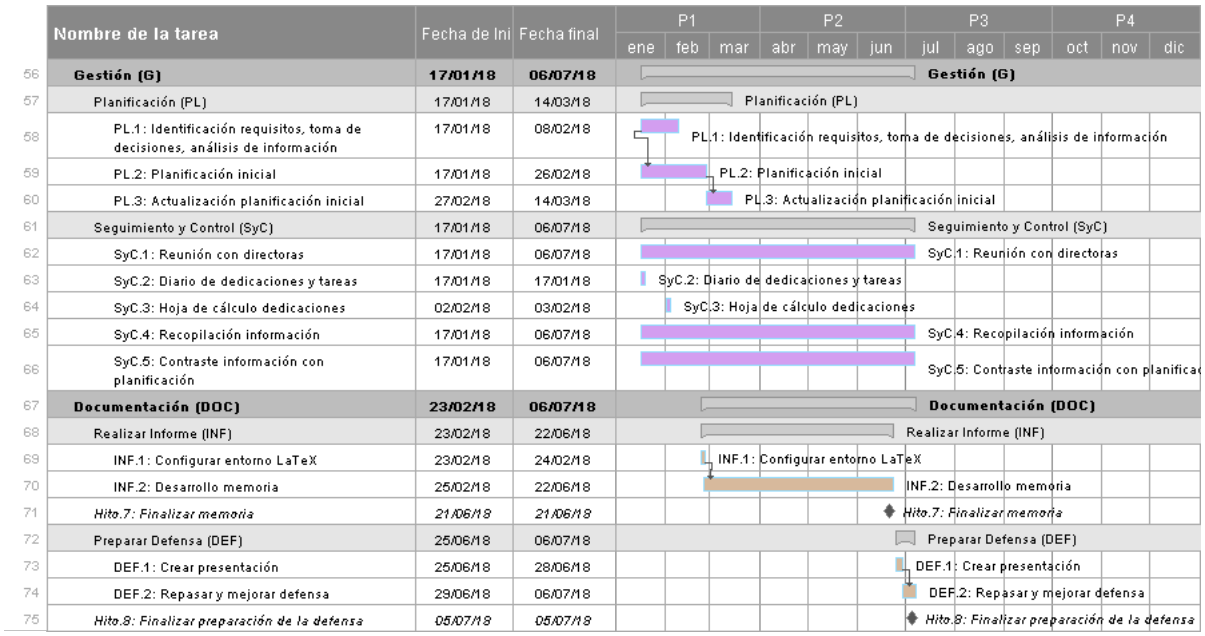

Figura 3.5: Diagrama Gantt por meses - Segunda parte

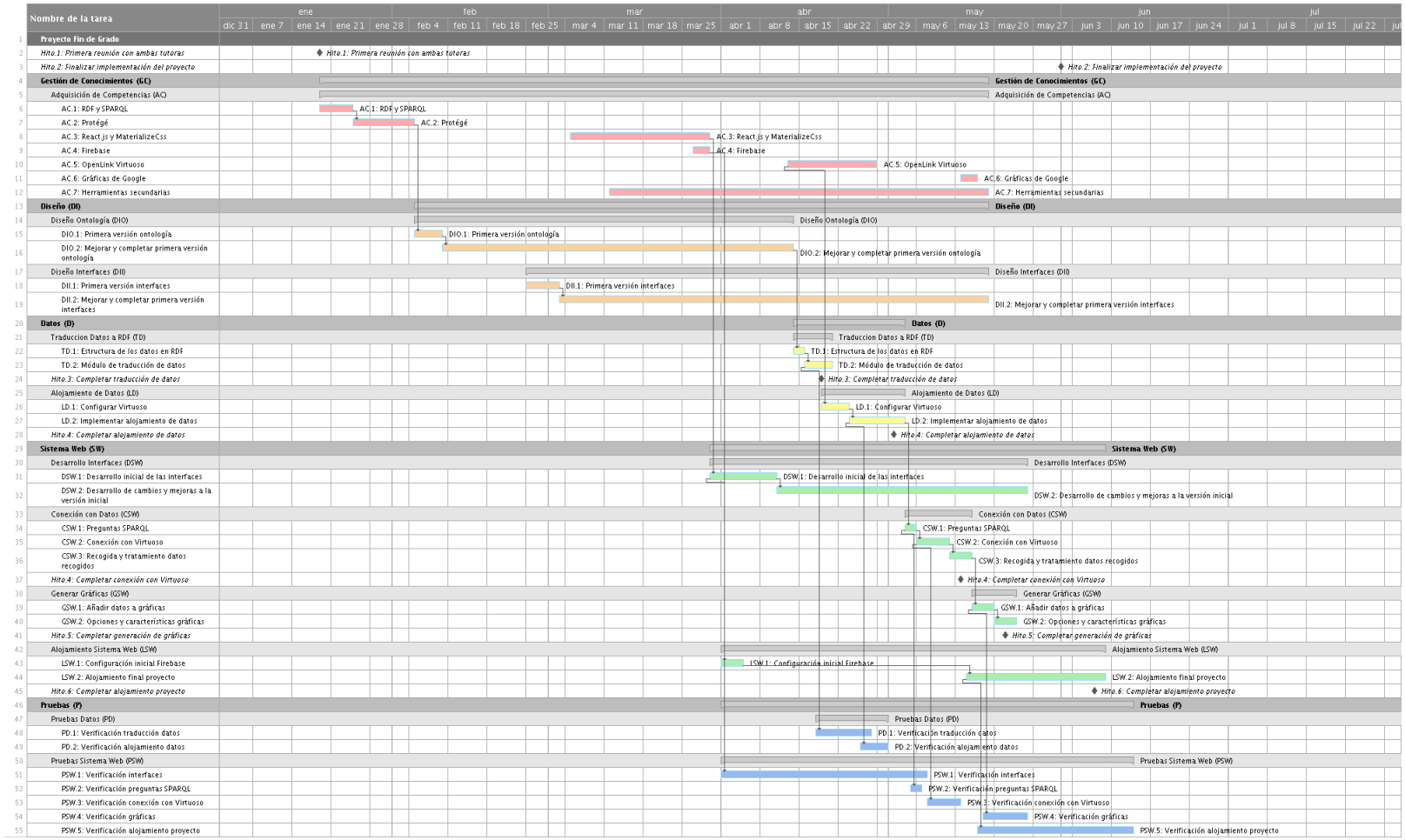

Figura 3.6: Diagrama Gantt por semanas - Primera parte

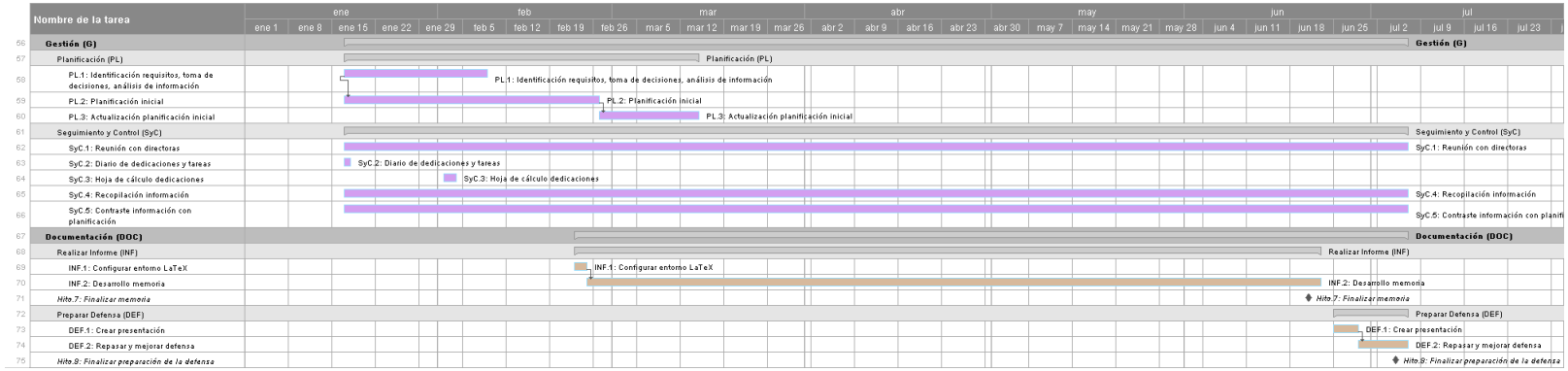

Figura 3.7: Diagrama Gantt por semanas - Segunda parte

3.5. Herramientas

Las herramientas que se utilizarán para el desarrollo del proyecto serán las detalladas a continuación.

3.5.1. Google Drive

Google Drive[2] es un servicio de almacenamiento de archivos en la nube proporcionado por Google. Éste permite a los usuarios acceder de manera online a los archivos guardados desde cualquier smartphone, tablet u ordenador. Además, también ofrece una versión de escritorio que facilita la creación de copias de seguridad mediante la sincronización de los archivos en la nube.

Se usará la versión gratuita de esta herramienta, en la que se ofrecen hasta 15GB de almacenamiento, y servirá de gran apoyo para las copias de seguridad de toda la documentación principal del proyecto. Se dispondrá de la versión de escritorio de Google Drive con un directorio dedicado al proyecto, consiguiendo una sincronización constante de todos los archivos contenidos en la misma siempre que la conexión a internet lo permita.

3.5.2. Git

Git[3] es un software de control de versiones que permite llevar registro de los cambios realizados en los archivos así como coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Esta herramienta se utilizará principalmente en conjunto con *GitHub*, herramienta que se detalla a continuación, para tener un control claro sobre los cambios que se realizan en el código fuente y para simplificar la realización de las copias de seguridad del mismo.

3.5.3. GitHub

GitHub[4] es una plataforma de desarrollo colaborativo que permite el almacenamiento del código fuente de diferentes proyectos utilizando el sistema de control de versiones Git.

De esta manera, se tendrá una copia de seguridad de todo el proyecto almacenada en GitHub, utilizando la herramienta Git ya mencionada para el control de dichos archivos.

[2]Google Drive: https://www.google.com/drive/
[3]Git: https://git-scm.com/
[4]GitHub: https://github.com/

3.5.4. Atom

Atom[5] es un editor de código fuente *open-source* desarrollado por GitHub y disponible para macOS, Linux y Windows. A pesar de ser una herramienta relativamente nueva (su primera versión se lanzó en 2014), Atom es ya uno de los editores de texto más utilizados y de más prestigio entre los editores libres de pago.

Atom dispone de una gran variedad de lenguajes de programación, un nivel muy alto de personalización y una serie de herramientas que facilitan en gran medida el desarrollo. Una de las ventajas más destacables de Atom es la integración de la herramienta Git en la propia aplicación, facilitando así el control de versiones del proyecto.

Por todo ello, Atom será la herramienta elegida para llevar a cabo la implementación del proyecto.

3.5.5. React.js

React.js[6] es una librería de JavaScript para la creación de interfaces de usuario que se basa en los principios de las *Single-Page Application (SPA)* o aplicaciones de una única página. De esta manera, React.js permite definir distintos componentes que se irán actualizando o renderizando dependiendo del estado de la aplicación en cada momento, evitando la carga continua de las páginas desde el servidor y mejorando la experiencia del usuario en la utilización de la aplicación.

React.js se utilizará para crear toda la interfaz del sistema web y para definir gran parte de la lógica de la aplicación.

3.5.6. MaterializeCss

MaterializeCss[7] es un Framework de CSS basado en los principios del *Material Design*.

Creado y diseñado por Google, *Material Design*[8] es un lenguaje de diseño que pretende sintetizar los principios clásicos de un buen diseño con la innovación de la ciencia y la tecnología. De esta manera, tienen como objetivo desarrollar un único sistema de diseño que unifique las experiencias de los usuarios a través de las diferentes plataformas, dispositivos y métodos de entrada.

[5] Atom: https://atom.io/
[6] React.js: https://reactjs.org/
[7] MaterializeCSS: https://materializecss.com/
[8] Material Design: https://material.io/design/introduction/

MaterializeCss nos permite entonces, de una manera sencilla, adoptar estos principios del *Material Design* para integrarlos en nuestra aplicación.

3.5.7. Gráficas de Google

Gráficas de Google[9] es un servicio Web interactivo que permite la creación de diferentes tipos de gráficas a partir de una especificación aportada en JavaScript.

Las gráficas que formarán parte del sistema web se harán a partir de las Gráficas de Google.

3.5.8. Firebase

Firebase[10] es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles que ofrece un amplio número de servicios y productos complementarios.

En este caso, la aplicación I4TSPS con la que se fusionará el módulo desarrollado en este TFG utiliza los servicios de *Hosting* y de Base de datos proporcionados por Firebase para su alojamiento y para el almacenamiento de distintos datos. Por ello, en este TFG también se utilizarán ciertas características de dichos servicios.

3.5.9. OpenLink Virtuoso

OpenLink Virtuoso[11] proporciona una solución para el acceso a los datos, visualización, integración y gestión de bases de datos relacionales multi-modelos (Tablas SQL y/o Grafos RDF).

De esta manera, Virtuoso proporcionará el almacenamiento de los datos, que en este caso estarán representados mediante Grafos RDF.

3.5.10. Google Cloud Platform

Google Cloud[12] es una plataforma que reúne un gran número de aplicaciones de desarrollo web ofrecidas por Google y que proporciona infraestructuras como servicio *(IaaS)*, plataformas como servicio *PaaS* y entornos de cómputo sin servidor (*serverless computing*).

[9] Gráficas de Google: https://developers.google.com/chart/
[10] Firebase: https://firebase.google.com/
[11] OpenLink Virtuoso: https://virtuoso.openlinksw.com/
[12] Plataforma Google Cloud: https://cloud.google.com/

En este TFG, se utilizará una máquina virtual creada en Google Cloud Platform para la instalación de Virtuoso OpenLink en la misma.

3.5.11. Protégé

Protégé[13] es un editor de ontologías y Framework para el desarrollo de sistemas inteligentes gratuito y de carácter *open-source*.

Protégé se utilizará para crear la ontología que describirá los sensores de la extrusora de la que se captan los datos manejados en el proyecto.

3.5.12. LaTeX y Overleaf

LaTeX[14] consiste en un sistema de composición de textos de alta calidad que facilita la creación de documentos técnicos y científicos. Dadas sus características y las facilidades que aporta, se ha decidido la utilización de LaTeX para la creación de la documentación del proyecto.

Para llevar esto a cabo, se ha decidido utilizar el sistema *Overleaf*[15]. Overleaf es un sistema colaborativo en línea de escritura y publicación de textos en LaTeX. Aparte de las cualidades específicas del sistema que permiten una edición sencilla y efectiva, al ser un sistema en línea ofrece además la ventaja de poder acceder al trabajo realizado desde cualquier dispositivo o plataforma. Se utilizará la versión gratuita que ofrece el mismo, teniendo 200MG de almacenamiento disponibles.

3.5.13. Smartsheet

Smartsheet[16] es una aplicación que permite la gestión de trabajos ofreciendo opciones de colaboración, de manera que se pueden crear y asignar tareas, gestionar calendarios, compartir documentos, etc.

En este caso, se utilizará únicamente la herramienta de creación de Diagramas Gantt que ofrece Smartsheet.

[13] Protégé: https://protege.stanford.edu/
[14] LaTeX: https://www.latex-project.org/
[15] Overleaf: https://www.overleaf.com/
[16] Smartsheet: https://www.smartsheet.com/

3.5.14. Draw.io

Draw.io[17] es una herramiento *open-source* que permite la creación de una amplia variedad de diagramas y figuras.

Esta herramienta será utilizada para la creación de los diferentes diagramas que serán parte de la documentación del proyecto.

3.6. Gestión de Riesgos

En todo proyecto es importante la gestión de los riesgos que puedan aparecer en el proceso de desarrollo del mismo. Una pronta identificación de dichos riesgos ayuda a llevar a cabo una mejor gestión de los mismos, teniendo la capacidad de prevenirlos en una mayor medida así como de mitigarlos de la forma más eficaz posible.

A continuación se detallan los posibles riesgos identificados para el proyecto actual, especificando también las prevenciones adquiridas para evitarlos y los planes de mitigación planeados en caso de que finalmente ocurran.

3.6.1. Dificultad compaginando el proyecto con asignaturas

El comienzo del TFG también coincide con el comienzo del último cuatrimestre del cuarto curso del grado de Ingeniería Informática, por lo que durante gran parte del desarrollo del proyecto éste deberá ser compaginado con la dedicación necesaria a invertir en las asignaturas cursadas. Esto implica el riesgo de no ser capaz de compaginar correctamente ambas actividades.

- **Probabilidad:** media.
- **Impacto:** alto.
- **Consecuencias:** disminución en la dedicación a la realización del proyecto, dificultando el cumplimiento de los plazos previstos inicialmente.
- **Prevención:** realizar una planificación realista teniendo en cuenta el tiempo necesario a invertir en el desarrollo de las asignaturas de la carrera que se deberán abordar a la vez que el proyecto.

[17] Draw.io: https://www.draw.io/

- **Plan de mitigación:** replanificar lo previsto para adecuarse mejor a compaginar ambas actividades.

3.6.2. Dificultad en el aprendizaje de las herramientas a utilizar

La mayor parte de las herramientas necesarias para este proyecto son desconocidas y no han sido utilizadas hasta el momento, implicando la necesidad de una adquisición de conocimientos sobre las mismas durante parte del desarrollo del TFG. Esto implica el riesgo de tener más dificultad de la esperada en el aprendizaje de dichas herramientas.

- **Probabilidad:** media.
- **Impacto:** medio.
- **Consecuencias:** retraso en el desarrollo de las funcionalidades del proyecto, dificultando el cumplimiento de los plazos previstos inicialmente.
- **Prevención:** reservar un tiempo considerable a la adquisición de conocimientos, teniendo en cuenta la cantidad de herramientas novedosas que se utilizan.
- **Plan de mitigación:** adelantar la realización de otras tareas mientras se adquiere un conocimiento adecuado de la herramienta en cuestión, intentando causar un menor impacto en la planificación, y replanificar si esto no es posible.

3.6.3. Planificación incorrecta

Como en cualquier proyecto que requiera una planificación, existe el riesgo de que dicha planificación no esté correctamente desarrollada.

- **Probabilidad:** media.
- **Impacto:** alto.
- **Consecuencias:** retraso en los cumplimientos de los plazos previstos.
- **Prevención:** realizar una planificación con amplios plazos para la realización de tareas, dejando margen disponible para los errores.
- **Plan de mitigación:** replanificar los plazos destinados a cada tarea intentando conseguir el menor impacto posible en la fecha de entrega final.

3.6.4. Problemas técnicos del software utilizado

El proyecto depende de cierto software ajeno que permite tanto su desarrollo como su posterior utilización, lo que implica el riesgo de tener problemas técnicos con dicho software.

- **Probabilidad:** baja.
- **Impacto:** medio.
- **Consecuencias:** imposibilidad de avanzar en ciertas tareas por un tiempo determinado, retrasando el cumplimiento de los plazos de las mismas.
- **Prevención:** realizar un mantenimiento regular del sistema, intentando instalar solo el software necesario.
- **Plan de mitigación:** adelantar la realización de otras tareas mientras se soluciona el problema y replanificar de no ser posible.

3.6.5. Problemas técnicos del hardware utilizado

El proyecto también depende del hardware en el que está siendo desarrollado y posteriormente utilizado, lo que implica de la misma manera el riesgo de tener problemas técnicos con dicho hardware.

- **Probabilidad:** baja.
- **Impacto:** alto.
- **Consecuencias:** imposibilidad de avanzar ciertas tareas por un tiempo determinado, retrasando el cumplimiento de los plazos de las mismas.
- **Prevención:** realizar un mantenimiento regular de las herramientas de hardware, asegurando el correcto funcionamiento de las herramientas que proporcionan las copias de seguridad para un respaldo de todos los progresos realizados en el proyecto.
- **Plan de mitigación:** replanificar procurando tener el menor impacto posible en la fecha de finalización del proyecto

3.6.6. Problemas de índole personal

Durante todo el periodo de desarrollo del proyecto existe el riesgo de que aparezcan problemas de índole personal inesperados.

- **Probabilidad:** baja
- **Impacto:** medio.
- **Consecuencias:** retraso en los cumplimientos de los plazos previstos para ciertas tareas.
- **Prevención:** no se puede prever ya que depende del tipo de problema surgido.
- **Plan de mitigación:** replanificar en consecuencia con los problemas surgidos.

CAPÍTULO 4

Captura de Requisitos

Para la captura de requisitos, por el marco en el que se desarrolla el proyecto actual y teniendo en cuenta los objetivos futuros del mismo, es necesario analizar dos principales fuentes de requisitos:

- Los requisitos específicos del producto a desarrollar, teniendo en cuenta las características acordadas para el mismo.
- Los requisitos implícitos que se derivan de la futura unión con la plataforma I4TSPS desarrollada por el grupo de investigación BDI. El hecho de que, en un futuro, el sistema web fruto de este TFG pase a formar parte de otro que ya está en fase de desarrollo impone unos requisitos previos específicos que es necesario tener en cuenta.

Teniendo esto en mente, a continuación se presentan los requisitos funcionales y no funcionales del proyecto, así como los modelos resultantes a partir de los mismos.

4.1. Requisitos funcionales

Para empezar, teniendo en cuenta la futura fusión con la plataforma web I4TSPS, suponemos que los usuarios que accedan al sistema web estarán ya identificados, ya que

se utilizará el sistema de control de accesos ya implementado en dicha plataforma web. Por esto, se desarrollará el sistema web planteando una futura diferenciación de roles de usuarios pero, como bien se ha especificado en las exclusiones del alcance del proyecto (sección 3.3.1), no se desarrollará la funcionalidad que permita dicha gestión de usuarios.

Dicho esto, una vez que los usuarios estén identificados, el sistema les ofrecerá algunas de las funcionalidades disponibles según el rol que tengan definido. Estas funcionalidades tendrán un paso previo que consistirá en seleccionar la máquina sobre la que se desean hacer las operaciones.

El sistema web a desarrollar en este TFG ofrecerá entonces dos funcionales principales:

- Anotar y almacenar de manera adecuada datos de ciertos sensores.
- Ofrecer la opción de realizar consultas personalizadas sobre los datos almacenados, mostrando el resultado mediante gráficas significativas.

Anotación y almacenamiento de datos

El sistema ofrecerá al usuario la posibilidad de subir archivos en formato *CSV*[1] que contengan datos de uno de los sensores contemplados en la máquina seleccionada, para que posteriormente el sistema los anote utilizando el modelo RDF siguiendo la ontología desarrollada.

Una vez que los datos estén correctamente almacenados, se le ofrecerá al usuario también la opción de descargarse dicho fichero generado en formato *Turtle*[2], formato elegido para la especificación de los datos en RDF.

Consulta de datos

En relación a la funcionalidad de consulta de datos, la aplicación ofrecerá al usuario la opción de seleccionar los sensores de los que se desea consultar cierta información. Los sensores disponibles serán los descritos en la ontología desarrollada para el tipo de máquina seleccionado.

Teniendo en cuenta los posibles objetivos de las consultas a realizar, se ha decidido diferenciar tres tipos de preguntas:

[1] Los archivos *CSV* o *Comma-Separated Values* son un tipo de documento de formato abierto para representar datos en forma de tabla, separando las columnas por comas y las filas por saltos de línea.

[2] *Turtle* o *Terse RDF Triple Language* es una sintaxis y formato de fichero para expresar datos que sigan el modelo RDF. Utiliza "tripletas"semánticas para su definición y la extensión asociada es *.ttl*

- **Preguntas de información general.** Este tipo de preguntas permitirán consultar información general sobre los sensores seleccionados, con la posibilidad también de personalizar la información a mostrar. De esta manera, se podrán filtrar los resultados por intervalos de tiempo o valores, agrupar los resultados para mostrar el valor máximo o mínimo, etc.
  Ejemplos:

  1. Consultar los valores de los sensores *sensorA* y *sensorB* entre las fechas 20-04-2018 y 24-04-2018.
  2. Consultar los valores del sensor *sensorA* cuando éstos estén dentro del rango 100-200 y entre las horas 10:00 y 15:00 de cada día.
  3. Consultar la media aritmética y el mínimo por cada día de los valores de los sensores *sensorA*, *sensorB* y *sensorC*.

- **Preguntas de relación entre sensores.** Este tipo de preguntas permitirán consultar los valores de ciertos sensores fijando el valor de otros. Es decir, el usuario proporcionará el valor de uno o varios de los sensores seleccionados y el sistema mostrará el valor del resto de sensores seleccionados cuando se cumplan los valores impuestos a los primeros. De esta manera, el usuario podrá tener una visión clara de las distintas relaciones que se dan entre determinados sensores.
  Ejemplos:

  1. Consultar los valores de los sensores *sensorA* y *sensorB* cuando el *sensorC* tenga el valor 185.
  2. Consultar los valores del sensor *sensorA* cuando el sensor *sensorB* tenga su valor mínimo entre las fechas 15-03-2018 y 18-03-2018. Para calcular el valor mínimo del sensor *sensroB* solo se tendrán en cuenta los valores que entren en el rango 50-300.

- **Preguntas de detección de anomalías.** Este tipo de preguntas permitirán expresar una relación que los datos de los sensores seleccionados deberían cumplir en situaciones normales, de manera que el sistema mostrará los momentos en los que esta relación se incumple. Estos momentos de incumplimiento de la relación descrita por el usuario son lo que llamamos *anomalías* en los datos. De esta manera, el usuario podrá visualizar y detectar fácilmente los comportamientos no deseados en los diferentes sensores. Las relaciones de datos expresadas podrán ser de dos tipos:

- *Personalizadas*: el usuario crea una relación personalizada puntual para los sensores seleccionados en ese momento, definiendo la tendencia (ascendente o descendente) que tendrían que tener los valores de los sensores numéricos y el estado en el que se deberían encontrar los sensores *booleanos* (activados o desactivados).
- *Predefinidas*: relaciones de anomalías entre sensores ya predefinidas en el sistema. Además, las relaciones personalizadas que se consideren significativas también podrán ser guardadas como predefinidas.

Ambos tipos de relaciones están definidas mediante la misma estructura, por lo que los ejemplos de consultas no varían de un tipo a otro, solo varía el modo de selección de las mismas.
Ejemplos:

1. Mostrar los valores de los sensores cuando se incumpla la relación ”cuando *sensorA* y *sensorB* aumentan su valor, el valor de *sensorC* debería disminuir”.
2. Mostrar los valores de los sensores cuando se incumpla la relación ”cuando aumenta el valor de *sensorA*, el valor de *sensorB* debería aumentar también” entre las fechas 10-05-2018 y 12-05-2018.

Una vez realizada la pregunta, el sistema mostrará los resultados de forma gráfica, de manera que dichas gráficas se adapten a las características propias de la pregunta realizada y de los datos obtenidos.

4.2. Requisitos no funcionales

Una de las funcionalidades principales de este TFG es la consulta personalizada de datos para un futuro análisis de los mismos. Para abordar esta funcionalidad de consulta se ha optado por desarrollar un *Visual Query System (VQS)* o sistema visual de consultas, de manera que se facilite la consulta de dichos datos mediante una interfaz visual y comprensible para los usuarios expertos en el dominio sin necesidad de tener conocimientos informáticos.

En un sistema visual de consultas de este estilo, la *expresividad* y la *usabilidad* forman una base fundamental de los requerimientos no funcionales del mismo [Catarci et al., 1997].

La **expresividad** de un lenguaje o sistema hace referencia al rango de ideas que pueden ser representadas y comunicadas a través del mismo. En el sistema web a desarrollar se debe tener muy en cuenta la expresividad obtenida en el mismo, de manera que podamos asegurar la representación de un amplio rango del dominio a través de las herramientas ofrecidas al usuario. Esta búsqueda de expresividad deberá estar guiada por las necesidades principales de los futuros usuarios de la aplicación, de manera que el rango de información a representar se ajuste a las mismas.

La **usabilidad** por otro lado, hace referencia a la calidad de experiencia que tiene un usuario cuando interactúa con un producto o sistema. Ésta se basa en tres conceptos principales: la *eficiencia*[3] con la que se realiza la tarea, lo *intuitiva*[4] que es la realización de la tarea y el *grado de satisfacción*[5] que obtiene el usuario al realizarla. La usabilidad será entonces un requisito fundamental en el sistema web a desarrollar, de manera que el usuario sea capaz de entender de manera sencilla cómo realizar las consultas y de ejecutarlas de manera correcta para obtener el resultado deseado, donde la combinación de ambos elementos afectará directamente en el grado de satisfacción de dicho usuario. Para ello, el sistema deberá ser capaz de representar los conceptos necesarios del lenguaje de consulta de una manera sencilla y visual.

Por último, un buen equilibrio entre ambos conceptos es lo que realmente determina la calidad del sistema, ya que si se obtiene demasiada expresividad se puede perder usabilidad en el proceso y viceversa.

4.3. Casos de uso

En esta sección se desarrollan los casos de uso concretos relacionados con los requisitos funcionales anteriormente detallados y se explican los actores que forman parte en los mismos.

[3] *Eficiencia*: capacidad para realizar o cumplir adecuadamente una función.
[4] *Intuitivo*: aquello que se comprende de inmediato, sin estudiarlo, sin dedicarle tiempo y sin razonarlo.
[5] *Grado de satisfacción*: medida en la que los productos o sistemas cumplen o superan las expectativas de los usuarios.

4.3.1. Jerarquía de actores

Se han definido tres actores principales que formarán parte en la aplicación:

- **Anónimo**: representa a los usuarios que todavía no han iniciado una sesión en el sistema web, por lo que solo pueden acceder a las funcionalidades de registro e inicio de sesión.
- **Consultor de datos**: representa a los usuarios registrados que tienen acceso a las funcionalidades relacionadas con la consulta de datos de sus máquinas asociadas. Se les considera expertos en dichas máquinas y en los datos representados de las mismas.
- **Gestor de datos**: representa a los usuarios registrados que tienen acceso a las funcionalidades relacionadas con la consulta e inserción de datos de sus máquinas asociadas. Se les considera expertos en dichas máquinas y en sus datos así como responsables de la gestión de los datos que se podrán analizar sobre las mismas.

4.3.2. Modelo de Casos de Uso

Siguiendo esta jerarquía de actores, en la figuras 4.1 (anónimo), 4.2 (consultor de datos) y 4.3 (gestor de datos) se representan los modelos de casos de uso correspondientes a cada actor.

En dichos modelos se han utilizado dos colores para diferenciar ciertas características de los casos de uso:

- En **magenta** se definen los casos de uso relacionados con la gestión de usuarios de la aplicación. Éstos han sido únicamente representados gráficamente para una correcta visión general del sistema web final resultante, pero no serán detallados a continuación ya que no han sido fruto de este TFG.
- En **negro** se definen los casos de uso que sí van a ser resultado del TFG actual, por lo que se desarrollan a continuación haciendo una breve descripción de los mismos así como explicando los flujos de eventos asociados a cada uno.

Figura 4.1: Modelo de Casos de Uso: Anónimo

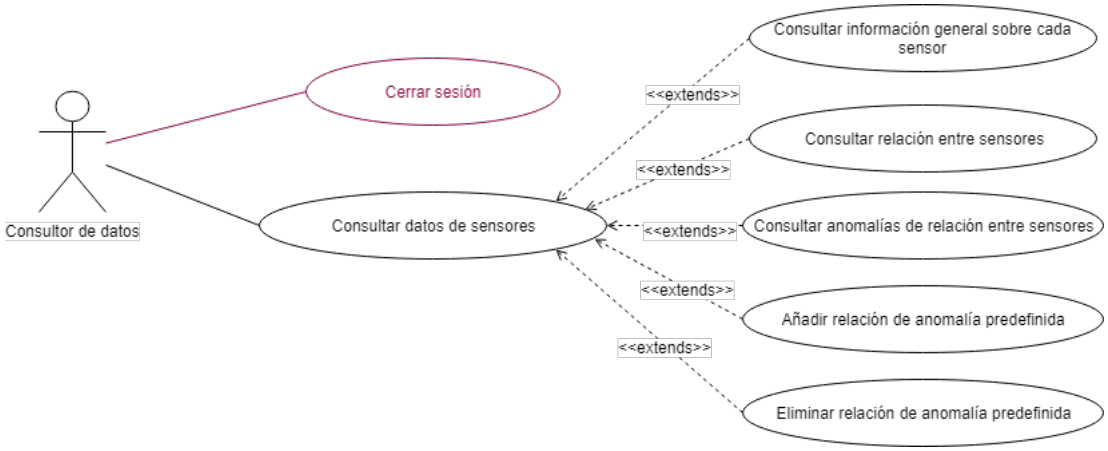

Figura 4.2: Modelo de Casos de Uso: Consultor de datos

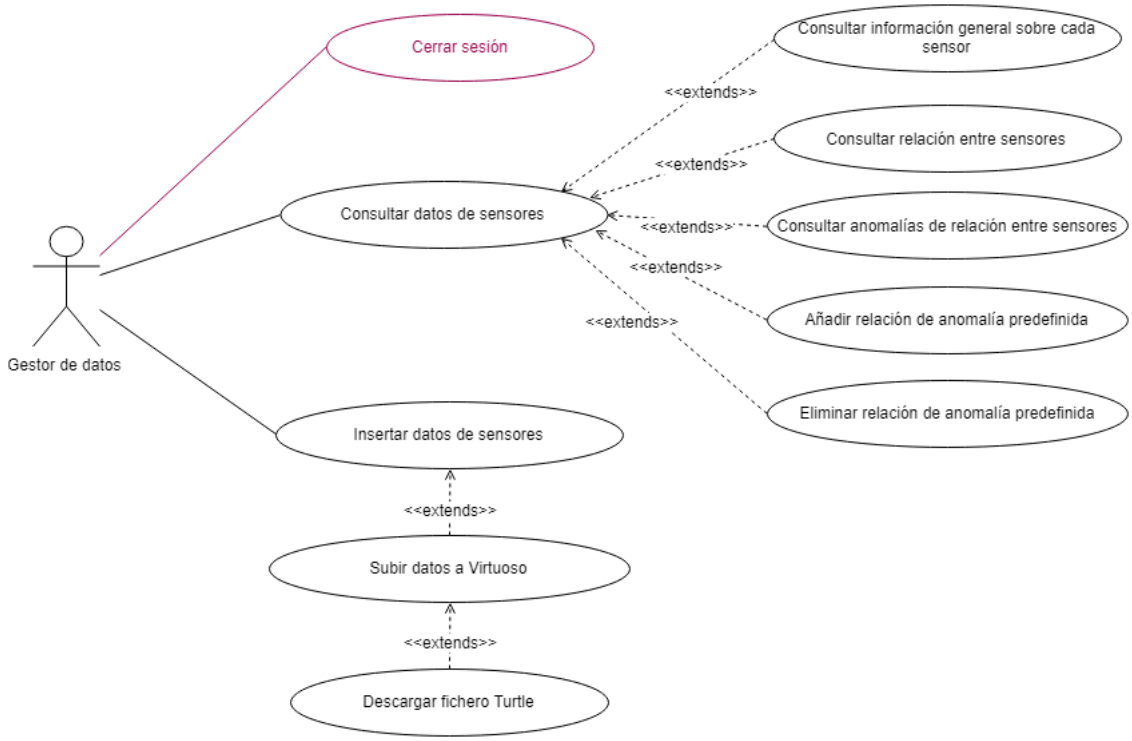

Figura 4.3: Modelo de Casos de Uso: Gestor de datos

Por último, los diagramas de secuencia asociados a cada uno de los casos de uso serán detallados en el anexo A de esta memoria.

Consultar datos de sensores

El usuario selecciona la máquina sobre la que quiere consultar información y el sistema muestra una serie de formularios correspondientes a los distintos tipos de consultas posibles a realizar.

Flujo de eventos:

1. El sistema muestra al usuario las máquinas extrusoras disponibles según la organización a la que pertenece. (Fig. 4.4)
2. El usuario selecciona la máquina sobre la que quiere realizar las consultas. (Fig. 4.5)
3. El sistema recopila la información necesaria sobre dicha máquina y muestra al usuario distintos formularios para la realización de las consultas. (Fig. 4.6)
    - En caso de no haber información disponible sobre la máquina seleccionada, el sistema informa al usuario y ofrece la opción de volver a escoger otra máquina.
4. Dependiendo de los objetivos del usuario, éste puede seleccionar la operación de consulta a realizar:
    - Si el usuario desea consultar información general sobre cada sensor, se extiende el caso de uso *Consultar información general sobre cada sensor*.
    - Si el usuario desea consultar información sobre la relación entre los valores de ciertos sensores, se extiende el caso de uso *Consultar relación entre sensores*.
    - Si el usuario desea consultar información sobre las anomalías aparecidas en la relación de valores de ciertos sensores, se extiende el caso de uso *Consultar anomalías de relación entre sensores*.
    - Si el usuario desea añadir una relación de anomalía a la lista de anomalías predefinidas, se extiende el caso de uso *Añadir relación de anomalía predefinida*.
    - Si el usuario desea eliminar una de las relaciones de anomalía aparecidas como predefinidas, se extiende el caso de uso *Eliminar relación de anomalía predefinida*.

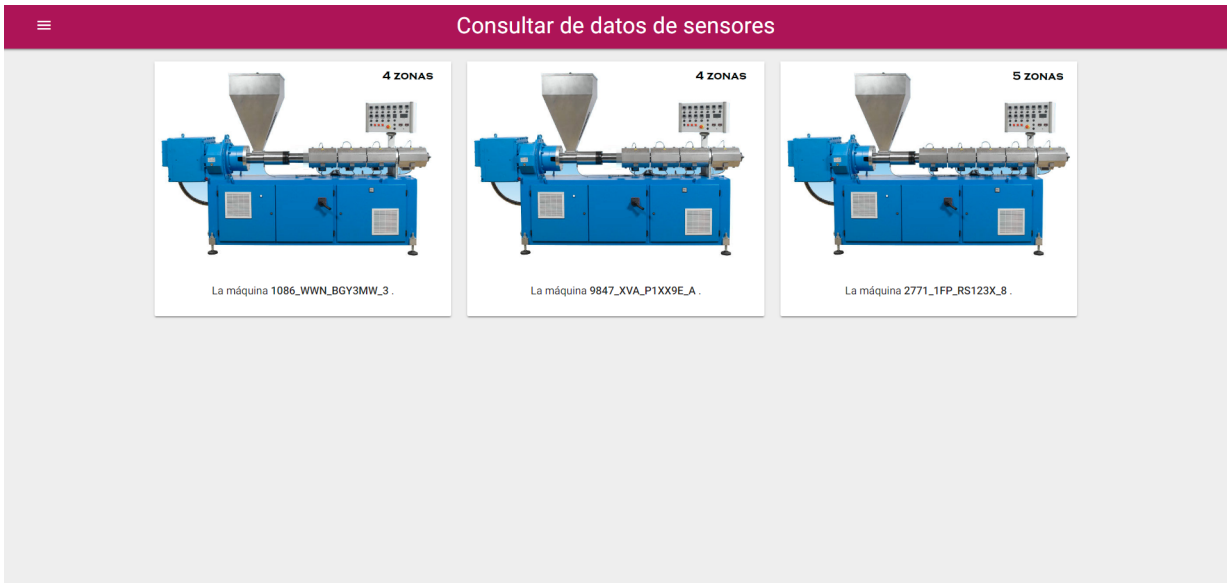

Figura 4.4: Caso de uso Consulta de Datos: máquinas disponibles en la organización.

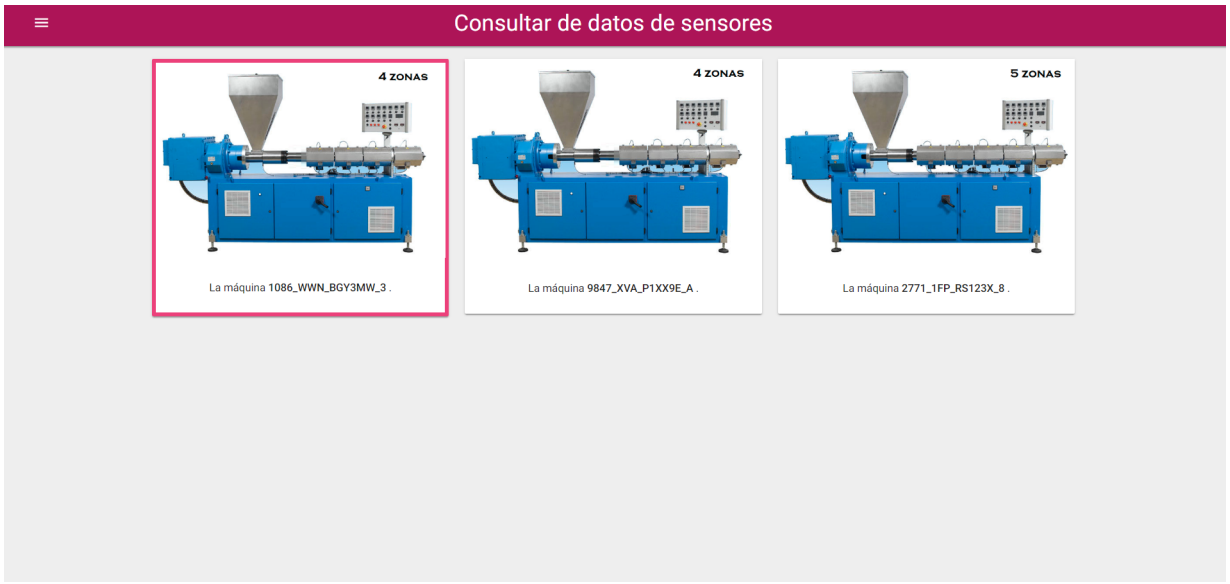

Figura 4.5: Caso de uso Consulta de Datos: selección de máquina a consultar.

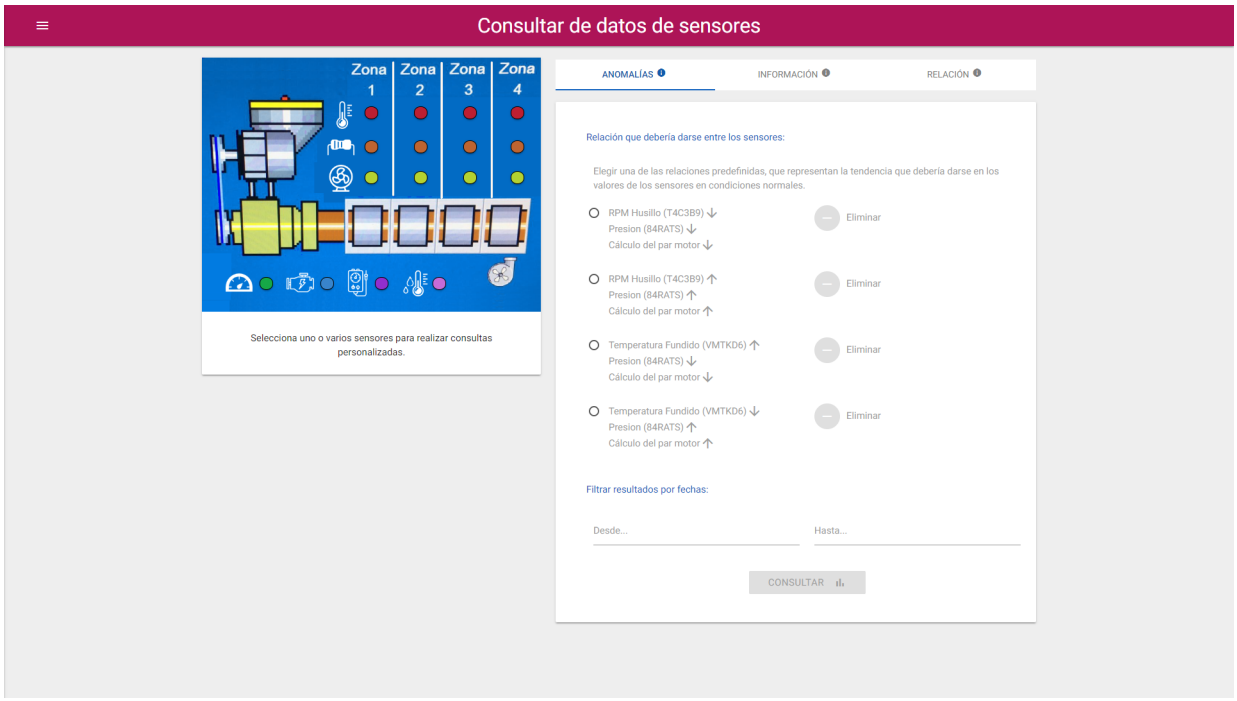

Figura 4.6: Caso de uso Consulta de Datos: formularios de consulta.

Consultar información general sobre cada sensor

El usuario selecciona el sensor o sensores sobre los que quiere consultar información y el sistema ofrece la posibilidad de personalizar dicha consulta mediante un formulario.

Flujo de eventos:

1. El usuario selecciona el sensor o sensores de los que quiere obtener la información en la imagen que representa a la extrusora y a sus sensores. (Fig. 4.7)

2. El sistema muestra al usuario un formulario mediante el cual se pueden filtrar los resultados:

   - Entre dos fechas, especificando fecha inicial y final mediante un pequeño calendario interactivo que muestra la aplicación al pulsar cada campo. (Fig. 4.8)
   - Entre dos horas del día, especificando hora inicial y final con precisión de segundo. (Fig 4.9)
   - Entre un rango de valores o dando un estado concreto, especificando mediante una barra el rango de valores seleccionado en el caso de los sensores de valores *double* y mediante un *interruptor* para indicar el estado de los sensores en el caso de sensores *booleanos*. (Fig. 4.10)
   - Haciendo agrupaciones de los resultados para mostrar el máximo, mínimo y/o el valor medio de los datos por día o por hora. (Fig. 4.11)

   Una vez completados los campos deseados, el usuario realiza la consulta.

3. El sistema recoge los datos introducidos en el formulario y genera la sentencia *SPARQL* correspondiente a dicha consulta.

4. El sistema envía dicha sentencia *SPARQL* a Virtuoso a través de una petición HTTP.

5. Cuando se reciben los resultados, el sistema trata los mismos y los muestra de manera gráfica. (Fig. 4.12)

   - En caso de no haber información en Virtuoso sobre la consulta realizada, el sistema informa de ello debidamente al usuario.

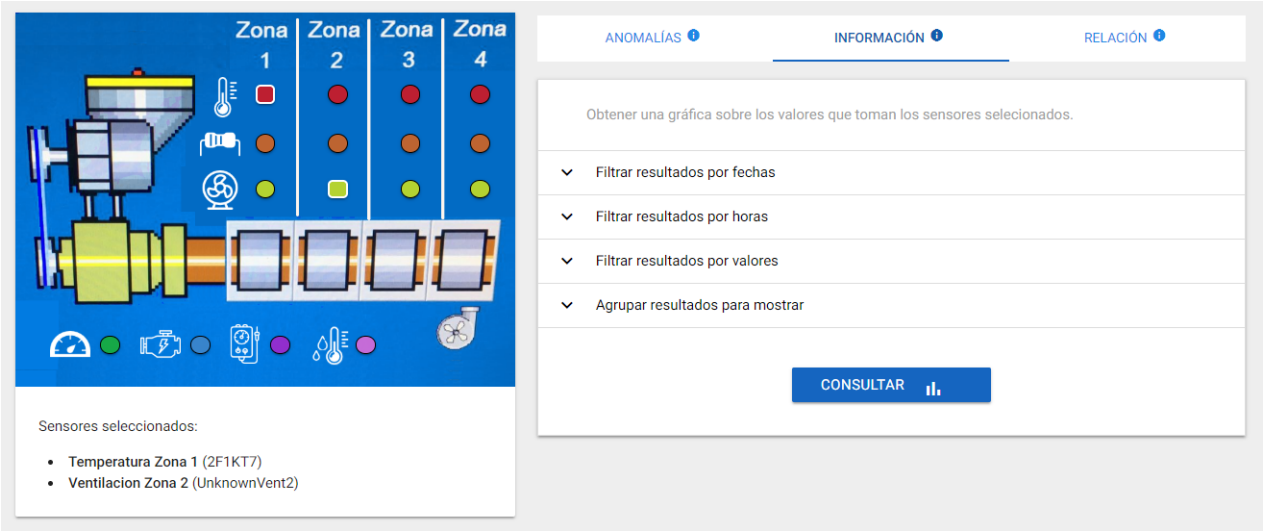

Figura 4.7: Caso de Uso Información General: seleccionar sensores

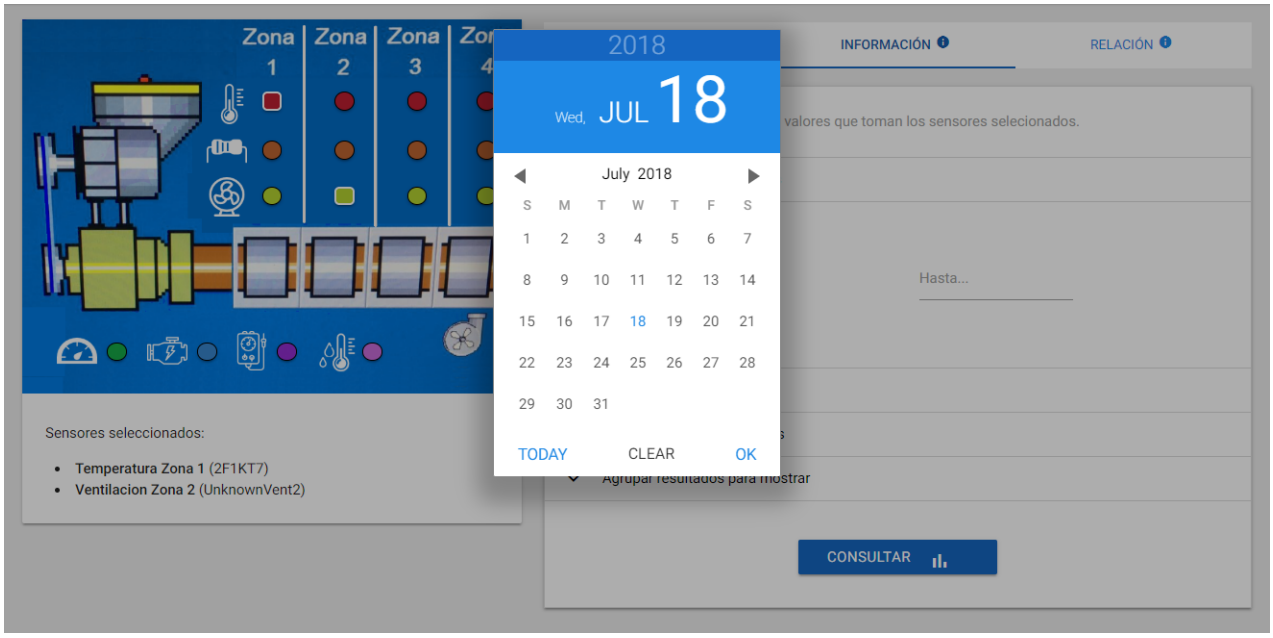

Figura 4.8: Caso de Uso Información General: seleccionar una de las fechas entre las que filtrar

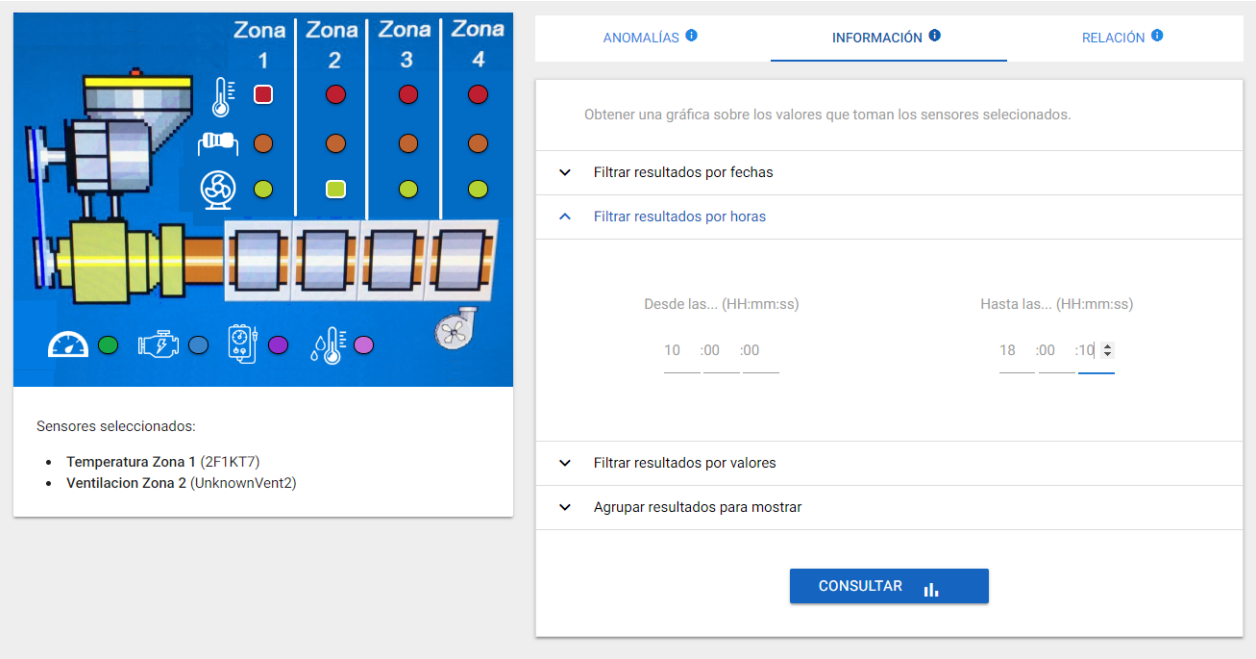

Figura 4.9: Caso de Uso Información General: filtrar entre hora

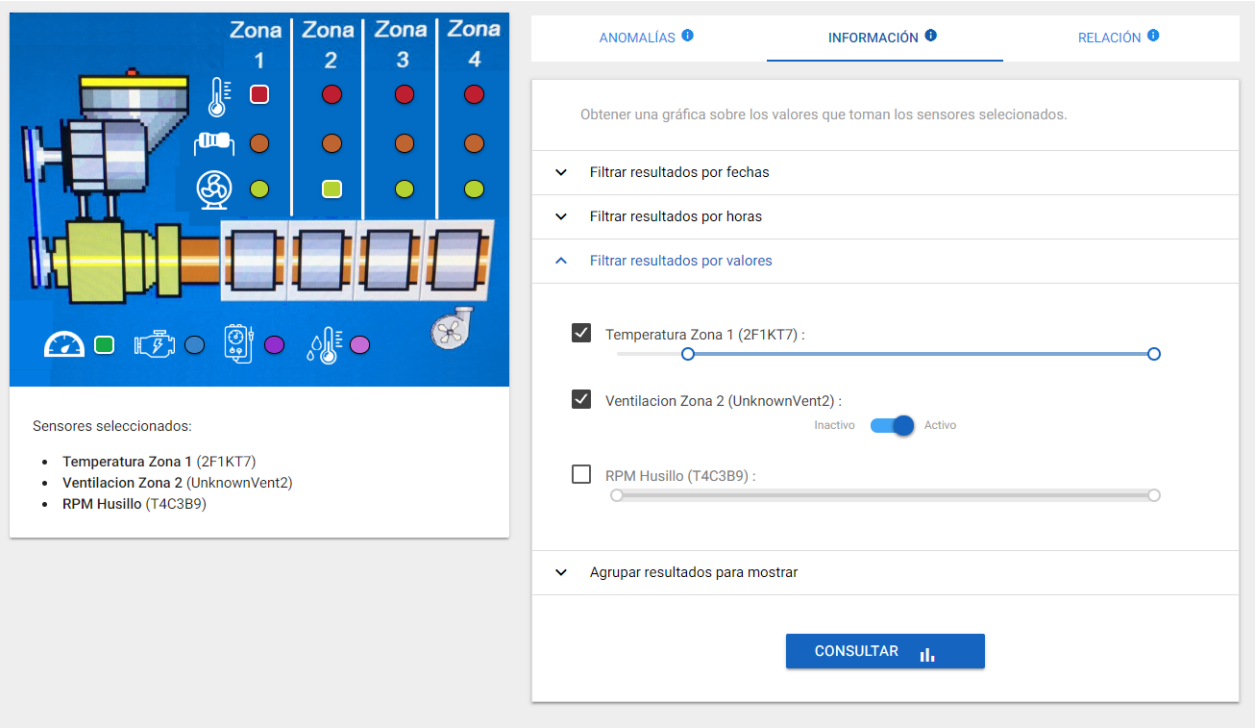

Figura 4.10: Caso de Uso Información General: filtrar los valores de los sensores

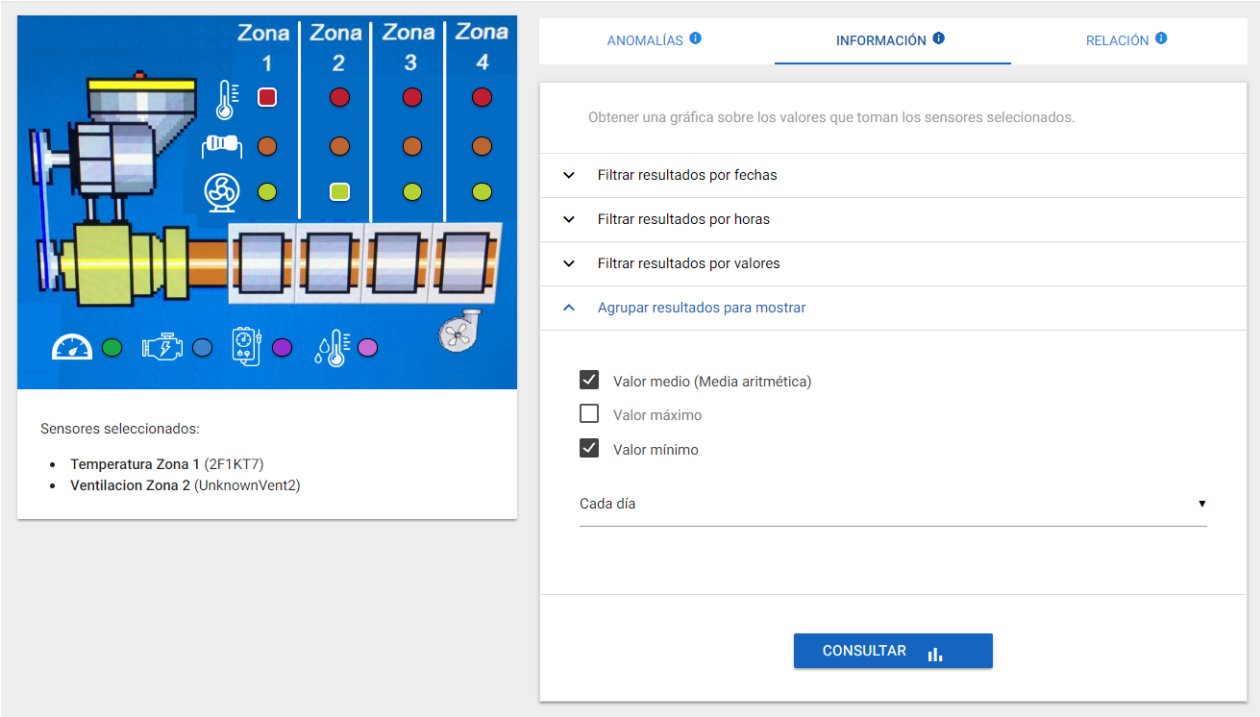

Figura 4.11: Caso de Uso Información General: crear agrupaciones

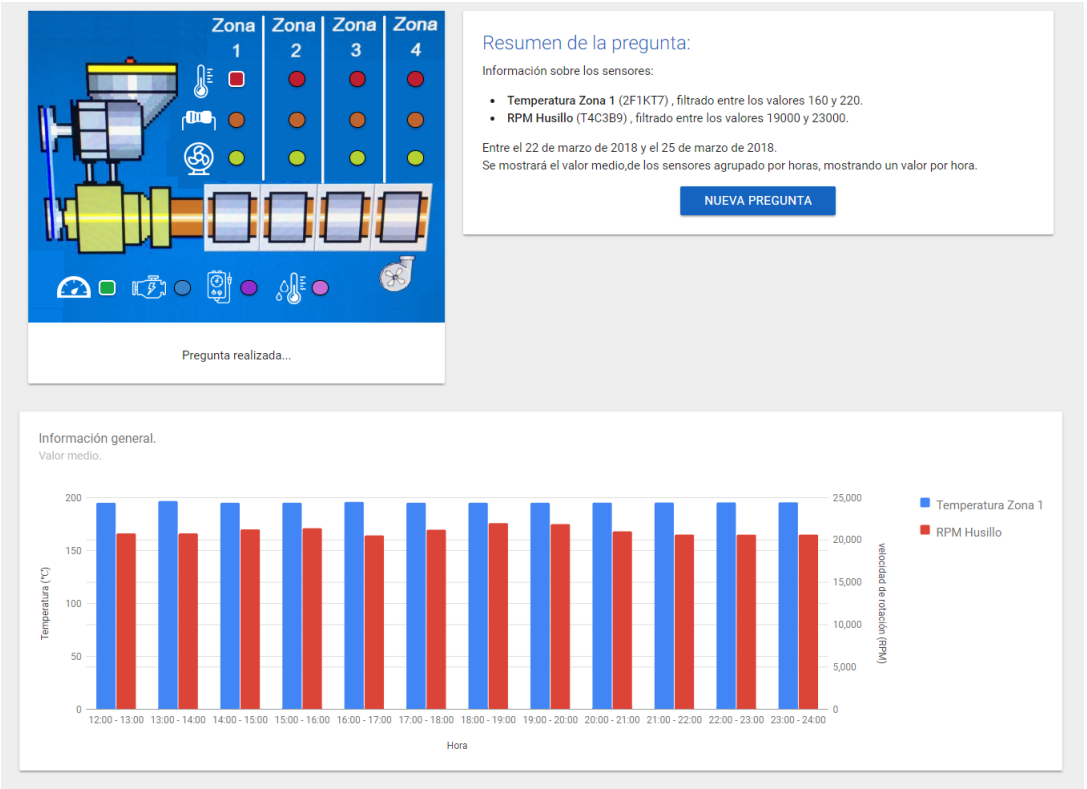

Figura 4.12: Caso de Uso Información General: resultados

Consultar relación entre sensores

El usuario selecciona los sensores (dos o más sensores) sobre los que quiere consultar la posible relación existente. Después, mediante el formulario ofrecido por el sistema, el usuario especifica la relación a consultar fijando el valor de ciertos sensores y preguntando los valores del resto.

Flujo de eventos:

1. El usuario selecciona los sensores sobre los que quiere realizar la consulta de relación. (Fig. 4.13)
2. El sistema le muestra un formulario para personalizar la consulta de relación (Fig. 4.14). Dicho formulario tiene tres partes:
   - En la primera parte el usuario selecciona los sensores cuyos valores quiere saber.
   - En la segunda parte el usuario especifica qué valores deben tomar el resto de sensores. Para ello, se puede proporcionar un valor específico (indicando el dato de manera numérica) o, alternativamente, se puede definir que el sistema calcule el valor mínimo o máximo del sensor.
     Para evitar posibles errores en los valores mínimos o máximos calculados, el usuario puede decidir el rango de valores de cada sensor que se tendrán en cuenta para dicho cálculo.
   - En la última parte se le da la opción al usuario de filtrar los datos por fechas, indicando una fecha inicial y una fecha final.
3. El sistema recoge los datos introducidos en el formulario y genera la sentencia *SPARQL* correspondiente a la consulta especificada por el usuario.
4. El sistema envía dicha sentencia *SPARQL* a Virtuoso.
5. Cuando se reciben los resultados, el sistema trata los mismos y los muestra de manera gráfica. (Fig. 4.15)
   - En caso de no haber información en Virtuoso sobre la consulta realizada, el sistema informa de ello debidamente al usuario.

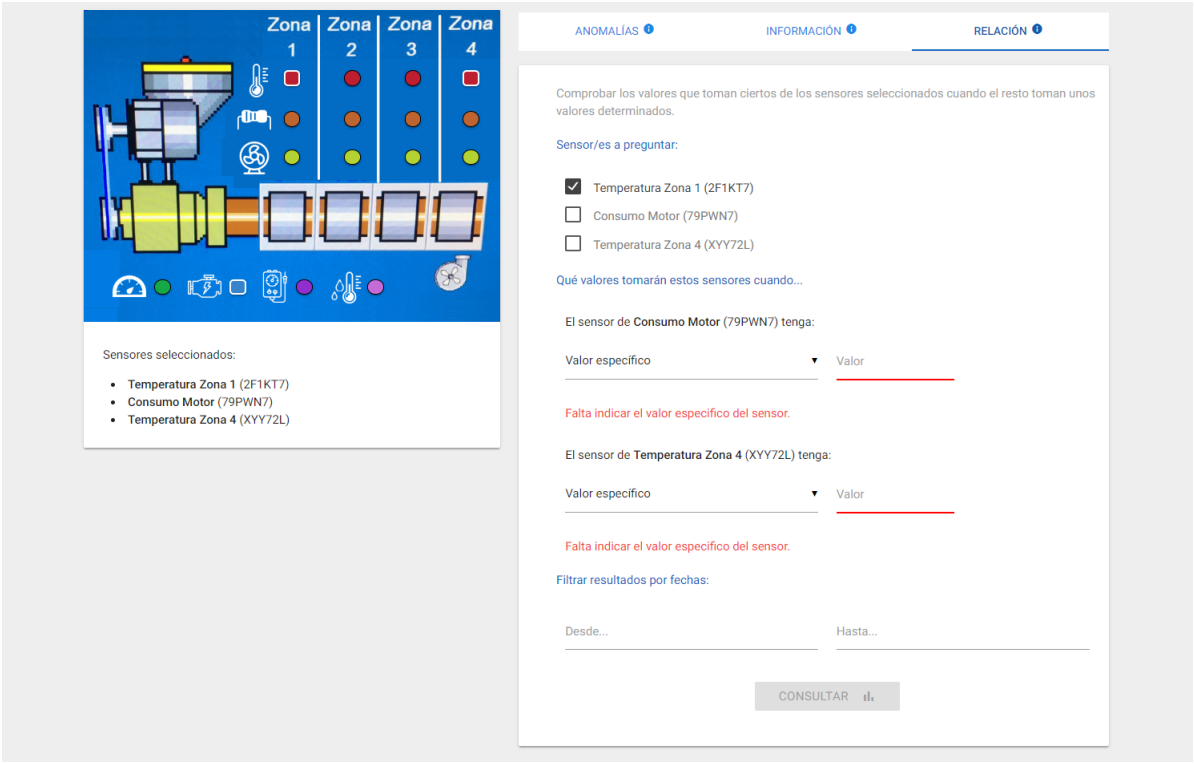

Figura 4.13: Caso de Uso Relación entre Sensores: seleccionar sensores

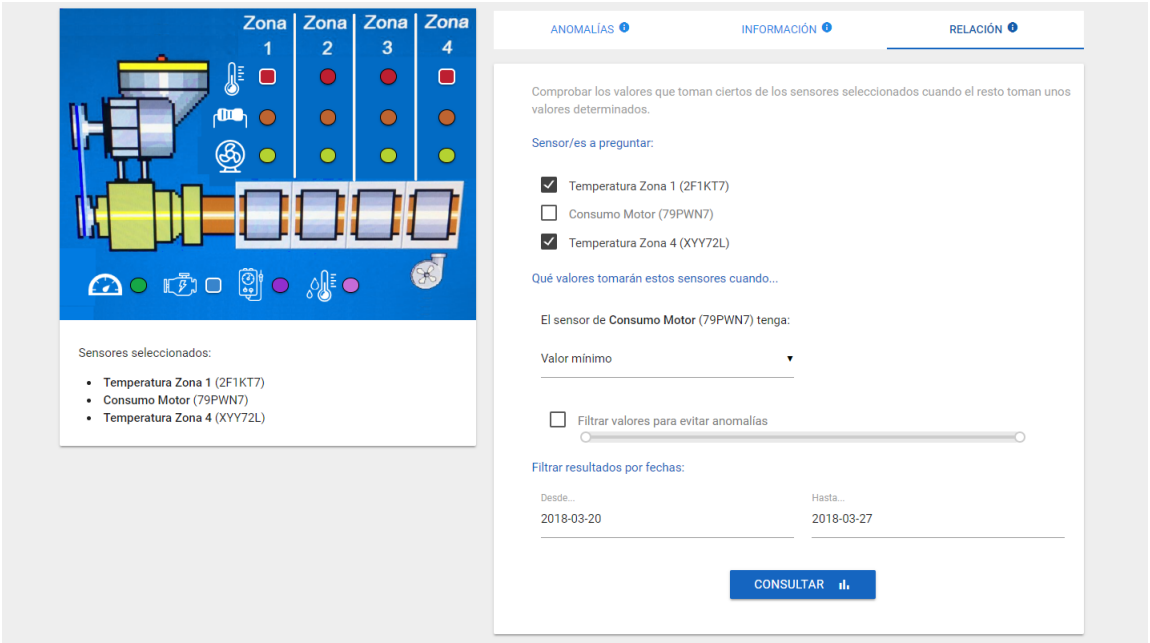

Figura 4.14: Caso de Uso Relación entre Sensores: formulario de personalización

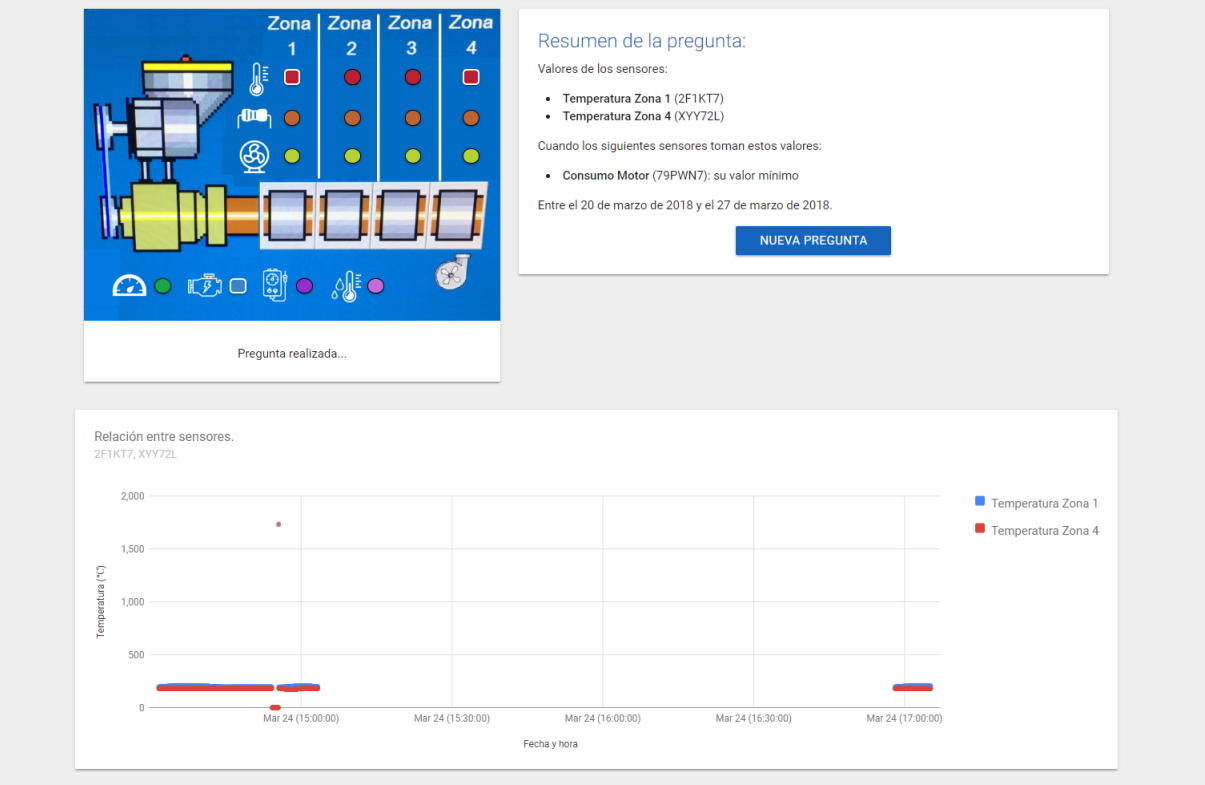

Figura 4.15: Caso de Uso Relación entre Sensores: resultados

Consultar anomalías en sensores

Sin importar el número de sensores seleccionados, el usuario puede consultar la existencia de alguna de las anomalías predefinidas en la aplicación.

Si el usuario selecciona dos o más sensores, el sistema le da también la opción de consultar anomalías personalizadas, de esta manera puede especificar la relación en la que quiere buscar anomalías mediante un pequeño formulario y a partir de los sensores seleccionados.

Flujo de eventos:

1. El sistema muestra desde un primer momento las relaciones de anomalías predefinidas anteriormente. (Fig. 4.16)

2. El usuario especifica la relación de anomalía a consultar deseada. Para ello, el usuario tiene dos posibilidades:

   - Sin importar el número de sensores seleccionados, el usuario puede seleccionar una de las relaciones predefinidas. (Fig. 4.17)
   - En caso de que el usuario seleccione dos o más sensores, el sistema ofrece también la posibilidad de especificar una relación personalizada teniendo en cuenta los sensores seleccionados.
     En caso de sensores de valores *double*, el sistema muestra unas flechas interactivas que cambian de dirección al ser pulsadas. Para los sensores de valores *booleanos* el sistema muestra un *interruptor* para decidir el estado de los mismos. (Fig. 4.18)

   En ambos casos, el usuario puede filtrar también los resultados entre dos fechas.

3. El sistema recoge los datos introducidos en el formulario y genera una sentencia *SPARQL* para obtener toda la información general sobre los sensores seleccionados, filtrando la misma entre dos fechas en caso de que el usuario las haya especificado en el formulario.

4. El sistema envía dicha sentencia *SPARQL* a Virtuoso.

5. Cuando se reciben los resultados, el sistema los analiza para buscar anomalías en los mismos, teniendo en cuenta para dicha búsqueda la relación especificada anteriormente por el usuario.

- En caso de no haber información en Virtuoso sobre la consulta realizada, el sistema informa de ello debidamente al usuario.

6. El sistema muestra los datos encontrados como anomalías en forma gráfica. (Fig. 4.19)

7. En caso de no haber encontrado anomalías en los datos de los sensores, el sistema informa de ello debidamente al usuario.

Añadir relación de anomalía predefinida

El usuario selecciona dos o más sensores y puede definir, mediante un pequeño formulario, la relación de anomalía que desee para después guardarla como predefinida. El sistema guarda los datos y la relación definida pasa a poder ser seleccionada en el formulario de anomalías predefinidas.

Flujo de eventos:

1. El usuario selecciona los sensores que son parte de la nueva relación de anomalía a definir.

2. Cuando dos o más sensores se han seleccionado, el sistema habilita la opción de personalización de la relación en el formulario. (Fig. 4.20)

3. El usuario personaliza la relación de anomalía y si dicha relación no está ya definida como predefinida, el sistema da la opción de guardarla como predefinida.

4. El usuario ejecuta la opción de añadir.

5. El sistema añade la anomalía personalizada al resto de anomalías definidas y guarda dicha información en la base de datos utilizada.

6. Una vez que la anomalía ha sido guardada, el sistema lo muestra por pantalla. (Fig. 4.21)

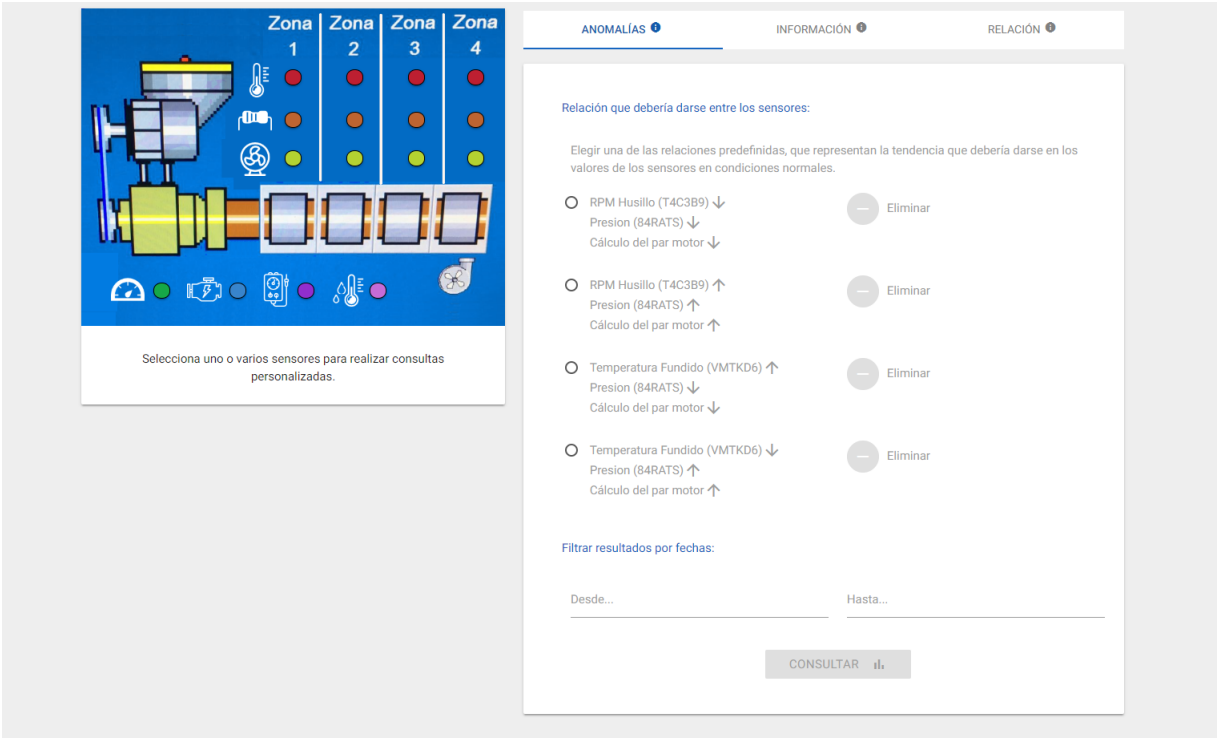

Figura 4.16: Caso de Uso Anomalías en Sensores: el sistema muestra las anomalías predefinidas.

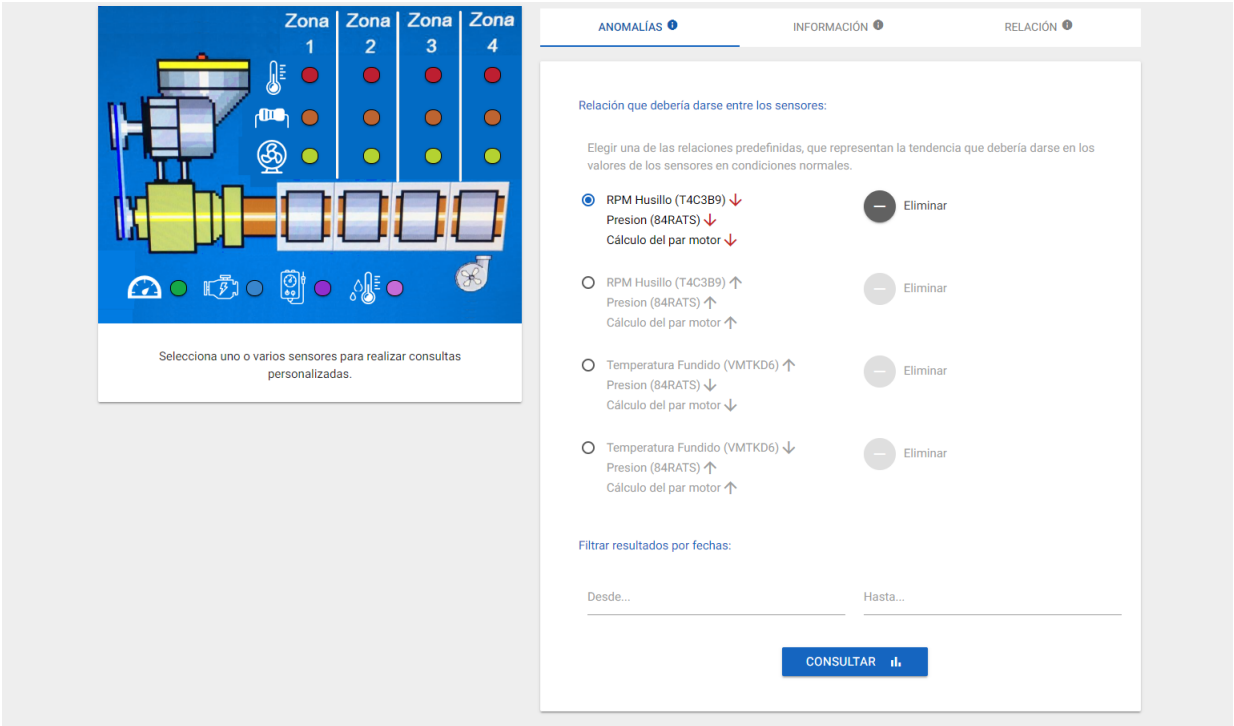

Figura 4.17: Caso de Uso Anomalías en Sensores: seleccionar anomalía predefinida.

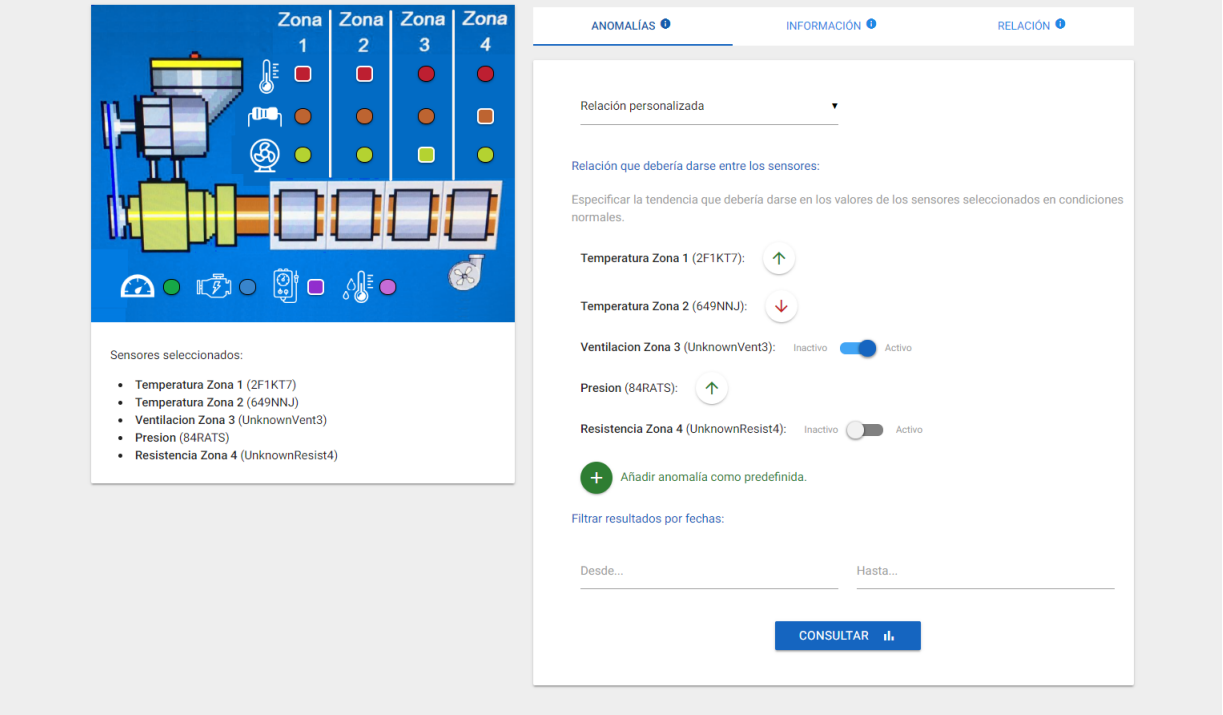

Figura 4.18: Caso de Uso Anomalías en Sensores: crear anomalía personalizada.

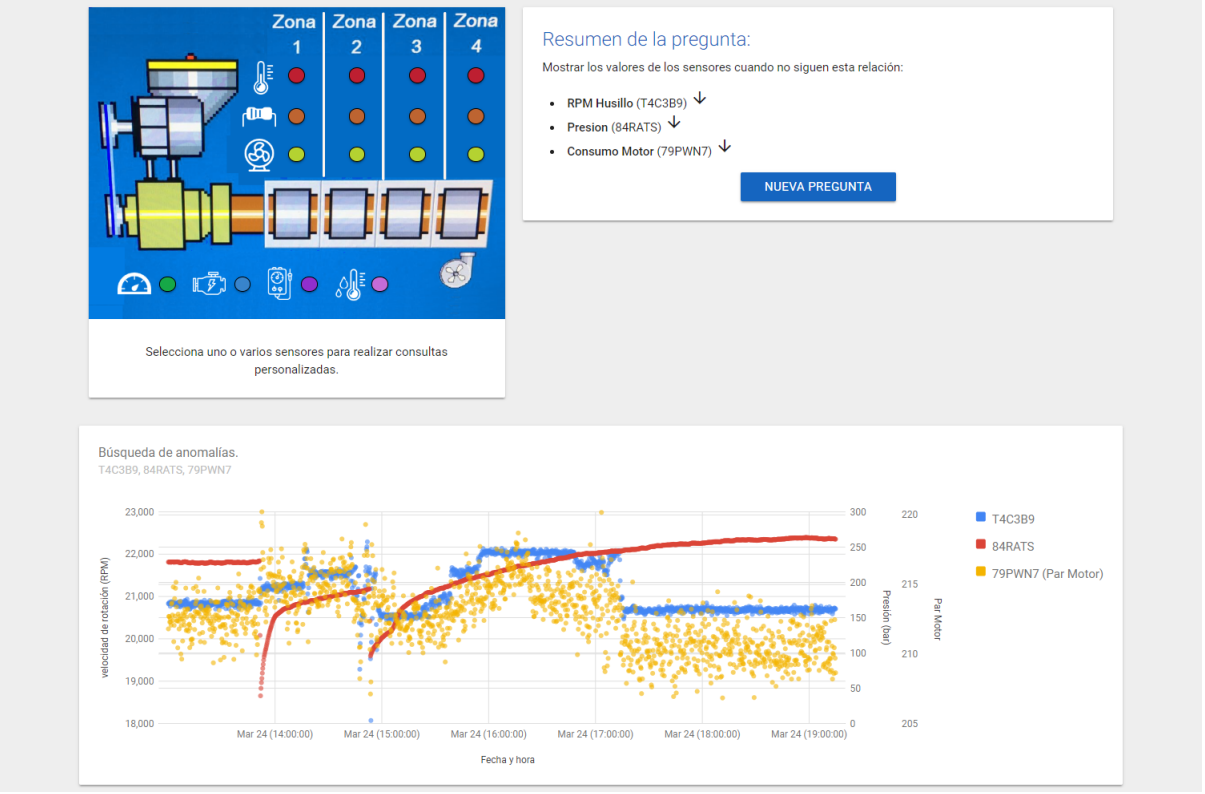

Figura 4.19: Caso de Uso Anomalías en Sensores: resultados.

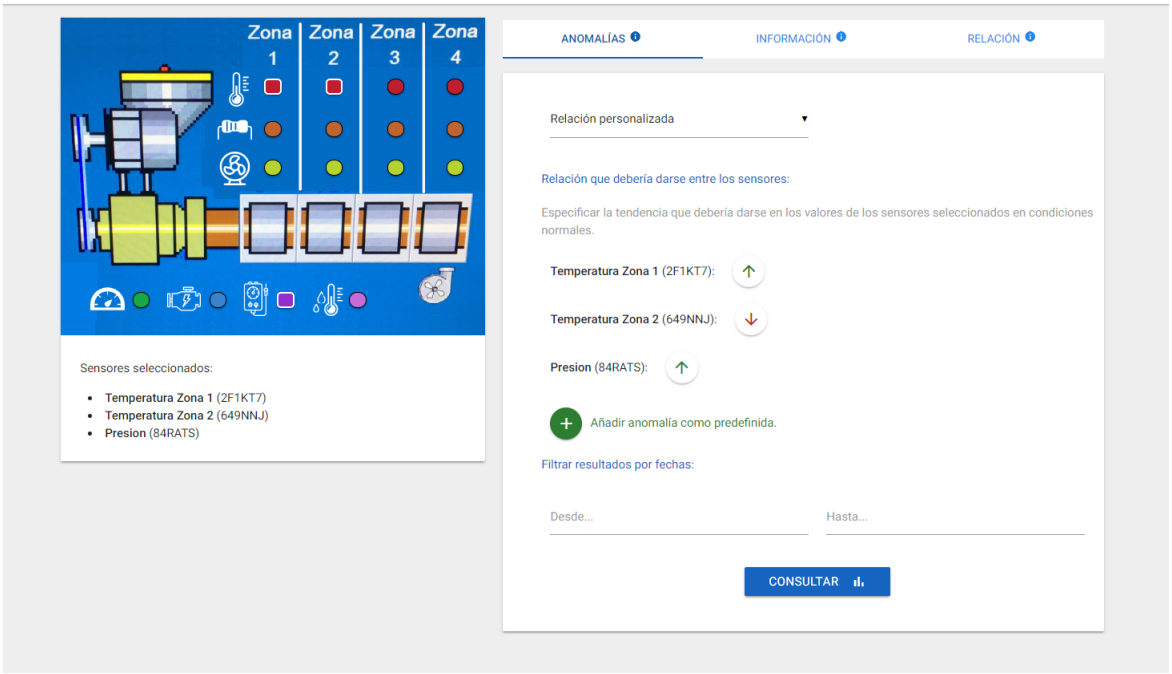

Figura 4.20: Caso de Uso Añadir Anomalía Predefinida: el usuario personaliza la nueva anomalía a añadir.

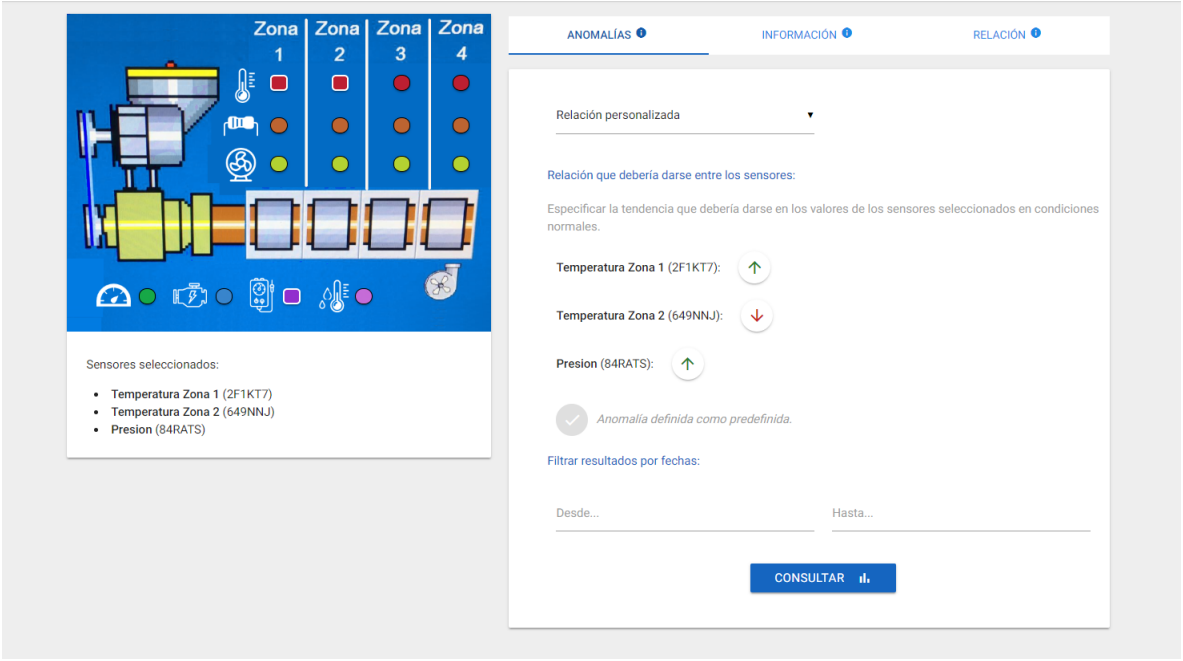

Figura 4.21: Caso de Uso Añadir Anomalía Predefinida: la nueva anomalía anomalía ha sido guardada correctamente.

Eliminar relación de anomalía predefinida

El usuario puede eliminar las relaciones de anomalía predefinidas que desee y éstas pasan a no aparecer en la lista de relaciones predefinidas.

Flujo de eventos:

1. El sistema muestra las relaciones de anomalía predefinidas actualmente asociadas a la máquina en cuestión.

2. El usuario selecciona la relación de anomalía predefinida a eliminar y ejecuta la opción de *Eliminar*. (Fig. 4.22)

3. El sistema elimina dicha anomalía predefinida y guarda la información en la base de datos, mostrando una nueva lista de anomalías predefinidas. (Fig. 4.23)

Insertar datos de sensores

El usuario selecciona la máquina a la que quiere insertar datos y el sistema muestra el formulario de inserción de datos correspondiente, mostrando información sobre los sensores que están disponibles en dicha máquina.

Flujo de eventos:

1. El sistema muestra al usuario las máquinas extrusoras disponibles según la organización a la que pertenece. (Fig. 4.24)

2. El usuario selecciona la máquina sobre la que quiere realizar las consultas. (Fig. 4.25)

3. El sistema recopila la información necesaria sobre dicha máquina y muestra al usuario el formulario para la inserción de datos. (Fig. 4.26)

    - En caso de no haber información disponible sobre la máquina seleccionada, el sistema informa al usuario y ofrece la opción de volver a escoger una máquina.

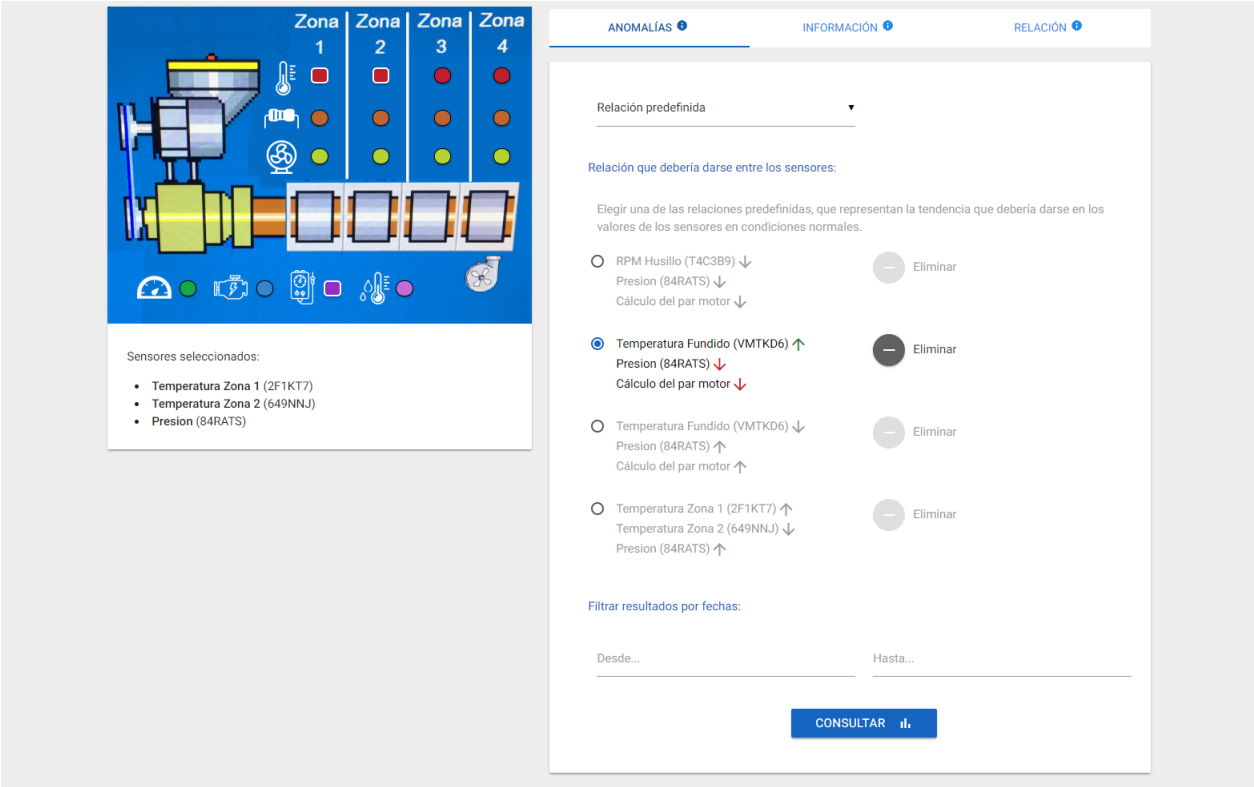

Figura 4.22: Caso de Uso Eliminar Anomalía Predefinida: el usuario selecciona la anomalía a eliminar.

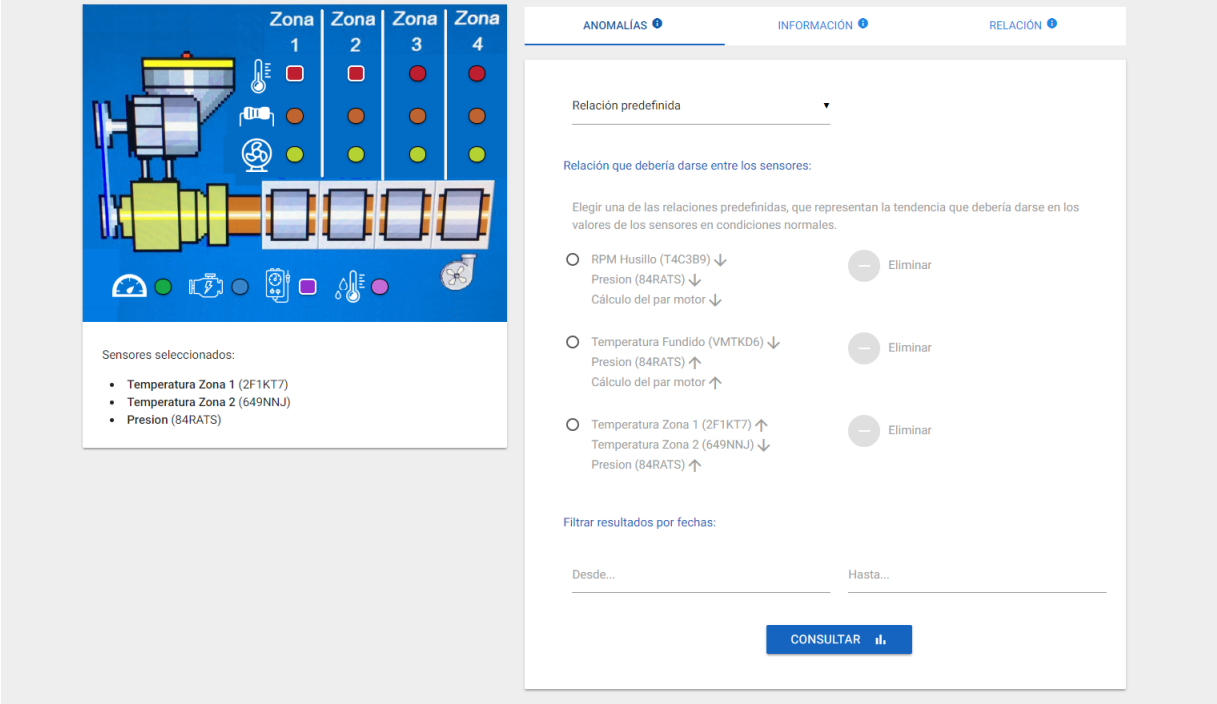

Figura 4.23: Caso de Uso Eliminar Anomalía Predefinida: nueva lista de anomalías predefinidas.

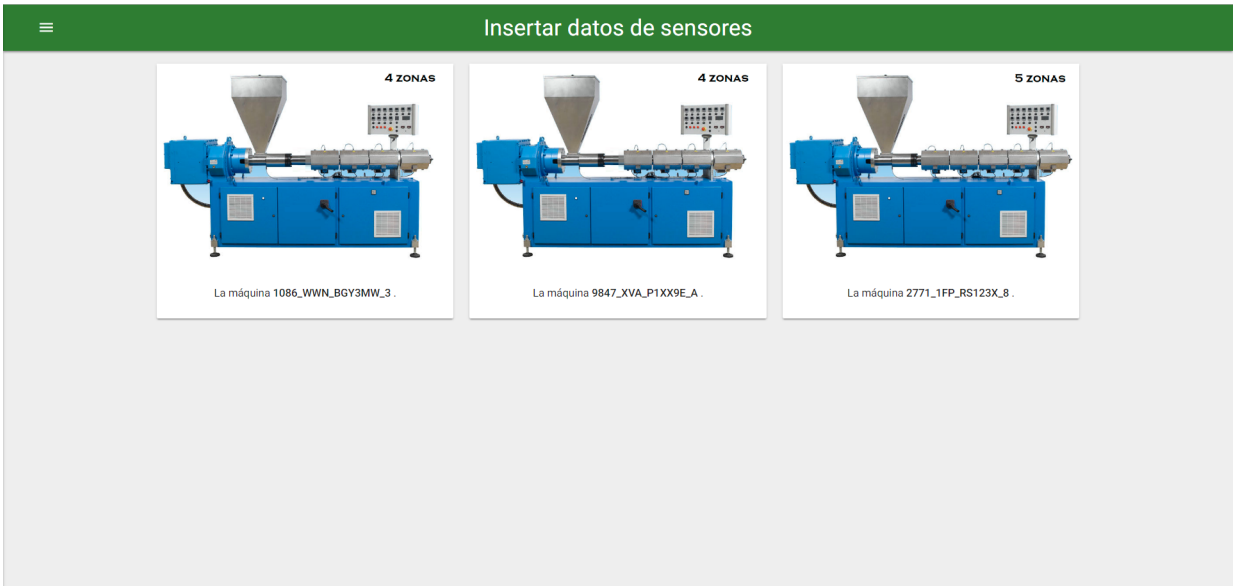

Figura 4.24: Caso de Uso Insertar Datos: máquinas disponibles en la organización.

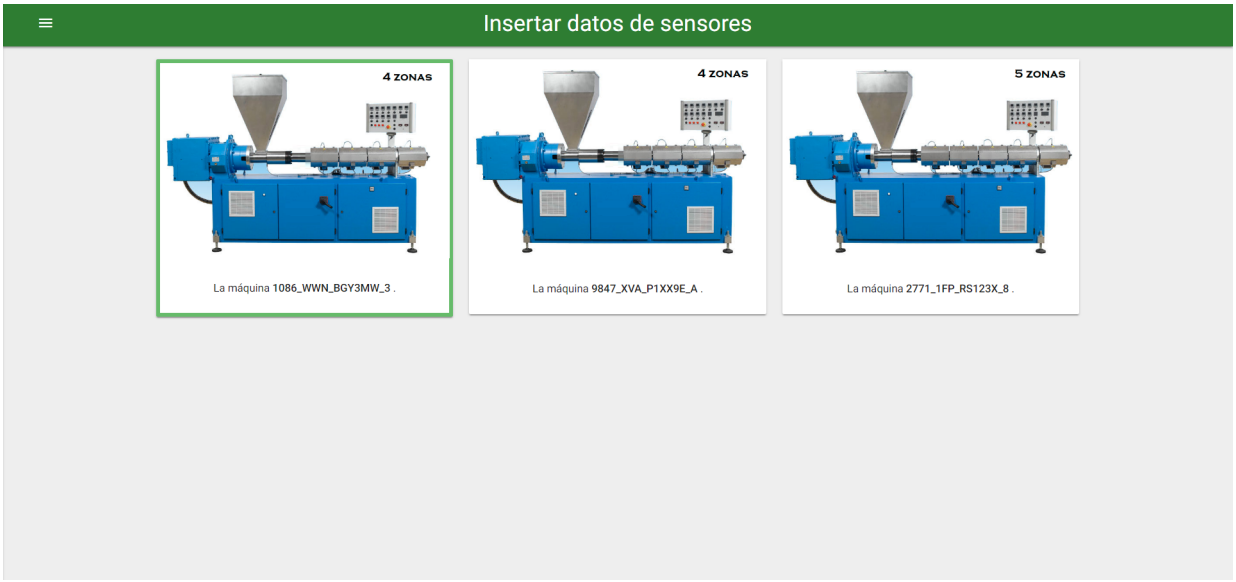

Figura 4.25: Caso de Uso Insertar Datos: selección de la máquina deseada.

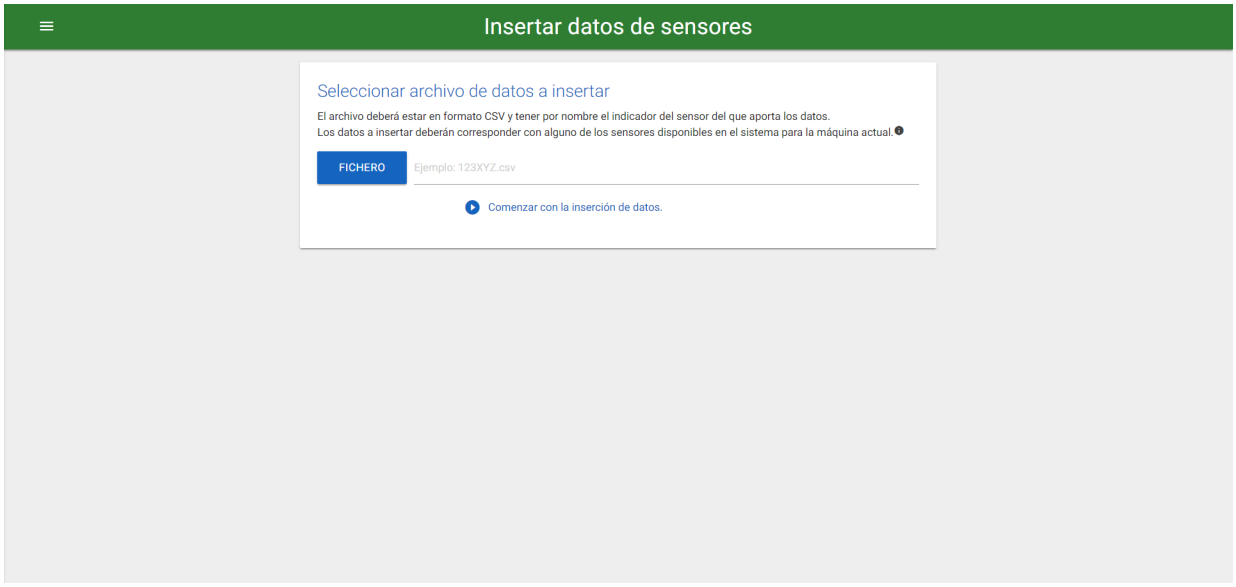

Figura 4.26: Caso de Uso Insertar Datos: formulario para la inserción de datos.

Subir datos a Virtuoso

El usuario sube un fichero de datos en formato CSV mediante un pequeño formulario y el sistema anota los datos contenidos en RDF para después insertarlos en Virtuoso.

Flujo de eventos:

1. El usuario sube el fichero con los datos a insertar. El fichero debe estar en formato CSV y contener el indicador del sensor como nombre del mismo. (Fig. 4.27)
   a) Si el usuario lo desea, puede seleccionar la opción de información para saber qué sensores están contemplados en la máquina actual.
   b) El sistema muestra dicha leyenda en el lateral del formulario. (Fig. 4.28)
2. El sistema lee los datos del fichero CSV y aplica un pequeño preprocesado a los mismos mediante el servicio ”Fix Data Servic” para corregir los datos que falten. (Fig. 4.29)
3. El sistema anota estos datos corregidos en RDF y genera sentencias SPARQL para la inserción de los mismos en Virtuoso. El sistema envía sistemáticamente dichas consultas hasta finalizar con todos los datos proporcionados por el usuario. (Fig. 4.30)
4. El sistema informa de que los datos han sido correctamente insertados en Virtuoso. (Fig. 4.31)
5. Si el usuario desea descargar un fichero con los datos generados en RDF, se extiende el caso de uso *Descargar fichero Turtle*.

Descargar fichero Turtle

El usuario puede descargar el fichero generado al anotar los datos en RDF, el cuál está definido en formato Turtle (extensión .ttl).

Flujo de eventos:

1. El sistema trata los datos anotados en RDF y genera un fichero en formato *Turtle* con los mismos.
2. El sistema descarga automáticamente a través del navegador el fichero Turtle creado.

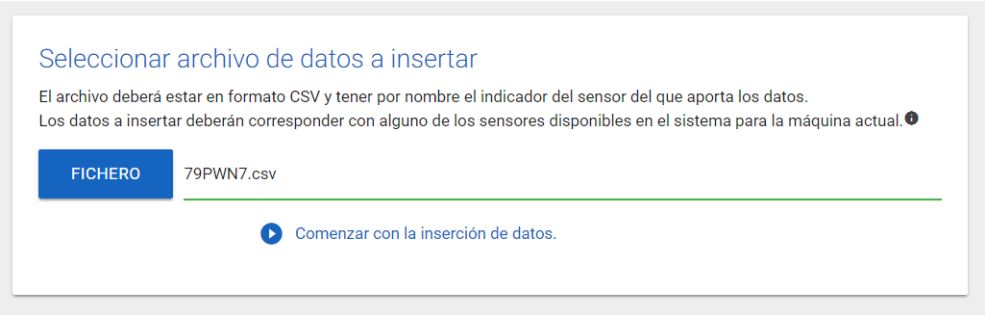

Figura 4.27: Caso de Uso Subir Datos a Virtuoso: seleccionar archivo con datos a insertar.

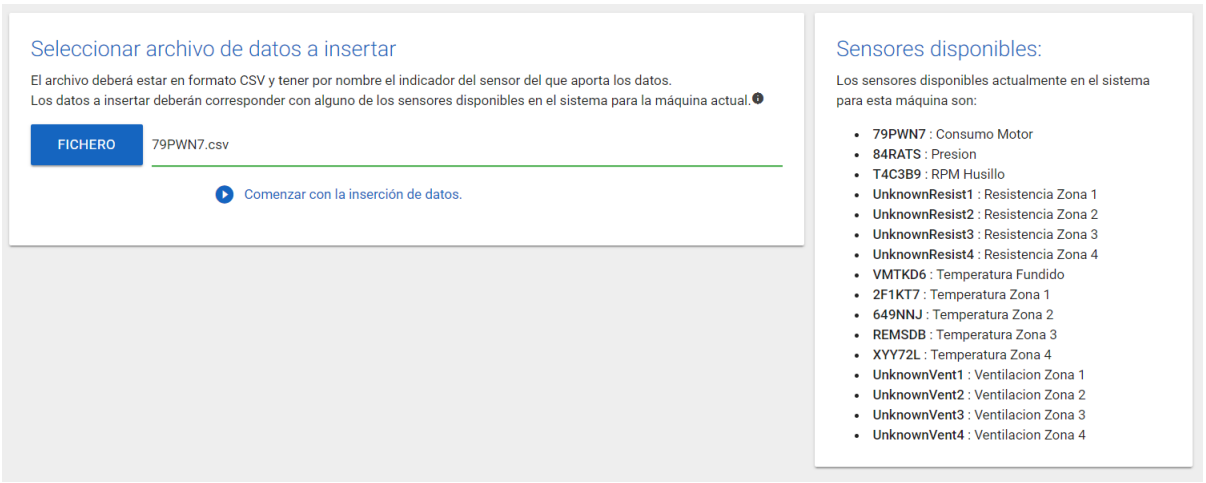

Figura 4.28: Caso de Uso Subir Datos a Virtuoso: leyenda con sensores disponibles en la máquina.

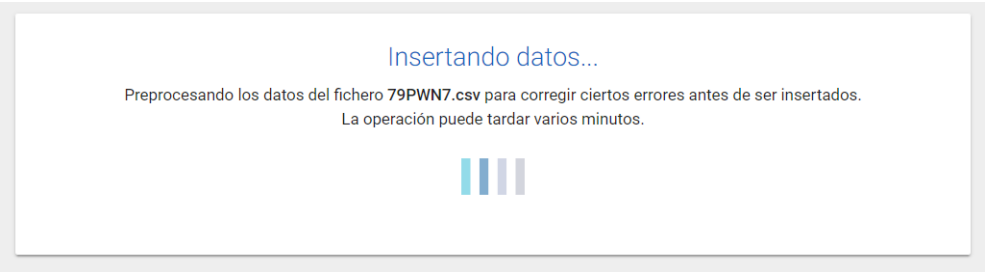

Figura 4.29: Caso de Uso Subir Datos a Virtuoso: preprocesado de los datos contenidos en el fichero.

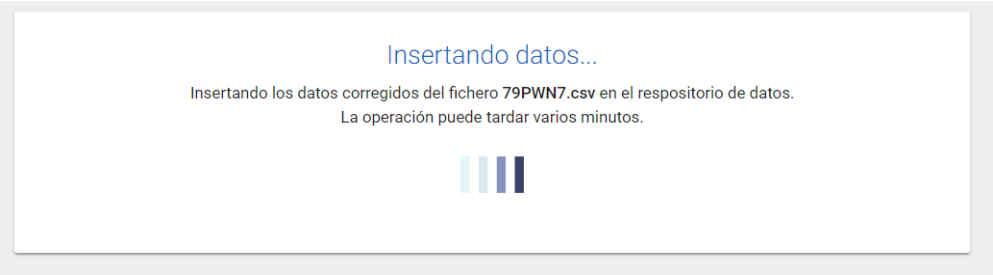

Figura 4.30: Caso de Uso Subir Datos a Virtuoso: inserción de datos preprocesados en Virtuoso.

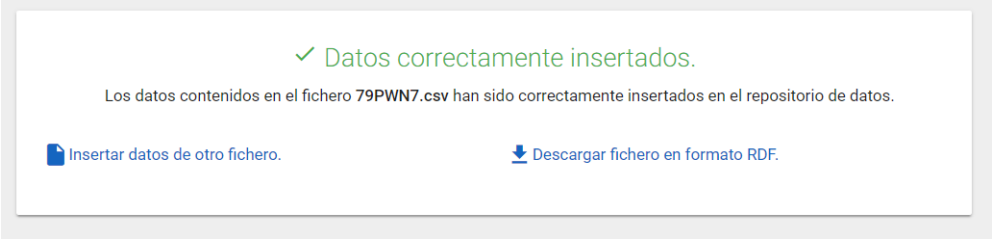

Figura 4.31: Caso de Uso Subir Datos a Virtuoso: datos correctamente insertados.

4.4. Modelo de dominio

El modelo de dominio que se presenta a continuación se ha desarrollado en base a las características de la máquina extrusora *Extrusora de Cuatro Zonas* fabricada por Urola.

De los diferentes sensores presentes en esta máquina, se han tenido en cuenta aquellos de los que más información y datos se posee, así como los que más información aportan sobre el proceso llevado a cabo en la misma.

Dichos sensores son:

- **Sensores de temperatura**, uno por cada zona presente en la máquina. Este tipo de sensores devolverán datos numéricos que expresan la temperatura de la extrusora en la zona correspondiente en grados *Celsius* o grados centígrados (ºC).

- **Sensores del estado de la ventilación**, uno por cada zona presente en la máquina. Se tratan de sensores que devuelven datos *booleanos*[6] que determinan el estado de la ventilación en la zona correspondiente: activada o desactivada.

[6]El tipo de dato *booleano* o lógico representa valores de lógica binaria.

- **Sensores del estado de la resistencia**, uno por cada zona presente en la máquina. Se tratan de sensores que devuelven datos *booleanos* que determinan el estado de la resistencia en la zona correspondiente: activada o desactivada.
- **Sensor de temperatura de fundido**. Este sensor devuelve datos numéricos que representan la *temperatura de fundido*[7] del material en grados *Celsius* o grados centígrados (ºC).
- **Sensor de revoluciones por minuto del husillo**. Este sensor devuelve datos numéricos que representan las RPM del *husillo*[8] presente en la máquina.
- **Sensor de consumo del motor**. Este sensor devuelve datos numéricos que representan el consumo del motor que provoca el giro del husillo de la máquina, medido en amperios (A).
- **Sensor de presión de fundido**. Este sensor devuelve datos numéricos representando la presión del fundido del material, medido en bares (bar).

A partir de esta información, se ha determinado el dominio del sistema web actual así como el modelo de datos que se presentará en el siguiente capítulo (sección 5.1).

En la figura 4.32 se muestra el diagrama del modelo de dominio del proyecto, donde se muestran los datos que utilizará nuestro sistema web así como la relación existente entre los mismos.

Al igual que en la sección anterior, en dicho modelo se han utilizado dos colores para diferenciar la distribución de los datos:

- En **magenta** se definen los datos que estarán definidos en el sistema web final resultante. Estos datos son proporcionados por la ya mencionada plataforma I4TSPS por lo que su definición no está en el alcance de este TFG.
- En **negro** se definen los datos específicos que van a ser definidos para los módulos desarrollados en este TFG. Estos datos se representarán en una ontología cuya estructura explicaremos en el capítulo siguiente y estarán almacenados en *Virtuoso*.

A continuación se detallan los datos correspondientes a la parte de las máquinas extrusoras así como la relación entre ellos.

[7] La *temperatura de fundido* hace referencia a la temperatura del material (plástico en este caso) justo antes de salir al molde donde será soplado.

[8] *Husillo*: Tornillo de hierro o madera que se usa para el movimiento de las prensas y otras máquinas.

- **Extrusora**: Esta clase representa las máquinas extrusoras implantadas en las distintas empresas, con su correspondiente identificador y nombre.

- **Sensor**: Representa cada uno de los sensores que contienen las diferentes extrusoras, y cuentan con un identificador, un nombre y la especificación de en qué zona de la máquina están presentes. Éstos, por los valores que capturan, podrán ser diferenciados en dos tipos:

  - ***SensorDeValor***: Representa a aquellos sensores que devuelven un valor de tipo *double*[9]. En este tipo de sensores, además de las características heredadas de la clase Sensor, también se definen su valor mínimo y su valor máximo habituales. Estos valores nos ayudarán a identificar los posibles *outliers*[10] en los datos capturados por los sensores.
  - ***SensorBooleano***: Representa a aquellos sensores que devuelven un valor de tipo *booleano* para la representación del estado de cierto componentes de la máquina. Por ello, en ellos también se debe especificar el componente de la máquina concreto que observan.

- **PropiedadObservada**: Refleja las distintas propiedades que pueden observar los sensores. Éstas, al igual que los tipos de sensores, las podemos diferenciar en dos tipos:

  - ***Cuantitativa***: Como el propio nombre indica, representa aquellas propiedades que pueden ser cuantificadas, como lo pueden ser la temperatura, la presión, la velocidad, etc. Esta clase, aparte de las características heredadas a partir de PropiedadObservada, también debe tener información sobre la unidad de medida utilizada para dicha propiedad.
  - ***Booleana***: Por el contrario, esta entidad representa a aquellas propiedades que no pueden ser cuantificadas, como por ejemplo el estado operacional de los componentes, la existencia de errores, etc. Para poder entender el resultado de este tipo de observaciones, en esta clase se indica también el significado de cada estado que se puede obtener.

- **Observación**: Esta entidad refleja las distintas observaciones realizadas por cada sensor, las cuales estarán siempre compuestas por el valor devuelto por el sensor

[9] El tipo de dato *double* hace referencia a números IEEE de punto flotante de doble precisión de 64 bits (8 bytes) que almacenan aproximaciones de números reales.

[10] Los *outliers* o valores atípicos en observaciones hacen referencia a aquellos datos numéricamente distantes del resto, lo que generalmente conlleva un posible error en la observación.

y un *timestamp* o marca de tiempo para representar el momento en el que dicho valor ha sido tomado. El tipo del valor de cada observación sin embargo, varía dependiendo del tipo de sensor que las realiza, por lo que dichos valores se definen en las dos especializaciones que se realizan de la clase:

- ***ObservacionDeValor***: Representa todas aquellas observaciones que tengan como valor un tipo de dato *double*.
- ***ObservacionBooleana***: Representa todas aquellas observaciones que tengan como valor un tipo de dato *booleano*.

En cuanto a la relación entre las clases, una **extrusora** tendrá implantados uno o más sensores, y un **sensor** pertenecerá únicamente a una extrusora.

Cada **sensor de valor** observará una única propiedad cuantitativa, pero una misma **propiedad cuantitativa** podrá ser objetivo de observación de diferentes sensores de valor.

De la misma manera, cada **sensor booleano** observará una única propiedad booleana, pero una misma **propiedad booleana** podrá ser objetivo de observación de diferentes sensores booleanos.

Por otro lado, un **sensor de valor** podrá realizar una o más observaciones de valor, pero una **observación de valor** pertenecerá únicamente a un sensor de valor.

Y por último, un **sensor booleano** podrá realizar una o más observaciones booleanas, pero una **observación booleana** pertenecerá únicamente a un sensor booleano.

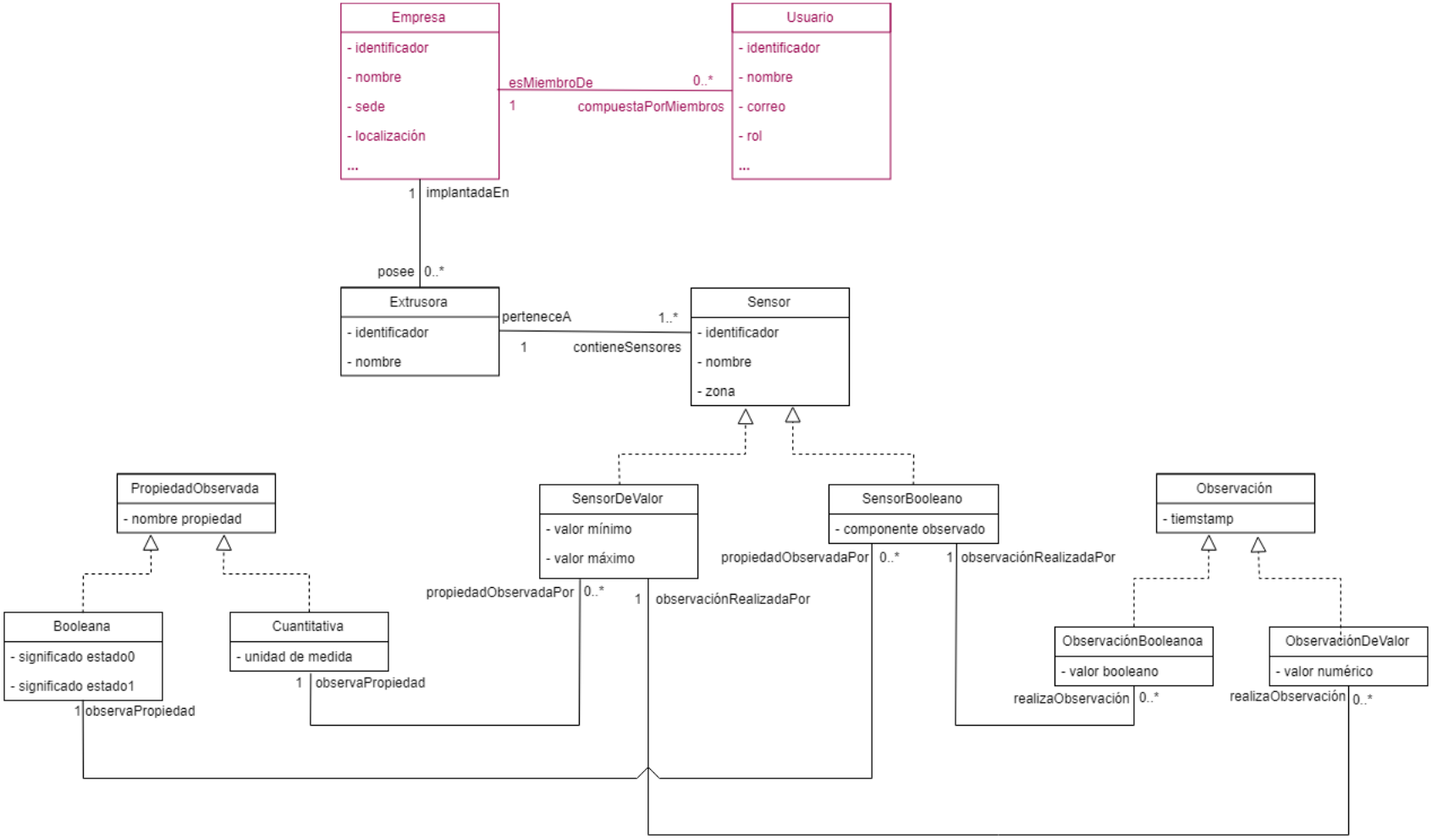

Figura 4.32: Modelo de Dominio

CAPÍTULO 5

Análisis y Diseño

Las tareas de análisis y diseño de un proyecto son una parte fundamental del mismo, donde se definen las características que después serán desarrolladas.

En este capítulo, por un lado, se detalla el diseño del modelo de representación de datos manejados en el proyecto, el cual se corresponde con la ontología desarrollada para la descripción de los sensores de la *Extrusora de Cuatro Zonas* de Urola Solutions.

Por otro lado, se explica el proceso de diseño de las diferentes interfaces de la aplicación llevado a cabo antes de pasar al desarrollo de las mismas.

Por último, en este capítulo también se detalla el diseño de la arquitectura final del sistema web, explicándola en conjunto con el sistema I4TSPS para poder visualizar el resultado final de la unión de ambos sistemas.

5.1. Modelo de representación de datos

Como bien se ha explicado anteriormente, en este TFG se ha explorado el potencial de uso de las técnicas semánticas para el análisis y almacenamiento de los datos, y para ello se ha desarrollado una ontología[1] que nos ayude a representar semánticamente el dominio a analizar.

[1] La ontología *sensors* desarrollada: http://bdi.si.ehu.es/bdi/ontologies/extrusion/sensors

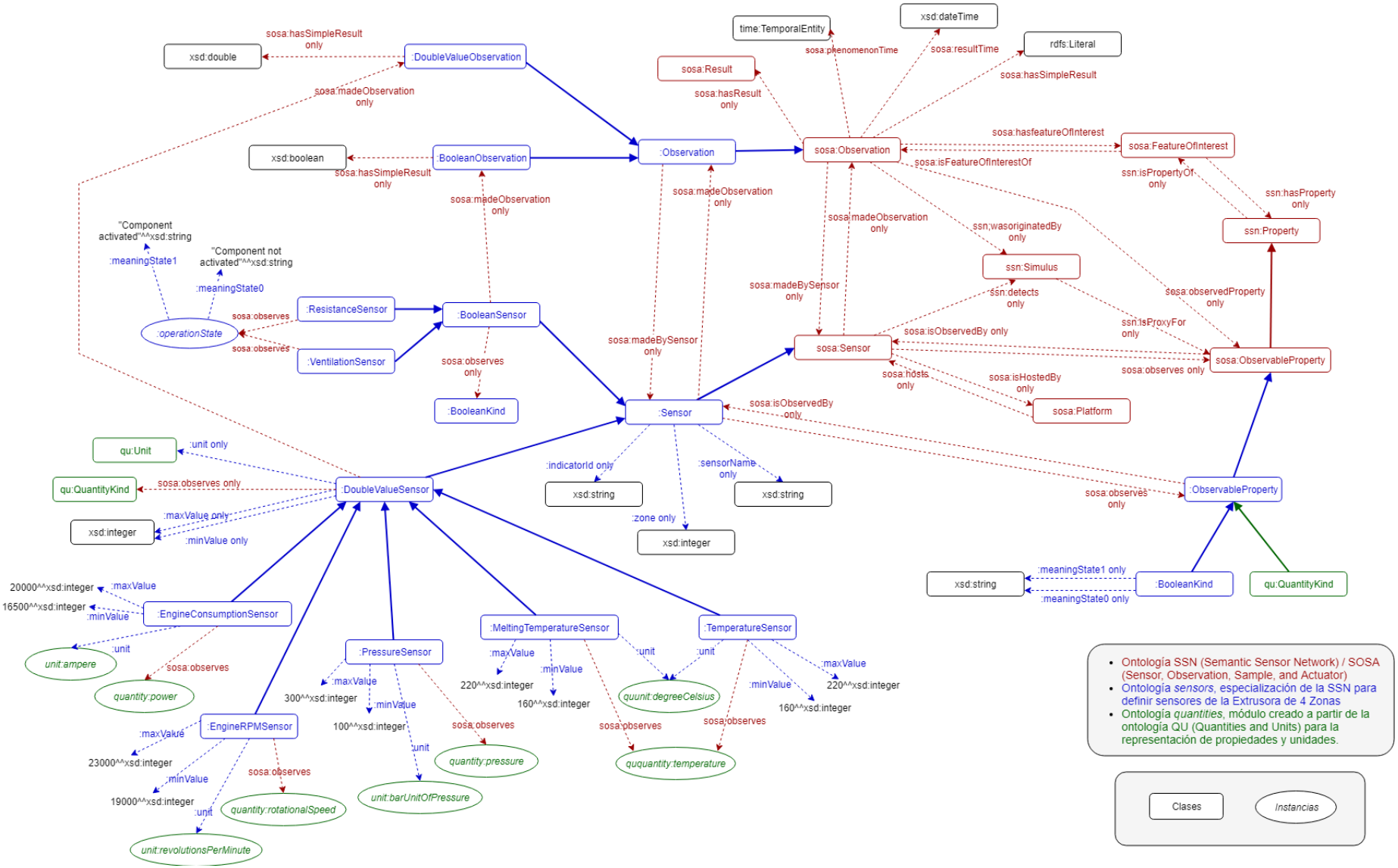

Figura 5.1: Diagrama de la ontología final desarrollada

En esta sección se detalla dicha ontología, explicando las clases y propiedades que aparecen en la misma. El proceso de desarrollo de la ontología en cuestión será abordado más adelante (concretamente en la sección 6.1), explicando la metodología utilizada para dicho desarrollo y el porqué de todas las decisiones tomadas para conseguir el resultado final.

En la figura 5.1 se presenta el diagrama final de la ontología desarrollada. En dicho diagrama se utilizan diferentes colores para representar la procedencia de las clases, instancias y propiedades de dato y objeto. Además, también se utilizan diferentes formas para diferenciar las clases de las instancias. Esta información está detallada en la leyenda que aparece en la propia figura.

Dicha ontología importa la ontología *Semantic Sensor Network (SSN)*[2], cuyo núcleo está compuesto por la ontología *Sensor, Observation, Sample, and Actuator (SOSA)*[3]. Una parte fundamental la SSN se centra en la definición de los sensores, sus observaciones y las propiedades que éstos observan (Fig. 5.2), de manera que en este TFG se ha hecho una especialización de dichas clases para representar los sensores concretos que componen la máquina extrusora a representar.

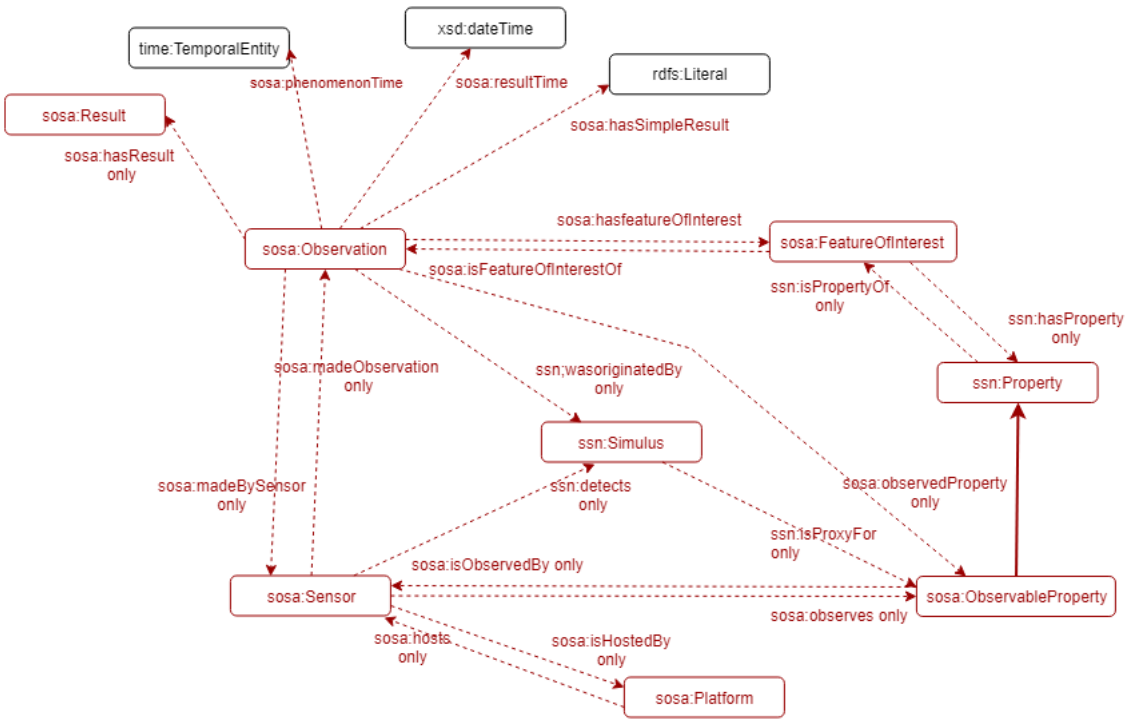

Figura 5.2: Parte del diagrama general de la ontología SSN/SOSA

[2]La ontología *SSN*: https://www.w3.org/ns/ssn/
[3]La ontología *SOSA*: https://www.w3.org/ns/sosa/

Empezando por los sensores, se ha hecho una especialización de la clase **sosa:Sensor** para la representación de los tipos de sensores concretos que forman parte del dominio de este TFG (Fig. 5.3).

Para ello, primero se ha creado una subclase de la misma, **:Sensor**, la cual representa a los sensores específicos de nuestro dominio. Esto además ayuda a mantener una clara separación entre los objetos importados desde la ontología SSN y los objetos y restricciones nuevas añadidas para este dominio concreto.

Esta clase :Sensor, por un lado se ha definido como dominio[4] de tres propiedades de dato nuevas:

- *:indicatorId*. Representa el identificador de cada sensor, el cual está compuesto por una cadena de caracteres, por lo que su rango[5] es de tipo xsd:string.
- *:zone*. Representa la zona numérica de la máquina extrusora en la que se encuentra el sensor, por lo que su rango es de tipo xsd:integer.
- *:sensorName*. Representa el nombre significativo del sensor, el cual está compuesto por una cadena de caracteres, por lo que su rango es de tipo xsd:string.

Además, se han definido dos subclases principales a partir de dicha clase :Sensor:

- **:BooleanSensor**. Representa a los sensores que capturan datos *booleanos*, correspondiéndose con la clase "SensoresBooleanos"del modelo de dominio. En la extrusora concreta a modelar encontramos dos sensores de este tipo, por lo que se han definido también dos subclases de :BooleanSensor:
  - **:ResistanceSensor**. Hace referencia a los sensores que observan el estado de la resistencia.
  - **:VentilationSensor**. Hace referencia a los sensores que observan el estado del ventilador.

[4]El *dominio* de una propiedad de dato u objeto hace referencia al tipo de dato u objeto admitido como sujeto de la relación.

[5]El *rango* de una propiedad de dato u objeto hace referencia al tipo de dato u objeto admitido como objeto de la relación.

- **:DoubleValueSensor**. Representa a los sensores que capturan datos de tipo *double*, correspondiéndose con la clase "SensoresDeValor"del modelo de dominio. Esta clase se ha definido como dominio de dos nuevas propiedades de dato:

  - *:minValue*. Representa el valor mínimo que los sensores tomarán en condiciones normales, por lo que su rango es únicamente de tipo xsd:integer.

  - *:maxValue*. Representa el valor máximo que los sensores tomarán en condiciones normales, por lo que su rango es únicamente de tipo xsd:integer.

  En la extrusora concreta a modelar encontramos cinco sensores de este tipo, por lo que se han definido cinco subclases a partir de :DoubleValueSensor. En éstas también se definen los valores concretos de las propiedades *:minValor* y *:maxValor* para cada sensor:

  - **:EngineConsumptionSensor**. Hace referencia a los sensores que observan el consumo del motor. En esta clase, *:minValor* tiene como valor 16500 y *:maxValor* tiene como valor 20000, ambos de tipo xsd:integer.

  - **:EngineRPMSensor**. Hace referencia a los sensores que observan las RPM del husillo del motor. En esta clase, *:minValor* tiene como valor 19000 y *:maxValor* tiene como valor 23000, ambos de tipo xsd:integer.

  - **:PressureSensor**. Hace referencia a los sensores que observan la presión en la máquina. En esta clase, *:minValor* tiene como valor 100 y *:maxValor* tiene como valor 300, ambos de tipo xsd:integer.

  - **:TemperatureSensor**. Hace referencia a los sensores que observan la temperatura en cada zona de la máquina. En esta clase, *:minValor* tiene como valor 160 y *:maxValor* tiene como valor 220, ambos de tipo xsd:integer.

  - **:MeltingTemperatureSensor**. Hace referencia a los sensores que observan la temperatura de fundido de la máquina. En esta clase, *:minValor* tiene como valor 160 y *:maxValor* tiene como valor 220, ambos de tipo xsd:integer.

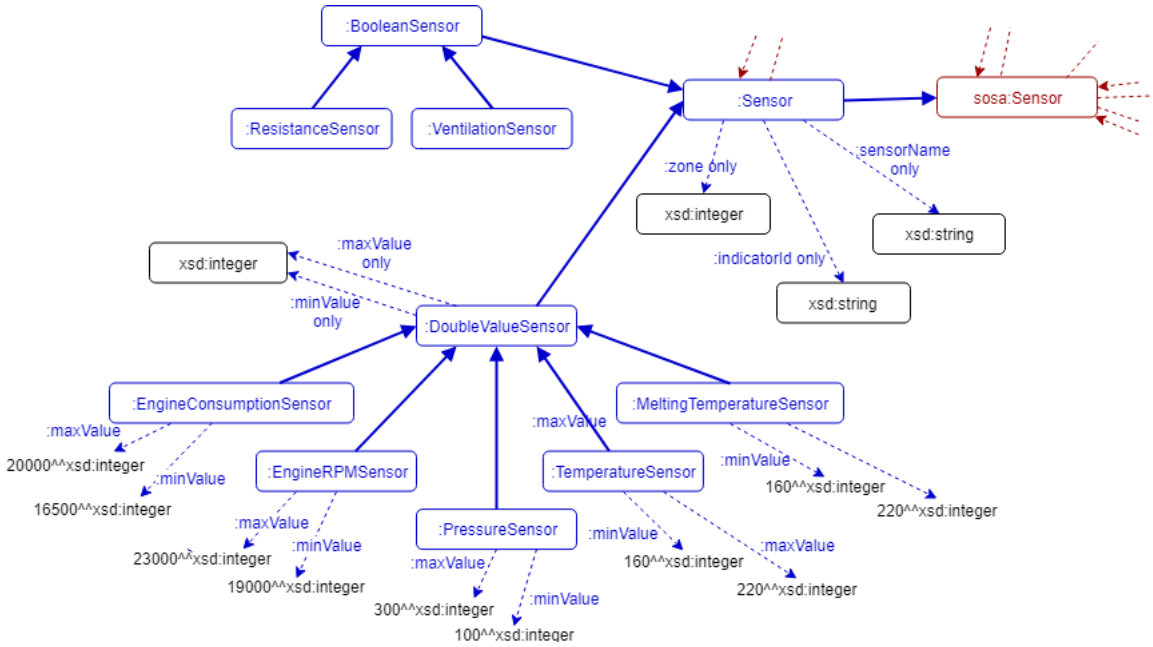

Figura 5.3: Especialización de la clase sosa:Sensor de la ontología SSN/SOSA

Por otro lado, en la SSN se define la clase **sosa:Observation**, la cuál modela las diferentes observaciones concretas realizadas por los sensores. Esta clase es dominio de dos propiedades de dato que se han utilizado para representar el resultado concreto de cada observación:

- *sosa:resultTime*. Hace referencia al momento en el que la observación ha sido realizada y su rango está definido como únicamente de tipo xsd:dateTime.

- *sosa:hasSimpleResult*. Hace referencia al valor concreto de la observación en cuestión y su rango está definido de tipo rdfs:Literal, el cuál incluye a cualquier tipo de valor *literal*[6].

Con estos datos y siguiendo el mismo esquema que anteriormente, se ha especializado la clase sosa:Observation (Fig. 5.4).

Para ello, primero se ha creado la clase **:Observation** como subclase de sosa:Observation, y después se ha seguido con la especialización de la misma añadiéndole dos subclases:

[6]Un *literal* en un grafo RDF, consiste en una forma léxica (cadena de caracteres Unicode) o en un tipo de dato IRI, de manera que la IRI identifique el tipo de dato determinado del valor literal.

- **:DoubleValueObservation**. Hace referencia a las observaciones cuyo valor es de tipo *double*. Por ello, se ha restringido en la misma el rango de la propiedad *sosa:hasSimpleResult* únicamente al tipo de dato xsd:double.

- **:BooleanObservation**. Hace referencia a las observaciones cuyo valor es de tipo *booleano*. Por ello, se ha restringido en la misma el rango de la propiedad *sosa:hasSimpleResult* únicamente al tipo de dato xsd:boolean.

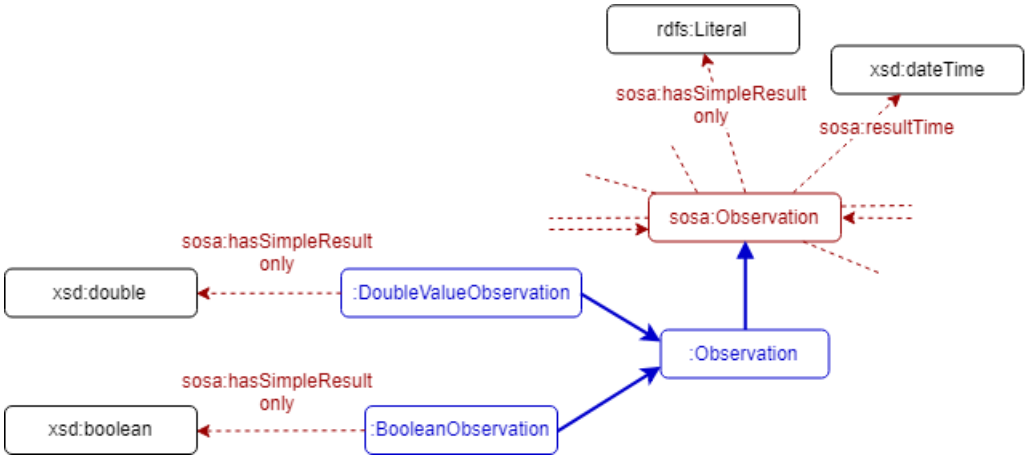

Figura 5.4: Especialización de la clase sosa:Observation de la ontología SSN/SOSA

En cuanto a las propiedades que los sensores observan, en la SSN éstas se definen mediante la clase **sosa:ObservableProperty**. Esta clase también ha sido especializada (Fig. 5.5), definiendo **:ObservableProperty** como subclase de la misma y añadiendo a esta última dos nuevas subclases:

- **:BooleanKind**. Esta clase define aquellas propiedades que tienen una cualidad *binaria*, es decir, las propiedades cuyo resultado podría ser especificado con un tipo de dato *booleano*.
  Para poder especificar el significado de cada uno de los dos estados posibles que pueden tener estar propiedades, se han creado dos nuevas propiedades de dato:

  - *:meaningState0*. Representa el significado concreto del valor *falso* en una propiedad booleana concreta.
  - *:meaningState1*. Representa el significado concreto del valor *verdadero* en una propiedad booleana concreta.

Centrándonos en el dominio a describir, a partir de esta clase se ha creado una instancia llamada ***:operationState***, que representa la propiedad ”estado operacional” de un elemento. Ésta indica si un componente está activado o desactivado, y también se han definido en la misma los valores de *:meaningState0* y *:meaningState1* acorde con su significado.

- **qu:QuantityKind**. Esta clase ha sido importada de un pequeño módulo (el cual se explica a continuación) creado para el desarrollo concreto de esta ontología y la cual representa las propiedades que son cuantificables. La clase en cuestión tiene definidas también cierto número de instancias que representan las propiedades cuantificables concretas que forman parte del dominio a describir: ***quantity:power***, ***quantity:pressure***, ***ququantity:temperature*** y ***quantity:rotationalSpeed***.

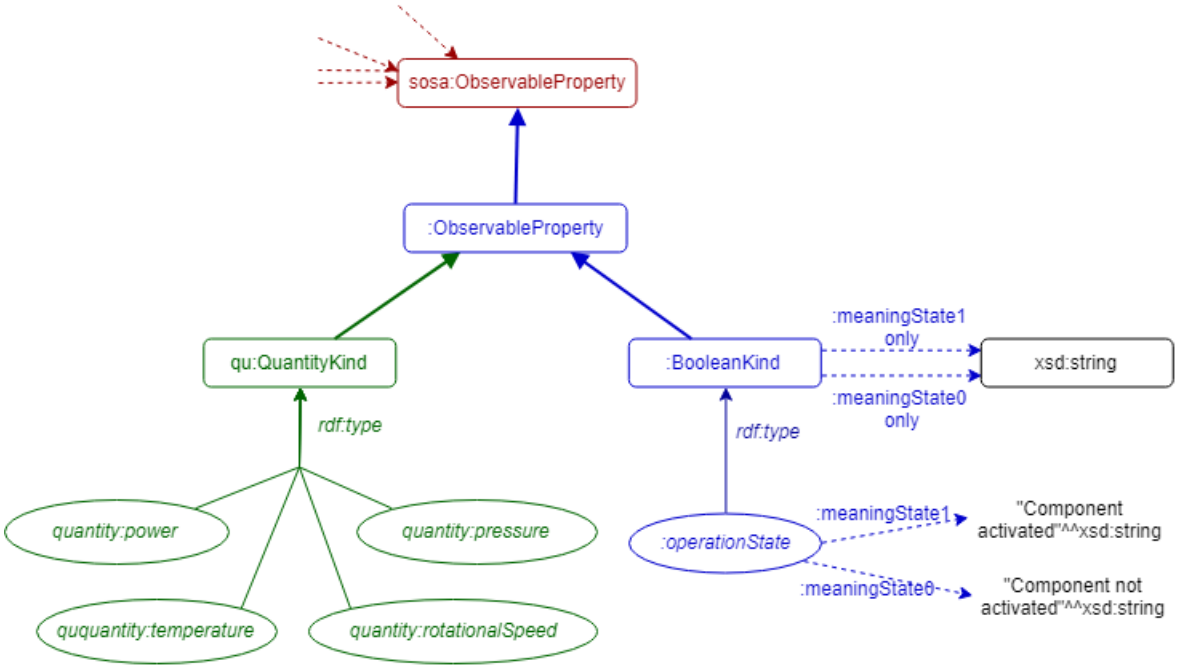

Figura 5.5: Especialización de la clase sosa:ObservableProperties de la ontología SSN/SOSA con la utilización del módulo *quantities*

Como bien se ha introducido en el punto anterior, para el desarrollo de esta ontología también se ha creado un pequeño módulo adicional que nos ayuda a definir las diferentes propiedades cuantificables utilizadas así como las unidades de medida de las mismas.

Este módulo, llamado *quantities*[7], es en definitiva un pequeño extracto de la ontología *Quantities, Units, Dimensions and Types (QUDT)*[8], de manera que contiene únicamente las propiedades y unidades relevantes para el desarrollo de este proyecto (Fig. 5.6).

Por ello, el módulo en cuestión está compuesto por dos clases principales y las instancias que derivan de ambas, las cuales están importadas a partir de dicha ontología:

- **qu:QuantityKind**. Esta clase agrupa las instancias de aquellas propiedades cuantificables que son relevantes en este dominio concreto.
- **qu:Unit**. Esta clase agrupa las instancias de aquellas unidades de medida que son relevantes en este dominio concreto, las cuales se corresponden con las propiedades cuantificables a observar.

Antes de continuar, se ha de puntualizar que las instancias que tienen que ver con la temperatura (***ququantity:temperature*** y ***quunit:degreeCelsius***) provienen de una versión algo diferente de la ontología QUDT, y por ello los prefijos de las mismas difieren del resto. El motivo de esta decisión se detallará en la sección 6.1 junto con el resto del desarrollo de la ontología.

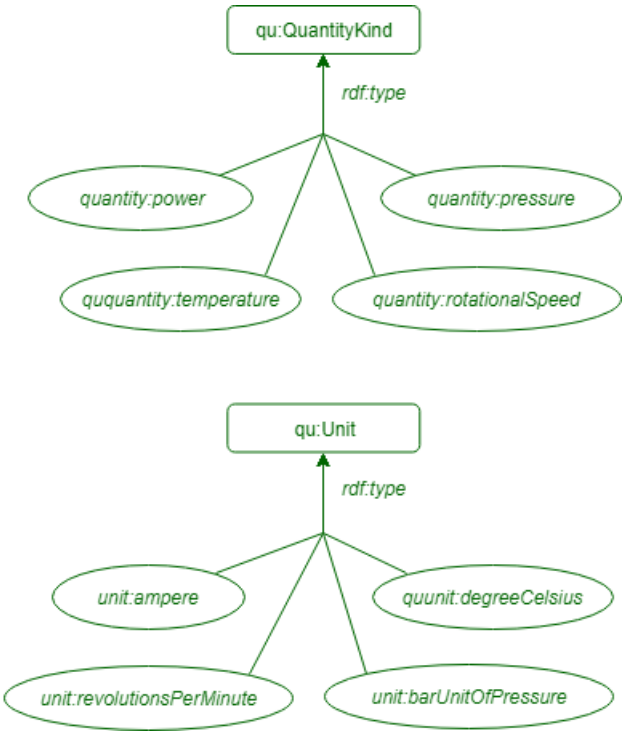

Figura 5.6: Componentes del módulo *quantities*.

[7]El módulo *quantities* desarrollado: http://bdi.si.ehu.es/bdi/ontologies/extrusion/quantities
[8]La ontología *QUDT*: http://data.qudt.org/qudt/owl/1.0.0/qudt.owl

Finalmente, una vez detalladas las especializaciones creadas a partir de las clases **sosa:Sensor**, **sosa:Observation** y **sosa:ObservableProperty** podemos pasar a detallar las relaciones y restricciones creadas a partir de dichas clases (Fig.: 5.7).

En la SSN existe una restricción que indica que todos los individuos que se relacionen con un individuo de la clase **sosa:Sensor** mediante la propiedad *sosa:madeObservation* deben pertenecer a la clase **sosa:Observation**. Teniendo en cuenta las especializaciones anteriormente explicadas, se ha restringido esta propiedad en las diferentes subclases de sensor creadas:

- En la clase **:Sensor** el rango de *sosa:madeObservation* se ha definido como de tipo **:Observation**, restringiendo la relación a nuestro dominio específico descrito.
- En la clase **:DoubleValueSensor** el rango de *sosa:madeObservation* se ha definido como de tipo **:DoubleValueObservation**, representando que los sensores de este tipo sólo podrán poseer observaciones de tipo *double*.
- En la clase **:BooleanSensor** el rango de *sosa:madeObservation* se ha definido como de tipo **:BooleanObservation**, representando que los sensores de este tipo sólo podrán poseer observaciones de tipo *booleano*.

En cuanto a las propiedades que los sensores observan, en la SSN existe una restricción que indica que todos los individuos que se relacionen con un individuo de la clase **sosa:Sensor** mediante la propiedad *sosa:observes* deben pertenecer a la clase **sosa:ObservableProperty**. Siguiendo la misma regla que antes, esta propiedad ha sido también restringida en las diferentes subclases de sosa:Sensor:

- En la clase **:Sensor** el rango de la propiedad *sosa:observes* se ha definido como **:ObservableProperty**, restringiendo la relación a nuestro dominio específico descrito.
- En la clase **:DoubleValueSensor** el rango de la propiedad *sosa:observes* se ha definido como **qu:QuantityKind**, representando que este tipo de sensores podrán observar únicamente propiedades cuantificables. Además, como las propiedades cuantificables deben tener una unidad de medida especificada, se ha definido la clase **:DoubleValueSensor** como dominio de la nueva propiedad de objeto *:unit* y cuyo rango es únicamente de tipo **qu:Unit**.

- En la clase **:BooleanSensor** el rango de la propiedad *sosa:observes* se ha definido únicamente como **:BooleanKind**, representando que este tipo de sensores sólo podrán observar propiedades de cualidad *binaria*.

Por último, se han definido las propiedades concretas que observa cada tipo específico de sensor (:PressureSensor, :TemperatureSensor, :ResistanceSensor, etc.), así como las unidades que se utilizan para cada propiedad en caso de que éstas sean cuantificables:

- En la clase **:ResistanceSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***:operationState***, el cual es una instancia de la clase :BooleanKind.
- En la clase **:VentilationSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***:operationState***, el cual es una instancia de la clase :BooleanKind.
- En la clase **:EngineConsumptionSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***quantity:power***, el cual es una instancia de la clase qu:QuantityKind. A su vez, la propiedad de objeto *:unit* tendrá como valor el objeto ***unit:ampere***, el cual es una instancia de la clase qu:Unit.
- En la clase **:EngineRPMSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***quantity:rotationalSpeed***, el cual es una instancia de la clase qu:QuantityKind. A su vez, la propiedad de objeto *:unit* tendrá como valor el objeto ***unit:revolutionsPerMinute***, el cual es una instancia de la clase qu:Unit.
- En la clase **:PressureSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***quantity:pressure***, el cual es una instancia de la clase qu:QuantityKind. A su vez, la propiedad de objeto *:unit* tendrá como valor el objeto ***unit:barUnitOfPressure***, el cual es una instancia de la clase qu:Unit.
- En la clase **:MeltingTemperatureSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***quantity:temperature***, el cual es una instancia de la clase qu:QuantityKind. A su vez, la propiedad de objeto *:unit* tendrá como valor el objeto ***unit:degreeCelsius***, el cual es una instancia de la clase qu:Unit.
- En la clase **:TemperatureSensor**, la propiedad de objeto *sosa:observes* tendrá como valor el objeto ***quantity:temperature***, el cual es una instancia de la clase qu:QuantityKind. A su vez, la propiedad de objeto *:unit* tendrá como valor el objeto ***unit:degreeCelsius***, el cual es una instancia de la clase qu:Unit.

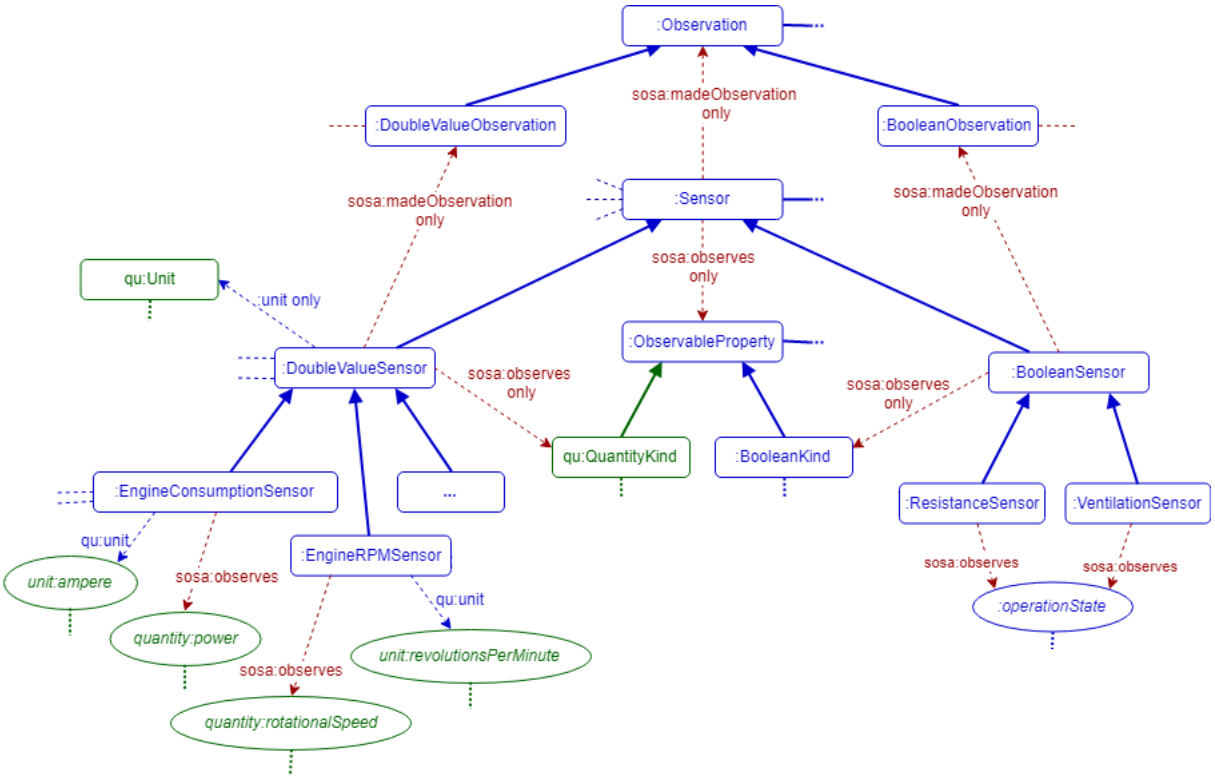

Figura 5.7: Parte de la unión entre las clases que representan los sensores, las observaciones y las propiedades observables

De esta manera se define la estructura general de la ontología desarrollada, definiendo los diferentes tipos de sensores que forman parte de nuestro dominio y especificando sus características.

Como último paso, a partir de cada tipo de sensor se definen las instancias concretas que forman parte de la máquina en cuestión, haciendo referencia concretamente a la *Extrusora de Cuatro Zonas* de Urola Solutions. Un pequeño ejemplo de dichas instancias se presenta en la figura 5.8, diagrama que muestra la definición de las instancias de los tipos de sensores **:TemperatureSensor** y **:PressureSensor**.

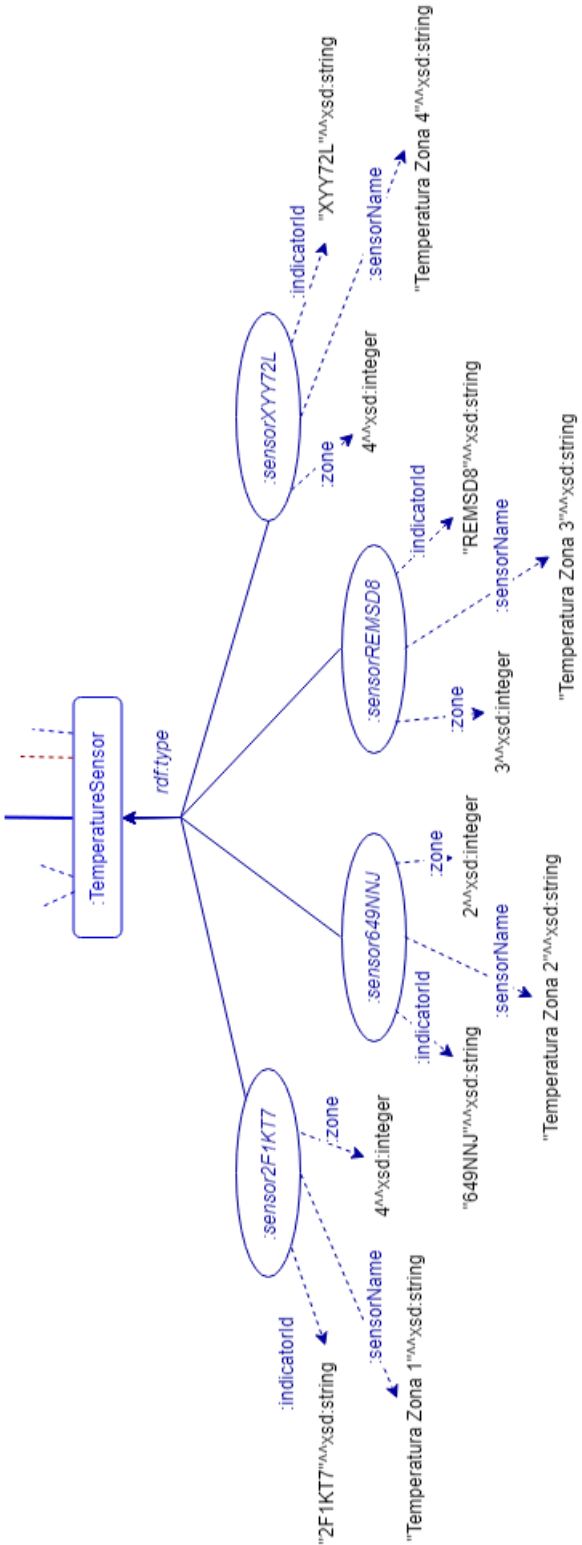

Figura 5.8: Ejemplo de instancias de diferentes sensores

5.2. Diseño de las interfaces

Antes del desarrollo de las interfaces que forman parte del sistema web, se ha llevado a cabo un proceso de diseño de las mismas, buscando ofrecer la mejor experiencia de usuario posible.

En esta sección se explican las decisiones de diseño más relevantes llevadas a cabo para el resultado final de la aplicación web resultante de este TFG.

5.2.1. Diseño del sistema visual de consultas

Una de las funcionalidades principales del sistema es la de consulta de datos, la cual se ha planteado con un enfoque visual, de manera que sea entendible para usuarios con un nivel básico de conocimientos informáticos.

Por ello, el diseño de las interfaces de dicha funcionalidad ha tenido un papel muy relevante en el desarrollo de este TFG, buscando satisfacer, en la mayor medida posible, los requisitos no funcionales impuestos por el enfoque visual comentado.

Este diseño se ha basado principalmente en una representación gráfica de la máquina extrusora a analizar y los sensores que capturan los datos de la misma (Fig. 5.9). Ésto ofrece una alternativa gráfica para la selección de los sensores, de manera que el usuario podrá seleccionarlos directamente sobre la imagen.

Dicha imagen y los sensores representados en ella varían dependiendo de la máquina a consultar seleccionada previamente, y para ello se utilizará la información recogida a partir de la ontología desarrollada. En el caso concreto de este TFG, por el momento solo se tiene información sobre la *Extrusora de Cuatro Zonas*, cuyo dominio ya se ha explicado anteriormente (sección 4.4).

Para mejorar la legibilidad visual de la extrusora sin recargar en exceso la imagen, se decidió asignar un icono lo suficientemente significativo a cada tipo de sensor, de manera que permitiera identificarlos por dichos iconos asociados. Además, para una mayor diferenciación entre los diferentes tipos de sensores, se ha asignado un color diferente a cada uno.

Sin embargo, puesto que ciertos iconos pueden llegar a ser algo confusos en una primera utilización del sistema, se decidió hacer uso de *tooltips* para proporcionar al usuario la información necesaria sobre dichos iconos. De esta manera, al posicionar el cursor sobre el icono deseado, aparece un texto explicativo sobre el mismo (Fig. 5.10).

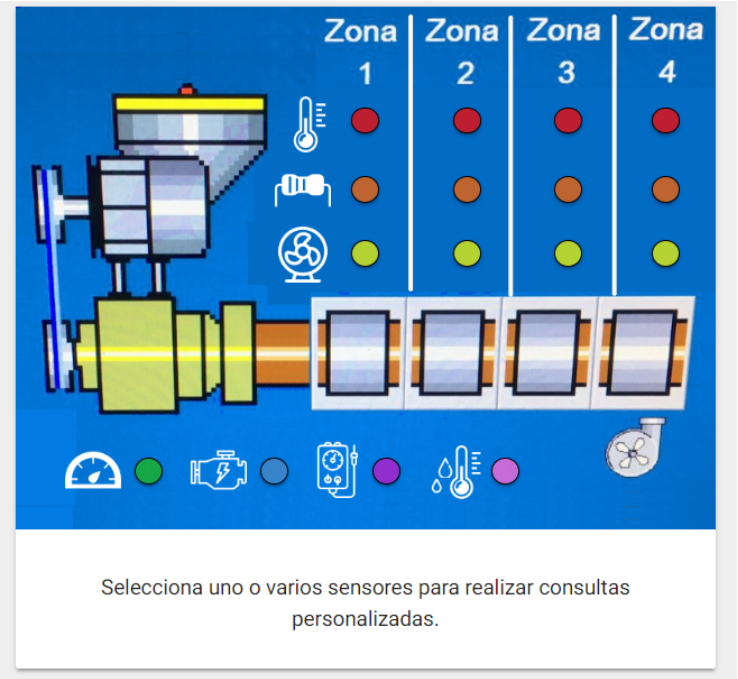

Figura 5.9: Representación gráfica de la *Extrusora de Cuatro Zonas* y los sensores que la componen

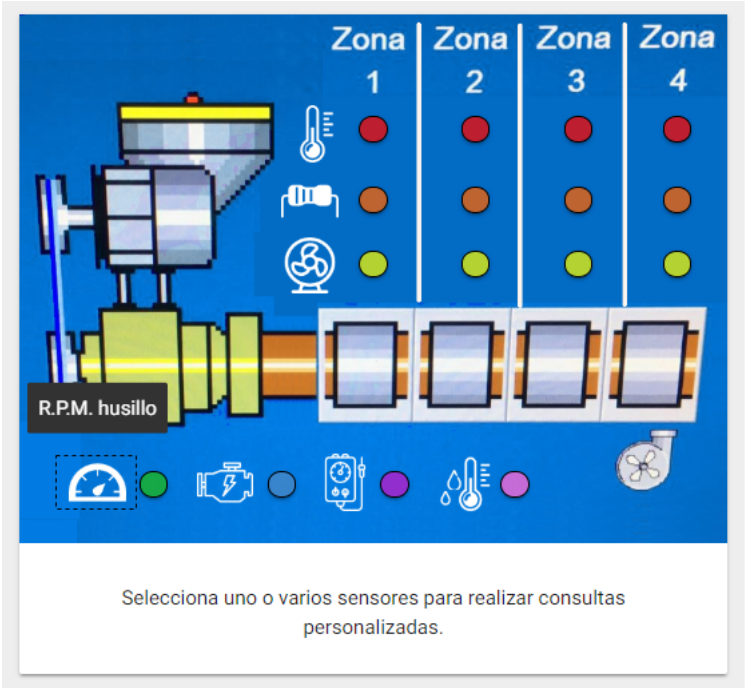

Figura 5.10: Ejemplo de información en forma de *tooltip* al posicionar el cursor sobre los iconos del mapa de la extrusora.

A la hora de seleccionar los sensores, se ha optado por representar dicha selección gráficamente sobre la imagen y, al mismo tiempo, añadir también una lista de los sensores seleccionados bajo la imagen de la extrusora (Fig. 5.11). En la imagen, el sensor seleccionado pasará a tener una forma cuadrada en vez de circular y su borde pasará a estar definido en color blanco para una mayor distinción. La lista de sensores seleccionados, por otro lado, ayuda a una identificación más rápida de los mismos después de haber pasado cierto tiempo desde la selección.

Figura 5.11: Representación de la selección de ciertos sensores sobre la imagen

Por otro lado, en función del número de sensores seleccionados, las consultas que el usuario pueda realizar también varían. Por esto, es importante tener un buen diseño de cómo serán presentados los distintos formularios de cada tipo de consulta, de manera que el usuario pueda identificar fácilmente qué consultas puede realizar en cada momento.

Con este fin, se ha diseñado un sistema de pestañas. En estas pestañas estarán definidos los distintos tipos de preguntas, de manera que dichas pestañas se habilitarán o deshabilitarán en función de los sensores seleccionados en cada momento. En la figura 5.12 se muestra el estado de dichas pestañas con un solo sensor seleccionado, de manera que las consultas de relación entre sensores están deshabilitadas. En la figura 5.13 sin embargo, se muestra el estado de las pestañas con dos sensores seleccionados, de manera que no

hay ninguna pestaña deshabilitada y, además, aparece una lista desplegable en las consultas de búsqueda de anomalías. Esto último se debe a que las consultas de anomalías predefinidas pueden ser realizadas sin importar el número de sensores seleccionados, pero las personalizadas sin embargo dependen de que se hayan seleccionado como mínimo dos sensores.

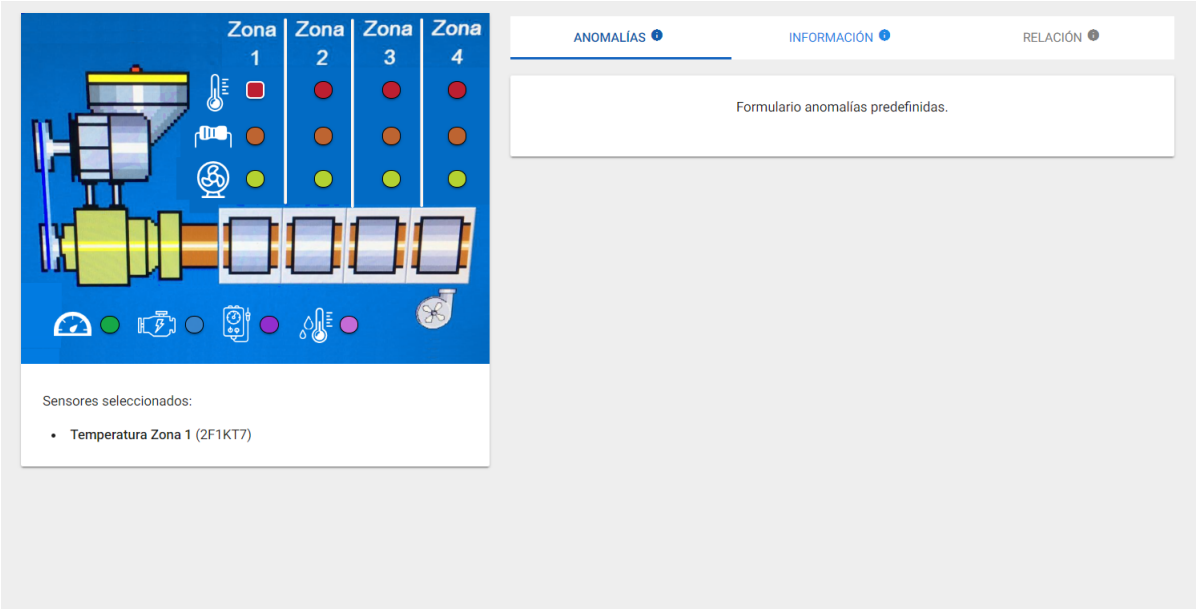

Figura 5.12: Estado de las pestañas de consultas con un sólo sensor seleccionado

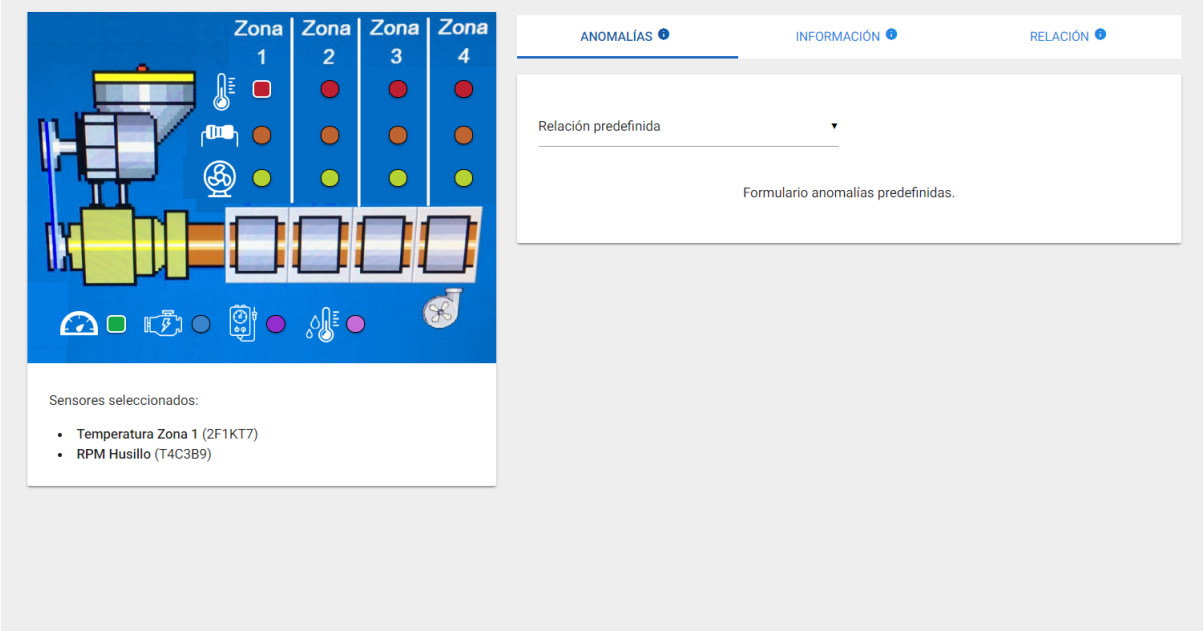

Figura 5.13: Estado de las pestañas de consultas con dos sensores seleccionados

Para la formulación de las consultas, se ha optado por un diseño en formato de formulario, representando en el mismo todas las posibilidades de personalización que se ofrecen al usuario, buscando siempre la manera más visual e intuitiva para ello.

Un ejemplo de esto último es la representación utilizada en los formularios para la realización de los filtros de valores. En el caso de sensores con valores de tipo numéricos, dicho filtro se ha representado mediante un "deslizador"ó *slider* de rangos, el cual permite al usuario especificar el rango de valores a consultar. En el caso de los sensores con valores de tipo *booleanos*, dicho filtro se ha representado mediante un *interruptor*, el cual permite al usuario seleccionar el estado del sensor a tener en cuenta en la consulta (activado o desactivado) (Fig. 5.14).

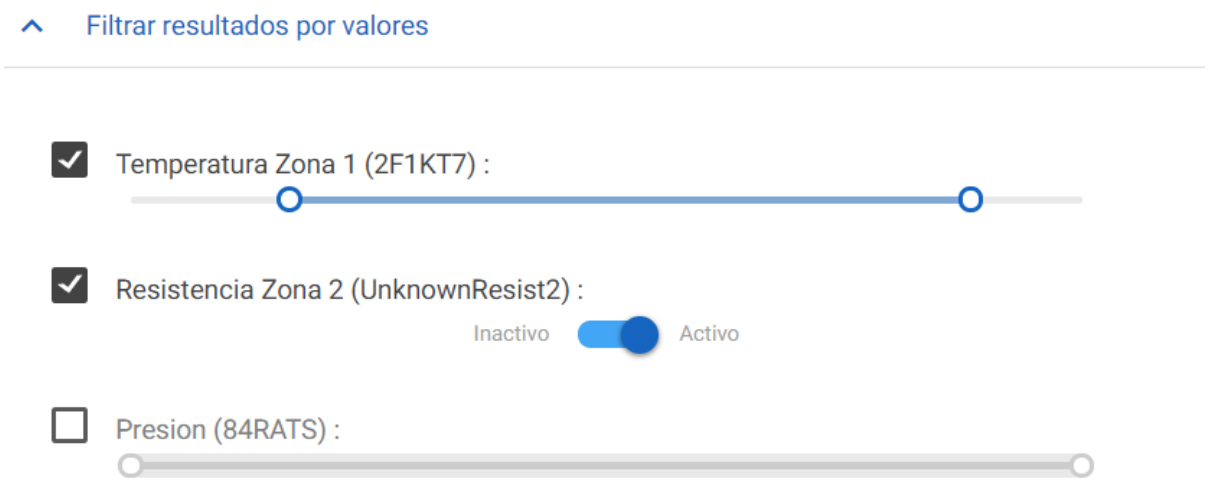

Figura 5.14: Ejemplo del filtro de valores de diferentes sensores

Por último, dada la gran cantidad de información que puede ser introducida y personalizada en los formularios, es necesario añadir cierta validación en los mismos, de manera que el usuario pueda identificar los posibles errores surgidos al rellenarlos. Además, se ha decidido que no se permitirá la realización de ninguna consulta si la información de los formularios es errónea.

En las figuras 5.15 y 5.16 se muestran distintos ejemplos de dichas validaciones en los formularios.

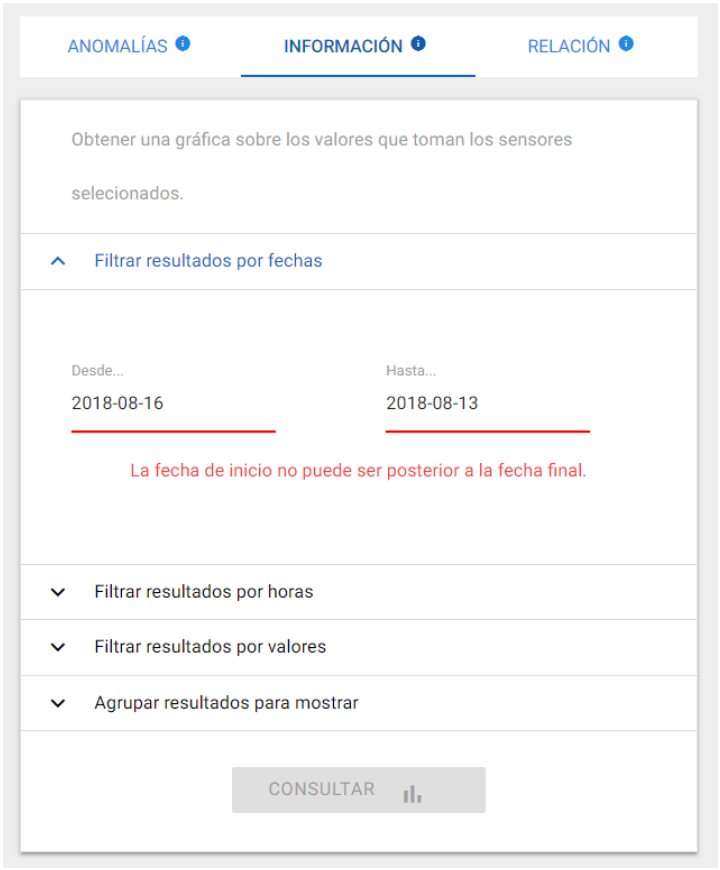

Figura 5.15: Ejemplo de validación en el cliente en la definición de filtros de fechas.

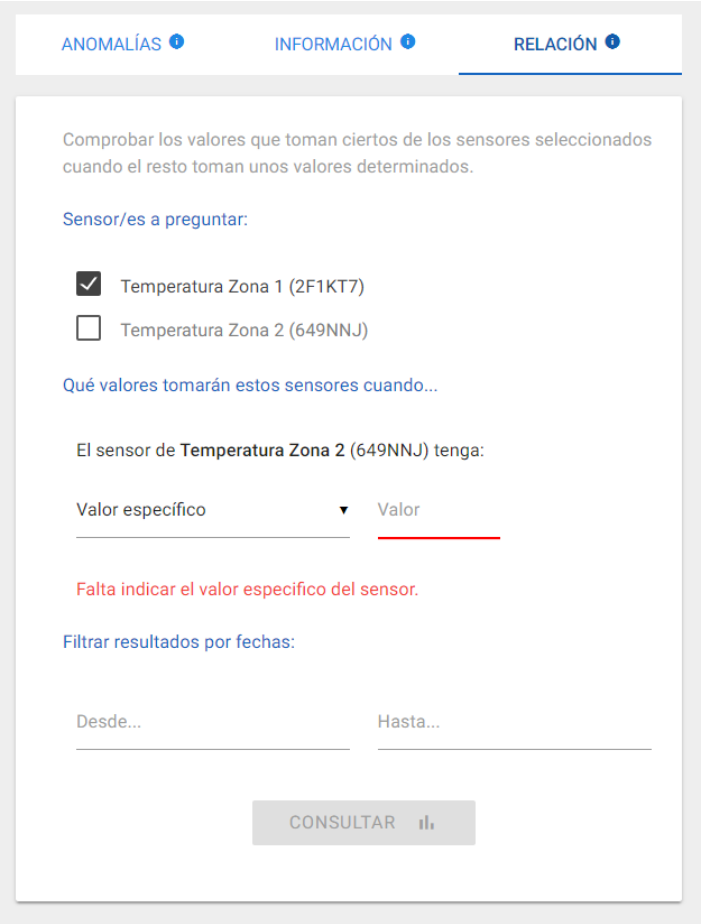

Figura 5.16: Ejemplo de validación en el cliente por falta de información a introducir.

5.2.2. Diseño de la visualización de los resultados

Una vez realizada la consulta, los resultados son mostrados gráficamente al usuario. En esta sección se explican las decisiones de diseño tomadas a cabo para la visualización de los resultados de las consultas, detallando las variaciones en las características de las gráficas dependiendo de las consultas realizadas.

Para empezar, para mantener al usuario informado en todo momento sobre a qué consulta pertenecen los resultados visualizados, se mostrará un pequeño resumen de la consulta realizada junto con la gráfica (Fig. 5.17).

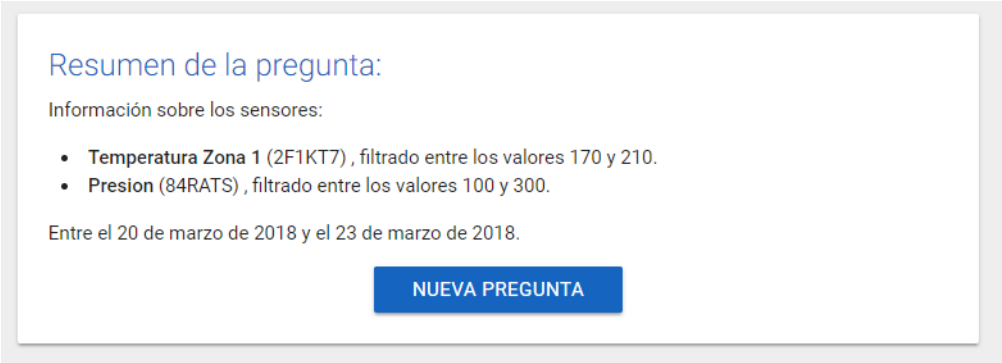

Figura 5.17: Ejemplo de la información mostrada al realizar una consulta, conteniendo el resumen de la misma.

En cuanto a las características concretas de las gráficas, se han definido tres tipos concretos de gráficas que podrán ser fruto de diferentes consultas:

- **Gráficas lineales.** Este tipo de gráficas se utilizarán cuando los resultados a mostrar sean continuos, ya que para su visualización se traza una línea que une los diferentes datos que se muestran en el gráfico. (Fig. 5.18)
- **Gráficas de puntos dispersos.** Este tipo de gráficas se utilizarán cuando los resultados a mostrar no sean continuos, ya que su visualización es un conjunto de diferentes puntos individuales que representan cada valor del gráfico. (Fig. 5.19)
- **Gráficas de barras.** Este tipo de gráficas se utilizarán para mostrar valores resultantes de funciones agregadas. (Fig. 5.20)

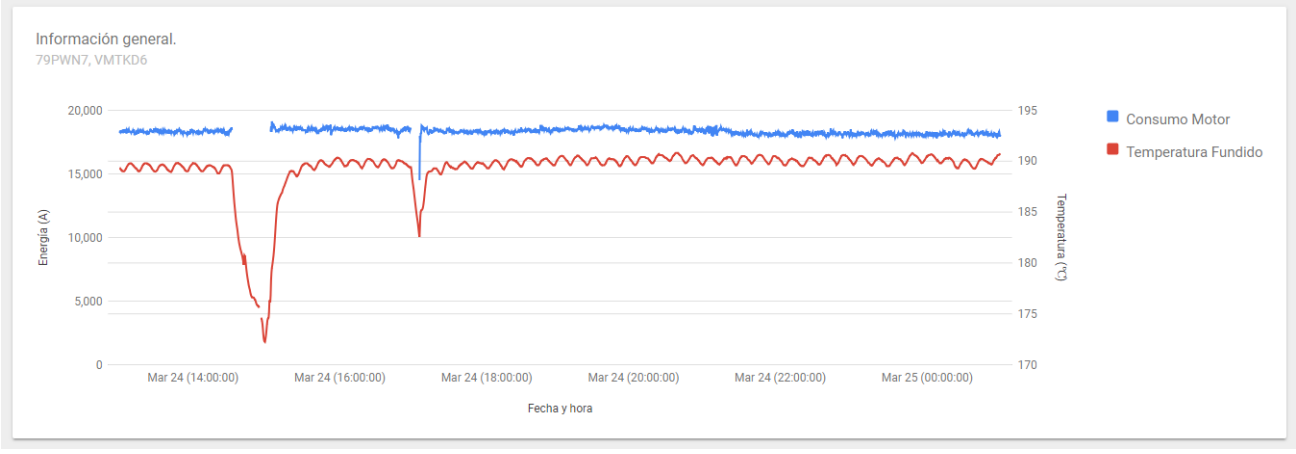

Figura 5.18: Ejemplo de una gráfica lineal.

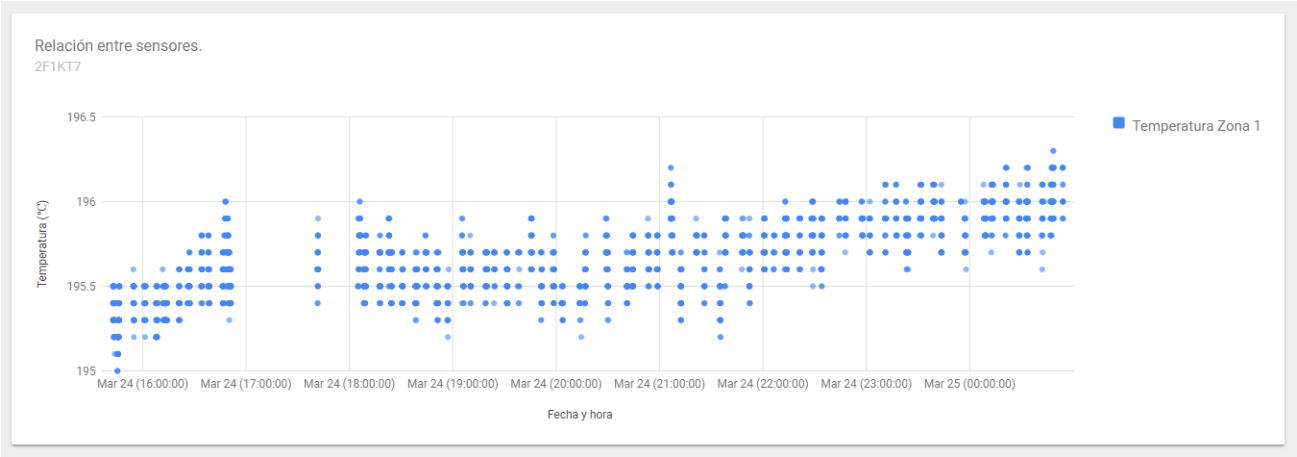

Figura 5.19: Ejemplo de una gráfica de puntos dispersos.

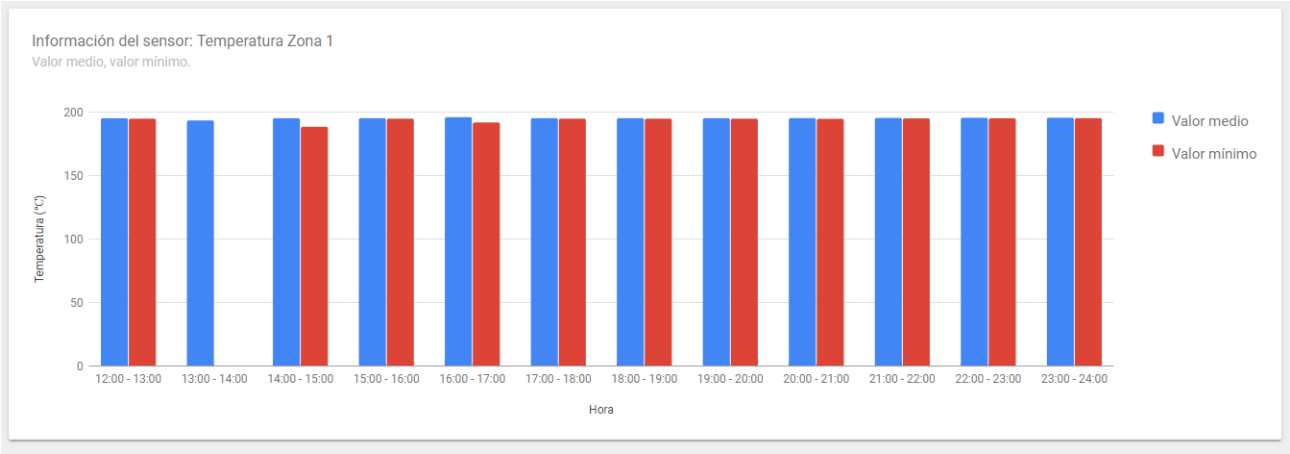

Figura 5.20: Ejemplo de una gráfica de barras.

Por otro lado, los sensores observan en su mayoría propiedades muy diferentes, teniendo distintas unidades de medida. Esto implica que, para poder representar los resultados de más de un sensor de tipos distintos en un mismo gráfico, dicho gráfico deberá ser capaz de mostrar debidamente las diferentes unidades de medida asociadas a cada uno.

Para ello, por cada tipo de sensor cuyos resultados se vayan a mostrar, se añadirá un nuevo eje *Y* en la gráfica, indicando en éste la unidad de medida asociada al mismo (Fig. 5.21).

También puede haber casos en los que se deba mostrar más de un valor por cada momento de tiempo de un mismo sensor. Esto puede ocurrir al querer mostrar más de un valor agrupado de cada sensor (el valor mínimo y el máximo de un sensor cada día, por ejemplo). En estos casos, se ha decidido que los resultados de cada sensor se mostrarán

en un gráfico diferente (Fig. 5.22), ya que de lo contrario el gráfico no quedaría intuitivo ni comprensible.

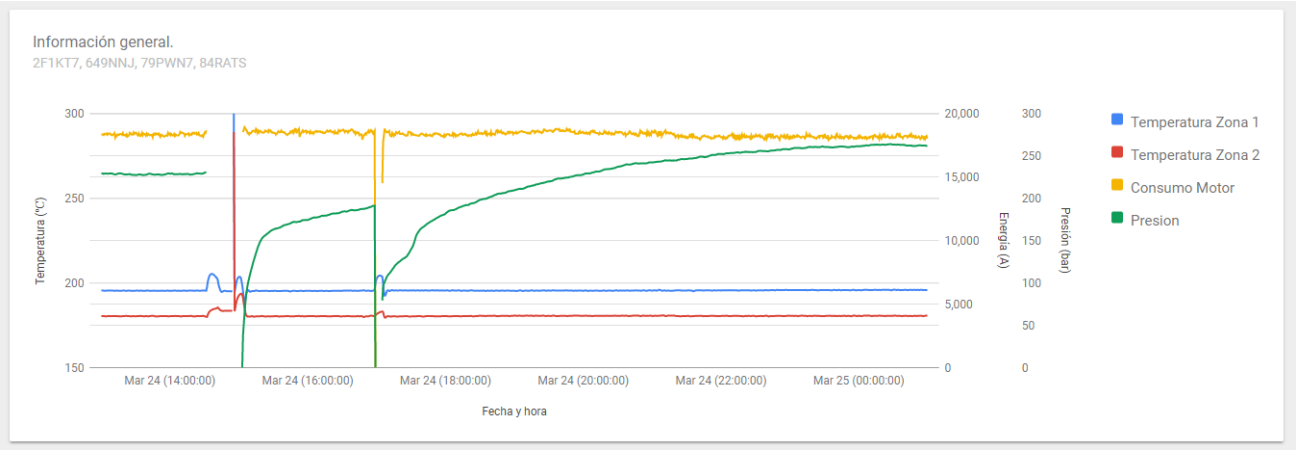

Figura 5.21: Ejemplo de una gráfica con tres escalas distintas.

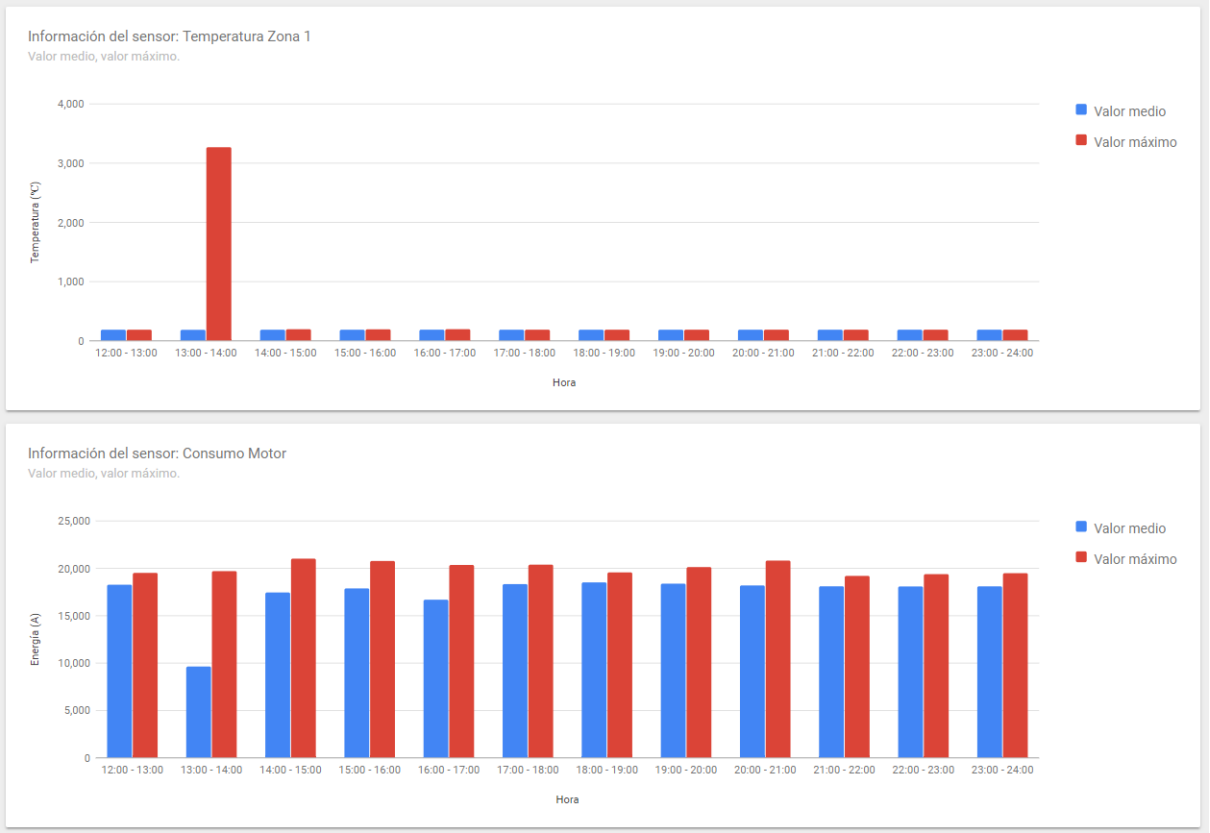

Figura 5.22: Gráficas resultantes con más de un valor cada unidad de tiempo por cada sensor.

Por último, en las gráficas de información general, también se muestran los *outliers* de cada sensor. En la ontología desarrollada están detallados los valores máximos y mínimos que debería tener cada sensor en situaciones habituales, por lo que se ha decidido utilizar dicha información para (Fig. 5.23). Esto sin embargo, se hará únicamente cuando se muestren datos de un mismo tipo de sensor, ya que de lo contrario se tendrían dos outliers por cada tipo de sensor (un outlier superior y otro inferior), recargando el gráfico con un exceso de información y afectando a la legibilidad del mismo.

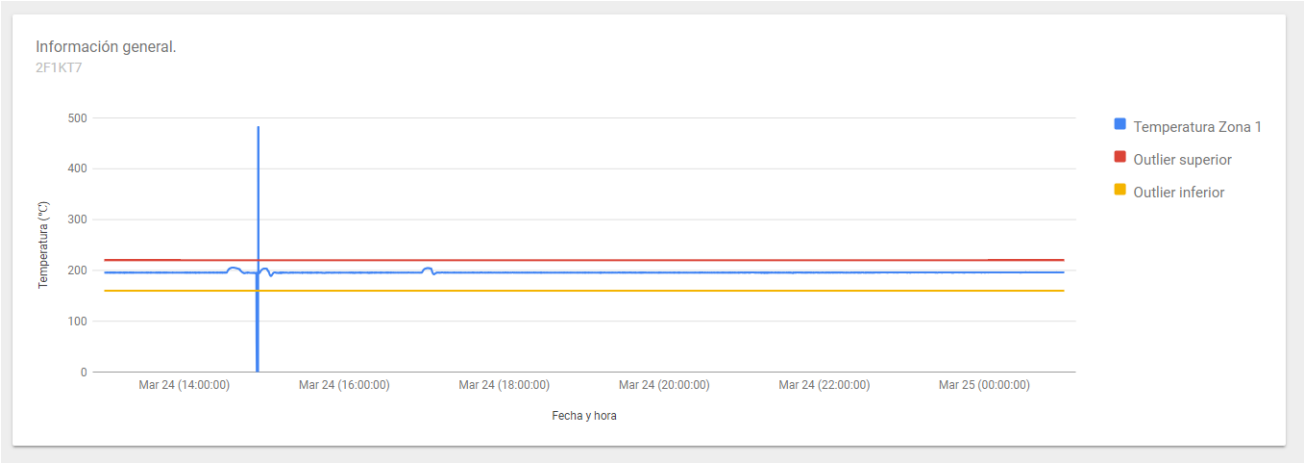

Figura 5.23: Ejemplo de una gráfica en la que se muestran los *outliers* del sensor consultado.

5.2.3. Diseño de la inserción de datos

Otra de las funcionalidades del sistema es la de inserción de datos, en la que el usuario puede insertar los datos deseados de los diferentes sensores contemplados en una determinada máquina extrusora.

Para esta funcionalidad también se ha decidido el uso de un formulario que permita al usuario subir el fichero CSV con los datos a insertar. Dicho fichero sin embargo, debe tener como nombre el identificador del sensor al que corresponden los datos. Si dicho identificador no corresponde con ninguno de los sensores contemplados en la ontología, el sistema no permitirá realizar la inserción. (Fig. 5.24)

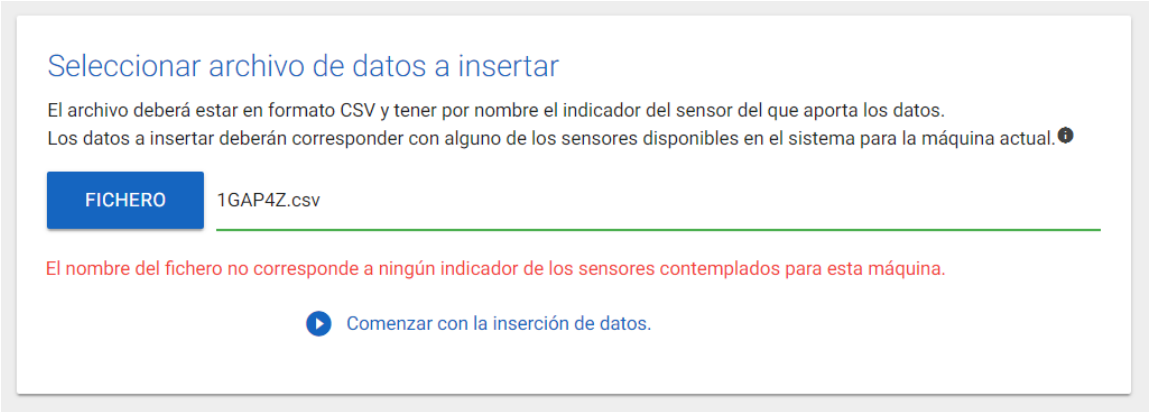

Figura 5.24: Ejemplo de validación en el cliente por información erronea introducida.

Para facilitar la identificación de dichos sensores contemplados en cada máquina, se ha decidido añadir una leyenda especificando los identificadores y nombres significativos de los mismos. Esta leyenda podrá ser activada y desactivada por el usuario cuando éste lo desee mediante un icono de información en el formulario de inserción (Fig. 5.25).

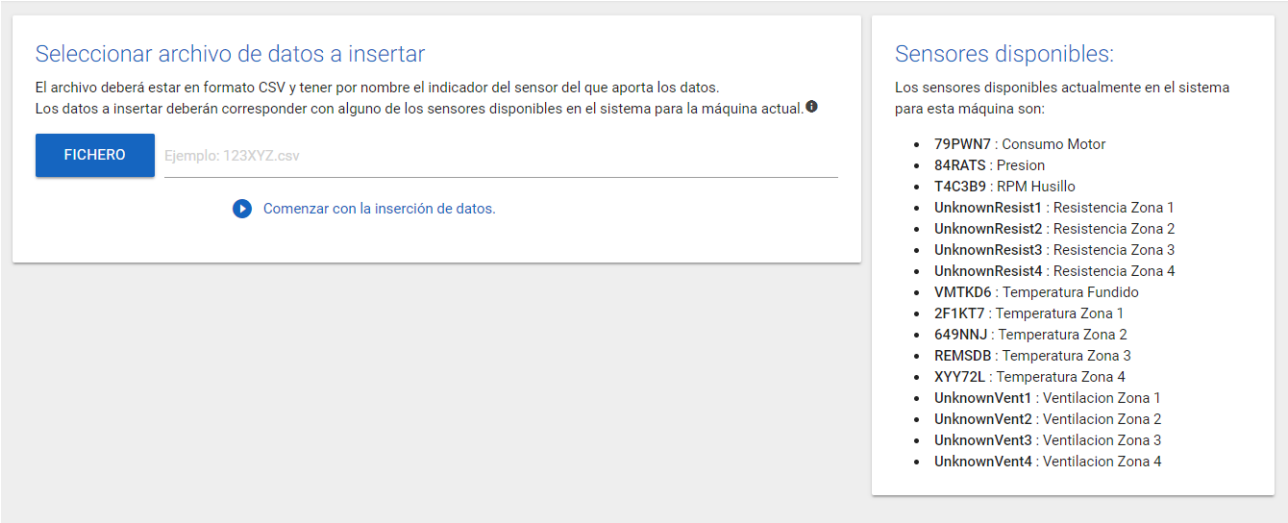

Figura 5.25: Información mostrada al *clicar* sobre el icono de información aparecido en el formulario de inserción de datos.

Por último, en el proceso de inserción de datos, la interfaz informará en todo momento del proceso en el que se encuentra el sistema, para así mantener debidamente informado al usuario.

5.3. Arquitectura

Dado que el sistema web desarrollado en este TFG va a ser incorporado en un futuro al sistema en desarrollo por el grupo BDI, la arquitectura del mismo se ha diseñado pensando en la unión de ambos sistemas.

En la figura 5.26 se presenta un diagrama de dicha arquitectura, en la que se pueden diferenciar tres colores distintos:

- **Magenta:** Representa las partes del sistema web que han sido desarrolladas e implementadas por el grupo de investigación BDI.
- **Negro:** Representa las partes del sistema correspondientes a lo desarrollado en este TFG.
- **Azul:** Representa las partes del sistema que han sido inicialmente desarrolladas por el grupo BDI pero a las que se les han añadido ciertas partes correspondientes a este TFG.

A continuación se detallan los diferentes módulos que aparecen en dicha arquitectura, definiendo el objetivo y función de cada uno.

I4TSPS Web App

Representa la aplicación web desarrollada por el grupo BDI.

El módulo *User Interface* hace referencia a la interfaz inicial de la aplicación, donde se le presentan al usuario los distintos módulos de los que se dispone.

En este TFG en concreto, se ha desarrollado el módulo que llamaremos *Semantic Module*, pero la aplicación I4TSPS tiene otros módulos ya desarrollados a los que los usuarios tendrán acceso, los que han sido representados por el conjunto de módulos *Others*. A continuación se explica con más detalle el módulo de *Semantic Module* mencionado.

Por otro lado, *Backend Object* hace referencia al módulo encargado de llevar a cabo la unión entre la aplicación web y el *Backend Service*, el cual ofrece a la aplicación la información necesaria para la autenticación de los usuarios y hace la función de base de datos general de la aplicación I4TSPS.

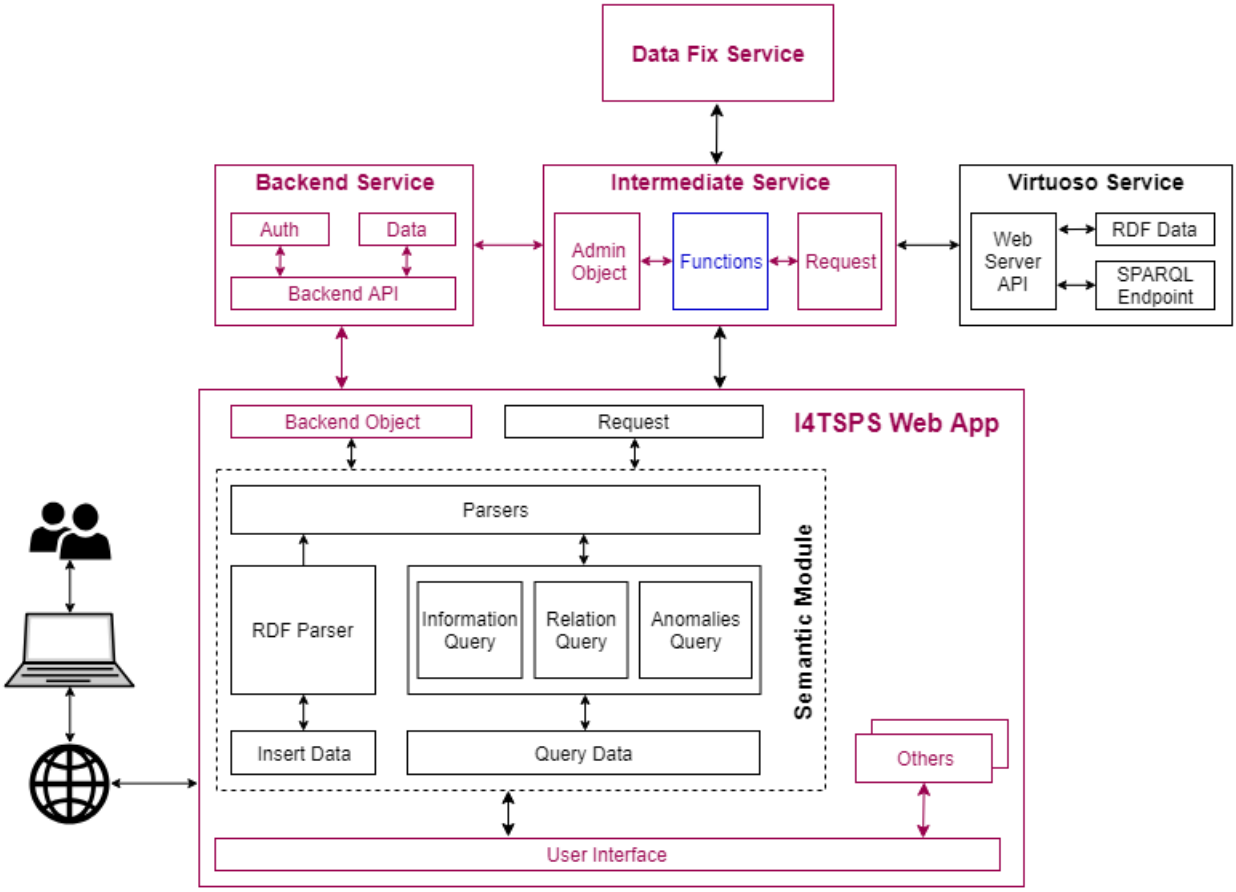

Figura 5.26: Diagrama de la arquitectura del sistema web

Finalmente, el módulo *Request* se encarga de realizar las peticiones HTTP al servicio *Intermediate Service*, el cual sirve de intermediario entre la aplicación y los diferentes servicios adicionales que utiliza.

Semantic Module

Los módulos *Insert Data* y *Query Data* hacen referencia a las interfaces que corresponden a las dos funcionalidades desarrolladas en este TFG.

Insert Data representa la interfaz que permite al usuario subir diferentes ficheros con los datos a insertar. Esta interfaz hace uso del módulo *RDF Parser* para llevar a cabo la funcionalidad asociada al mismo, el cual anota los datos recibidos de la interfaz en modelo RDF y los deja preparados para su inserción en el repositorio de datos.

Query Data representa la interfaz que permite al usuario realizar distintos tipos de consultas especificando las características de las mismas a través de formularios. La interfaz después muestra al usuario las gráficas correspondientes al resultado de dichas consultas. Esta interfaz hace uso de un conjunto de tres módulos simultáneamente, correspondiéndose cada uno con los tipos de consultas específicos ofrecidos en dicha funcionalidad:

- *Information Query*: Módulo encargado de realizar la lógica de las consultas de información general. Recibe de la interfaz los datos introducidos por el usuario en el formulario y devuelve los datos a mostrar en el gráfico.

- *Relation Query*: Módulo encargado de realizar la lógica de las consultas de relación entre los valores de los sensores. Recibe de la interfaz los datos introducidos por el usuario en el formulario y devuelve los datos a mostrar en el gráfico.

- *Anomalias Query*: Módulo encargado de realizar la lógica de las consultas de anomalías en los valores de los sensores. Recibe de la interfaz los datos introducidos por el usuario en el formulario y devuelve los datos a mostrar en el gráfico.

Por último, el módulo *Parser* es el encargado de intermediar la comunicación entre el módulo semántico y el respositorio de datos del mismo, el servicio Virtuoso en este caso. Este módulo genera las sentencias SPARQL necesarias para poder realizar las peticiones HTTP al servicio de Virtuoso. De la misma manera, cuando se reciben datos desde Virtuoso, dicho módulo modifica la estructura de los mismos antes de pasárselos a los módulos encargados de la funcionalidad de consulta de datos.

Backend Service

Hace referencia al servicio que proporciona la información de autenticación y base de datos a la aplicación I4TSPS.

Para la conexión con el servicio, se utiliza el módulo *Backend Object* de la aplicación y se hace referencia al *Backend API* del servicio, donde se ofrecen las diferentes funcionalidades del servicio.

En el caso concreto de este TFG, el módulo *Auth* proporciona el servicio de autenticación necesaria para los usuarios de la aplicación, y el módulo *Data* recoge la información asociada a estos usuarios, como la organización a la que pertenece, las máquinas extrusoras aparecidas en dicha organización, etc.

Intermediate Service

Este servicio hace de intermediario entre la aplicación y los distintos servicios (a excepción del servicio de *Backend*) que utiliza la misma, para así realizar una serie de funciones asociadas a las peticiones de manera que no se realicen en el cliente y para solventar también ciertos problemas de seguridad en algunas de las peticiones realizadas a URLs no seguras.

En el módulo *Functions* están definidas dichas funciones asociadas a determinadas peticiones, y una vez realizadas las tareas definidas en dichas funciones, se realiza la petición HTTP a los servicios destino mediante el módulo *Request*.

Algunas de estas funciones necesitan a su vez de una conexión con el *Backend Service*, la cual se consigue a través del módulo *Admin Object*.

Virtuoso Service

Es el servicio encargado de contener el repositorio de datos en RDF necesario para el módulo semántico desarrollado en este proyecto.

La conexión a éste se hace a través de peticiones HTTP al servidor web que ofrece, el cual posee de una API con sus funcionalidades ofrecidas. Esto está representado en el módulo *Web Server API*.

El servicio de Virtuoso tiene una infinidad de funcionalidades disponibles, pero en este TFG sólo se utilizarán el módulo de almacenamiento de grafos RDF (*RDF Data*) y el módulo de realización de consultas SPARQL (*SPARQL Endpoint*).

Data Fix Service

Este servicio es utilizado para la funcionalidad de inserción de datos del módulo semántico, pero sus módulos internos no han sido desarrollados en el diagrama de la arquitectura ya que no entra en el alcance del proyecto.

Éste recibe como entrada una serie de datos devolviendo la misma serie pero con correcciones en los timestamps de los mismos (añadiendo los que faltan).

Este servicio se utiliza mediante peticiones HTTP desde la aplicación I4TSPS, concretamente desde el módulo *RDF Parser*.

CAPÍTULO 6

Desarrollo del proyecto

En este capítulo se aborda el desarrollo del sistema web a partir de todo lo comentado anteriormente. Para ello, se detalla un proceso clave del desarrollo en cada sección del mismo. Estas secciones no están necesariamente ordenadas cronológicamente en base a la realización de cada parte, si no que se ha seguido un orden lógico que facilite la comprensión del documento.

6.1. Describiendo el dominio: desarrollo de la Ontología

La ontología explicada en el modelo de representación de datos (sección 5.1) representa la base fundamental del sistema web desarrollado, ya que modela el dominio de la aplicación, aportando la información necesaria sobre el mismo al sistema y permitiendo el almacenamiento de los datos que después serán consultados y analizados.

Para el desarrollo de dicha ontología se ha seguido la metodología *NeOn* [Suárez-Figueroa et al., 2012a], una metodología diseñada para el proceso de creación de ontologías compuesta por un *set* de 9 escenarios. Cada escenario propone una serie de actividades o procesos que pueden ser combinados para la obtención del resultado esperado.

Concretamente, para el desarrollo de la ontología presentada en este TFG, se han utilizado y combinado cuatro de los escenarios propuestos en la metodología: escenario 1, escenario 2, escenario 3 y escenario 4.

A continuación se detallan dichos escenarios, explicando su finalidad y las actividades y procesos realizados en los mismos.

Escenario 1: de la especificación a la implementación

Este escenario es el eje vertebral del proceso de desarrollo de la ontología, que abarca desde la especificación de requisitos hasta la implementación de la ontología.

Dicho escenario está compuesto por cinco actividades base que deberían ser realizadas en el desarrollo de toda ontología:

- **Especificación de los requisitos.** Esta actividad se centra en definir los requisitos que la ontología debería cumplir, los cuales se exponen en el *Ontology Requirements Specification Document (ORSD)* (Tabla 6.1). Una parte importante de este documento son las llamadas "preguntas de competencia", que consisten en la especificación de las diferentes preguntas que la ontología final debería ser capaz de responder.

- **Búsqueda de fuentes de conocimiento.** Siguiendo las especificaciones definidas en el *ORSD*, esta actividad pretende reflejar la búsqueda del conocimiento necesario para poder cumplir dichos requisitos. Esta actividad ha sido realizada siguiendo las actividades presentadas en los escenarios 2, 3 y 4 que son explicados a continuación. De esta manera, se ha recogido el conocimiento ontológico y no ontológico necesario para el desarrollo de la ontología específica de este TFG.

- **Planificación.** En esta actividad se pretende describir el ciclo de vida que tendrá el desarrollo de la ontología. En este caso se ha seleccionado el ciclo de vida de seis fases en cascada definido en [Suárez-Figueroa et al., 2012b] debido a los escenarios específicos que serán llevados a cabo. Este ciclo de vida incluye los procesos presentes en el ciclo de vida de cuatro fases (inicialización, diseño, implementación y mantenimiento), el proceso extra presente en el ciclo de vida de cinco fases (reutilización) y por último se añade un proceso de rediseño.

- **Conceptualización y formalización.** Esta actividad representa el proceso de organización del conocimiento para el desarrollo de un modelo conceptual, que después de la formalización pasará a ser un modelo semi-computable. En este caso, a partir de las preguntas de competencia desarrolladas en el *ORSD* se desarrolló el modelo presentado en la sección 5.1.

- **Implementación.** Esta actividad representa el proceso de implementación de la ontología. Para ello se ha utilizado Protégé 5.2.0[1] como herramienta y se ha implementado en el lenguaje OWL 2[2].

Tabla 6.1: Resultado del *Ontology Requirements Specification Document (ORSD)*

1.	**Propósito**
	El propósito de esta ontología es representar y modelar con precisión los sensores implantados en la máquina extrusora *Extrusora de Cuatro Zonas* de Urola Solutions así como los resultados de las observaciones que éstos realicen.
2.	**Alcance**
	La ontología se centrará en los sensores presentes en una máquina extrusora concreta, la *Extrusora de Cuatro Zonas* desarrollada por la empresa *Urola Solutions*.
3.	**Lenguaje de implementación**
	La ontología deberá ser implementada en un lenguaje formal que permita determinar relaciones de jerarquía entre clases y clasificar instancias en clases.
4.	**Requisitos de la ontología**
	• *Requisitos no funcionales*: no es aplicable. • *Requisitos funcionales*: se representan a través de las preguntas de competencia. - GPC1: Preguntas relacionadas con los sensores. - PC1.1: ¿Qué propiedad observa el sensor? - PC1.2: ¿Qué tipo de resultado devuelve el sensor? - PC1.3: ¿Cuál es la unidad de medida utilizada en el sensor? - PC1.4: ¿En qué zona está situado el sensor? - PC1.5: ¿Cuál es el nombre del sensor? - PC1.6: ¿Cuál es el identificador del sensor? - PC1.7: ¿Cuál es el valor mínimo esperado como resultado del sensor? - PC1.8: ¿Cuál es el valor máximo esperado como resultado del sensor? - GPC2: Preguntas relacionadas con las observaciones de los sensores. - PC2.1: ¿En qué momento ha sido realizada la observación? - PC2.2: ¿Qué valor ha sido el resultado de la observación?
5.	**Usuarios destinados**
	• *Usuario 1*: Gestor de datos de las máquinas extrusoras. • *Usuario 2*: Consultor de datos de las máquinas extrusoras.

[1]Protégé: https://protege.stanford.edu/
[2]Especificación de OWL 2: https://www.w3.org/TR/owl2-overview/

Continuación de la tabla 6.1

6.	**Usos destinados**
	• *Uso 1*: Recoger información sobre los diferentes de tipos de sensores que componen la máquina y sus características. • *Uso 2*: Servir como base para la anotación de información concreta de cada sensor y de las observaciones efectuadas por los mismos.
7.	**Pre-glosario de términos**
	Sensor, observación, resultado, propiedad observada...

Escenario 2: reutilizando y rediseñando fuentes no ontológicas

En este escenario se refleja la búsqueda de fuentes de conocimiento que no estén expresadas en forma de ontología. Para ello se definen dos principales procesos en la misma: la reutilización de dichas fuentes y el rediseño de las mismas.

En este caso, se han buscado diferentes fuentes que detallen las características concretas de las extrusoras así como de los sensores implantados en la máquina extrusora a analizar. Para llevar esto a cabo, se han realizado las actividades relacionadas con cada proceso del escenario.

Actividades necesarias para la reutilización de las fuentes no ontológicas:

- **Búsqueda de fuentes no ontológicas.** A pesar de que se ha tenido que adquirir un nivel de comprensión básico sobre el proceso de la extrusión de polímeros en general, por el objetivo concreto de la ontología a desarrollar, la fuente de información no ontológica principal debe describir específicamente los sensores de la máquina extrusora a modelar. Por otro lado, el grupo de investigación BDI ha desarrollado un documento explicativo con las características específicas de las distintas extrusoras que son fabricadas por *Urola Solutions*. En este documento aparece la *Extrusora de Cuatro Zonas* a describir y ha sido realizado en base a datos reales sobre la empresa desarrolladora de la misma, por lo que finalmente ha sido la fuente no ontológica seleccionada.

- **Evaluación y selección de fuentes no ontológicas.** Por los motivos comentados en el punto anterior, estas dos actividades no han sido realizadas ya que desde un primer momento se ha seleccionado una única fuente no ontológica.

Actividades necesarias para el rediseño de las fuentes no ontológicas:

- **Ingeniería inversa de la fuente no ontológica.** Esta actividad consiste en el análisis de la información contenida en la fuente no ontológica para conseguir los distintos niveles de abstracción concretos de la misma (el diseño, los requisitos y la conceptualización). En este caso, partiendo del documento explicativo realizado por el grupo BDI, se han identificado las características concretas de los sensores implantados en la máquina *Extrusora de Cuatro Zonas*, como lo son las propiedades concretas que observa cada sensor, el tipo de datos que devuelven, etc. Con estas características se han creado unos requisitos concretos que deben ser abordados para la creación de la ontología.
- **Transformación de la fuente no ontológica.** A partir de los requisitos extraídos en la actividad anterior, se genera un modelo conceptual. Este modelo conceptual corresponde a las especializaciones de las clases sosa:Sensor, sosa:Observation y sosa:ObservableProperty realizadas en la ontología desarrollada para poder representar las características concretas de la máquina en cuestión. Estas especializaciones han sido explicadas en la sección 5.1.
- **Ingeniería directa de la ontología.** En esta actividad finalmente se realiza un diseño a partir del modelo conceptual mecionado, implementándolo con la herramienta Protégé y el lenguaje OWL 2 (Fig. 6.1).

Escenario 3: reutilizando fuentes ontológicas

En este escenario la metodología agrupa las actividades que reflejan la búsqueda y utilización de otras fuentes ontológicas que puedan ser interesantes en el desarrollo de la nueva ontología, evitando ”reinventar la rueda” y aprovechando además las ventajas que dichas ontologías ya definidas puedan ofrecer.

En este caso, se ha reutilizado una ontología diseñada para la representación general de los sensores y otra diseñada para la representación de las unidades de medida y propiedades cuantitativas que estos sensores pueden medir.

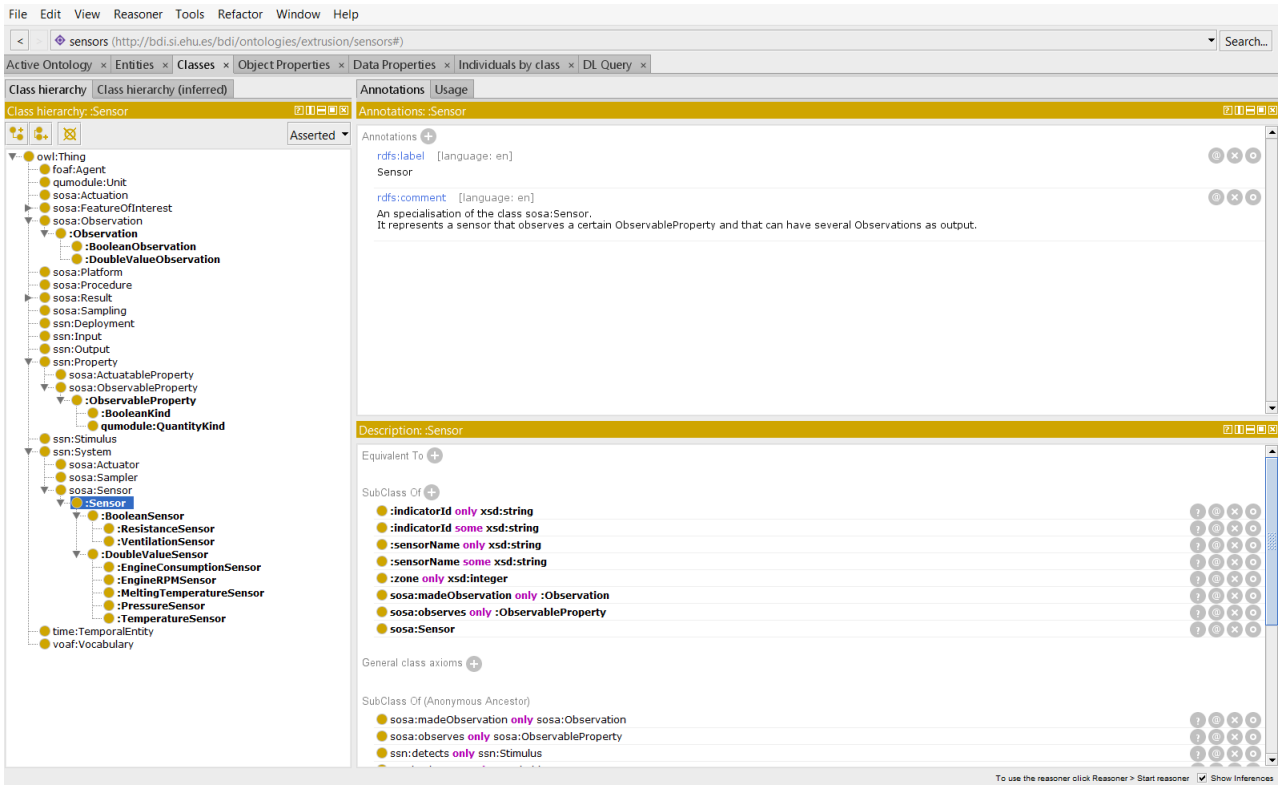

Figura 6.1: Captura del programa Protégé con la especificación de los sensores realizada.

Para el proceso de reutilización se han seguido cinco actividades principales:

- **Búsqueda de ontologías.** Para la representación de sensores encontramos la ontología *Semantic Sensor Network (SSN)*[3], una ontología desarrollada por el W3C. Esta ontología es altamente reconocida y utilizada en este campo, por lo que no se ha seguido buscando otra ontología para el mismo fin. En cuanto a la representación de las propiedades cuantitativas y las unidades de medida utilizadas en las mismas, por un lado encontramos la ontología *Quantities, Units, Dimensions and Types (QUDT)*[4], desarrollada originalmente para el proyecto *NASA Exploration Initiatives Ontology Models (NExIOM)*. Por otro lado, también encontramos la ontología *Ontology of units of Measure (OM)*[5] desarrollada por el departamento de ciencias computacionales de la Universidad *Vrije Universiteit (VU)* de Amsterdam.

- **Evaluación de ontologías.** Para la evaluación de las distintas ontologías se han tenido en cuenta los siguientes aspectos: capacidad de representar los conceptos necesarios del dominio concreto a modelar, grado de utilización de las mismas (buscando una mayor interoperabilidad) y fuente de procedencia las ontologías en cuestión. La ontología SSN cumple todos los requisitos impuestos, ya que la ontología desarrollada por el W3C permite la especificación de todas las características de los sensores de la extrusora y es una ontología altamente reconocida y utilizada. Por otro lado, la QUDT permite la especificación de las propiedades a observar y las unidades de medida y es una de las ontologías más utilizadas sobre dicho dominio, teniendo además un alto nivel de prestigio al haber sido desarrollada dentro de un programa colaborador de la NASA. Sin embargo, el nivel de detalle de la misma es excesivo en cuanto a lo realmente necesario para la ontología a desarrollar, y está compuesta por diferentes módulos y se pueden encontrar diferentes versiones de la misma, lo que dificulta la comprensión y reutilización de la misma. Por último, la OM permite la especificación de las propiedades a observar y las unidades de medida y está formada por una estructura sencilla de comprender. Sin embargo, el nivel de detalle de la misma es también excesivo para este dominio en concreto, y además la OM no es tan reconocida ni utilizada en el dominio en cuestión.

- **Comparación y selección de ontologías.** En base a la evaluación realizada en el punto anterior, se hace una comparación entre las dos ontologías candidatas sobre

[3]Documentación de la ontología *SSN*: https://www.w3.org/TR/vocab-ssn/
[4]Página web oficial de la ontología QUDT: http://qudt.org/
[5]Documentación de la ontología *OM*: http://www.ontology-of-units-of-measure.org/page/om-2

las unidades de medida para después seleccionar la más adecuada. Ambas tienen un gran nivel de detalle sobre las propiedades y las unidades de medida de las mismas, pero el reconocimiento y utilización general de la ontología QUDT hace que se incline la balanza hacia la misma. Por otro lado, dentro de la QUDT hay distintas versiones, y se ha encontrado que la versión más actual de la misma posee un gran número de errores. Por todo ello, se ha decidido que las ontologías a reutilizar serán la SSN y la QUDT 1.0. Una vez seleccionadas las ontologías a reutilizar, es importante decidir el modo de reutilización de las mismas. En este caso, se ha decidido que la SSN será reutilizada tal y como está definida y que, sin embargo, la QUDT necesitará de ciertos procesos de rediseño para poder ser reutilizada. Estos procesos de rediseño de la ontología QUDT se llevarán a cabo mediante las actividades definidas en el escenario 4.

- **Integración de ontologías.** Finalmente, las ontologías seleccionadas son integradas con el modelo ontológico desarrollado a partir de las fuentes no ontológicas, obteniendo el resultado final del desarrollo de la ontología.

Escenario 4: reutilizando y rediseñando fuentes ontológicas

Este escenario representa las actividades necesarias cuando las ontologías a reutilizar necesitan un proceso de rediseño antes de integrarlas en el resultado final.

Para ello, se realizan los procesos de reutilización especificados en el escenario 3 y, después de la selección de las ontologías, se ejecutan las actividades correspondientes al rediseño de las mismas.

En este caso, se han realizado estas actividades de rediseño para la ontología QUDT, que como se ha explicado en el escenario 3, servirá para la representación de propiedades cuantitativas y las unidades de las mismas. Esta ontología es excesivamente grande para lo que realmente se necesita representar en nuestro dominio concreto, por lo que se ha decidido extraer sólo la parte necesaria de la misma en un pequeño módulo. De esta manera, será este módulo creado el reutilizado por la ontología principal a desarrollar.

El rediseño de una ontología puede hacerse siguiendo distintos niveles de abstracción: especificación, conceptualización, formalización e implementación. En este caso, el rediseño de la ontología QUDT se ha realizado a nivel de conceptualización, ya que no se han modificado los requisitos o especificaciones de la misma, y simplemente se ha extraído un pequeño módulo de la misma que después se ha implementado.

Antes de realizar las actividades necesarias para el rediseño de la ontología, es necesario tener en mente cómo se va a querer utilizar el módulo resultante en la ontología principal a desarrollar.

La manera tradicional de representar en SSN el resultado obtenido en cada observación es mediante un objeto "*resultado*" que incluye el valor y la unidad de medida del mismo, en contraposición a una manera más simple que implicaría indicar un dato de tipo literal que especifique únicamente el valor.

En el dominio concreto a modelar, cada sensor mide únicamente una propiedad y siempre en la misma unidad de medida, por tanto todos sus resultados serán sobre la misma propiedad y en la misma unidad de medida. Teniendo en cuenta los grandes volúmenes de datos que se manejan y el rendimiento de la futura aplicación que utilizará la ontología, si se utilizara la manera de representación tradicional, el incrementar en tres las tripletas necesarias para anotar cada observación afectaría directamente al tiempo necesitado para dicha inserción, así como en el espacio necesario para el almacenamiento de la información.

Por todo ello, se ha decidido simplificar el método de utilización de la ontología, de manera que tanto la propiedad observada como la unidad de medida utilizada se definirán únicamente en cada uno de los sensores. Después, en los resultados de cada observación se definirá directamente el valor de dicha observación junto con el *timestamp* concreto.

Con este objetivo en mente, son tres las actividades a llevar a cabo en este proceso de re-diseño:

- **Ingeniería inversa de la fuente ontológica.** En esta actividad, a partir de la implementación de la ontología QUDT, se pretende llegar al nivel de conceptualización de la misma, realizando una ingeniería inversa para ello. Centrándonos en las partes que más interesan para este TFG, en la QUDT están definidas distintas propiedades cuantitativas, como pueden ser la temperatura, la presión, la velocidad, etc. Por otro lado, se representan las diferentes unidades de medida que pueden ser utilizadas para la medición de dichas propiedades. Como unión entre ambos conceptos, la QUDT define un rango de posibles unidades de medida que pueden ser utilizadas para cada propiedad cuantitativa.

- **Re-estructuración de la fuente ontológica.** Una vez entendida la estructura conceptual de la ontología QUDT, se reestructura la misma. Para ello, se ha decidido extraer únicamente las instancias que representan las propiedades y unidades de

medida que serán utilizadas en la ontología principal. Con la propiedad y unidad de medida relacionadas con la temperatura sin embargo, se han encontrado ciertos inconvenientes. En la QUDT 1.0, curiosamente, no hay ninguna representación de la temperatura habitual, sólo se modela la *temperatura absoluta*. Por esto, la propiedad de temperatura y la unidad de medida grados centígrados o *Celsius* han sido extraídas de otra ontología[6], cuya base es muy similar a la QUDT, evitando así tener que crear dichas propiedades y siguiendo con la dinámica de la reutilización.

- **Diseño de la fuente ontológica.** Definida la nueva conceptualización, en esta actividad se realiza un proceso de diseño llegando así a la implementación del módulo de la QUDT. En este caso, se han creado dos clases principales, *qu:Unit* y *qu:QuantityKind*, que agruparán las instancias extraídas explicadas. Finalmente, se ha pasado a la implementación del módulo en el lenguaje OWL 2 y utilizando Protégé para ello, al igual que el resto de la ontología (Fig. 6.2).

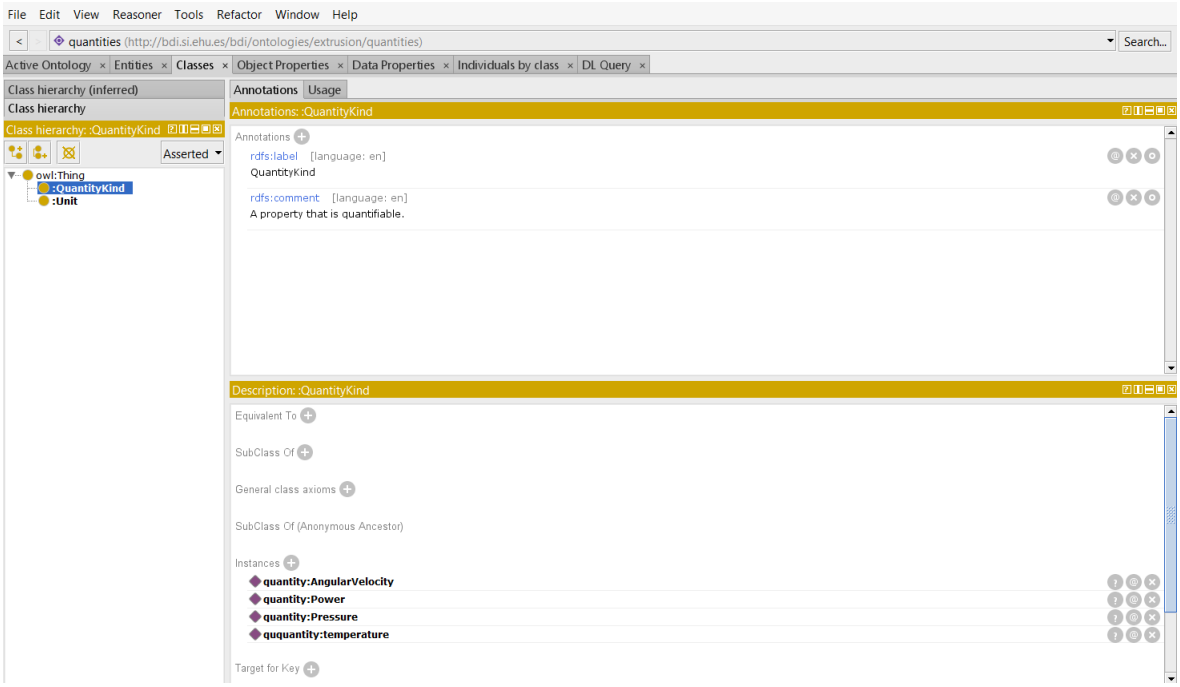

Figura 6.2: Captura del programa Protégé en la implementación del módulo *quantities*.

[6]Ontología utilizada para las instancias relacionadas con la temperatura: https://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu-rec20.html

Resultados

Como resultado de la explicación de la metodología NeOn, se ha desarrollado la ontología descrita anteriormente en el apartado 5.1.

En los siguientes enlaces se pueden encontrar la ontología concreta descrita y el módulo desarrollado para las propiedades y unidades de medida:

- Ontología *sensors* desarrollada: http://bdi.si.ehu.es/bdi/ontologies/extrusion/sensors
- Módulo *quantities* desarrollado: http://bdi.si.ehu.es/bdi/ontologies/extrusion/quantities

Por último, en las tablas 6.3 y 6.2 se detallan las métricas concretas de ambas ontologías obtenidas a través de Protégé. En estas tablas se han omitido aquellas propiedades que tenían valor nulo.

Tabla 6.2: Métricas del módulo desarrollado para las propiedades a observar (*quantities*)

Ontología *quantities*	
Metrics	
Axiom	23
Logical axiom count	9
Declaration axioms count	10
Class count	2
Individual count	8
Annotation Property count	5
DL expressivity	ALCROIN(D)
Class axioms	
DisjointClasses	1
Individual axioms	
ClassAssertion	8
Annotation axioms	
AnnotationAssertion	4

Tabla 6.3: Métricas de la ontología final desarrollada (*sensors*)

Ontología *sensors*	
Metrics	
Axiom	855
Logical axiom count	278
Declaration axioms count	128
Class count	40
Object property count	38
Data property count	10
Individual count	27
Annotation Property count	21
DL expressivity	ALCROIN(D)
Class axioms	
SubClassOf	153
DisjointClasses	5
Object property axioms	
SubObjectPropertyOf	1
InverseObjectProperties	14
ObjectPropertyDomain	2
ObjectPropertyRange	2
SubPropertyChainOf	4
Data property axiom	
DataPropertyDomain	8
DataPropertyRange	9
Individual axioms	
ClassAssertion	31
DataPropertyAssertion	46
Annotation axioms	
AnnotationAssertion	396

Comprobación de la ontología: OOPS!

Por último, aunque este paso no entra en la metodología seguida, para asegurar un buen resultado final se ha utilizado la herramienta OOPS! (OntOlogy Pitfall Scanner)[7] Esta herramienta ayuda a detectar algunos de los más comunes errores cometidos al desarrollar ontologías, especificando las causas de los errores y las posibles soluciones de los mismos [Poveda-Villalón et al., 2014].

Con ayuda de esta herramienta, se han conseguido identificar y solucionar ciertos errores cometidos en el desarrollo de la ontología, como rangos y dominios de propiedades sin especificar, falta de comentarios o etiquetas, etc. Los errores generados a partir de las ontologías importadas no han sido tenidos en cuenta.

6.2. Creando la aplicación web con React y Materialize

Para el desarrollo del sistema web, se ha utilizado React.js para la implementación de la interfaz y de gran parte de la lógica de la aplicación, con ayuda de Materialize para añadir estilo a dicha interfaz.

A continuación se detallan los primeros pasos dados para la creación de la aplicación, así como los conceptos básicos necesarios para entender las tecnologías utilizadas.

React.js es una librería de Javascript para el desarrollo de interfaces de usuario y de la lógica de la aplicación, y está basada en el concepto de aplicaciones de una sola página (SPA) ya explicado en capítulos anteriores de la memoria.

Antes de comenzar con la explicación del desarrollo de aplicaciones React, es necesario comentar que las aplicaciones implementadas con React utilizan *Node.js*[8]. Node.js es un entorno multiplataforma y de código abierto (open-source) que permite la ejecución de código JavaScript fuera del navegador. Node permite así una manera sencilla y rápida de implementar tanto el código del cliente como el del servidor, todo con JavaScript.

6.2.1. Configuración inicial de una aplicación con React

Siguiendo las recomendaciones del tutorial ofrecido en la página web oficial de React[9], se ha utilizado el entorno *Create React App*[10] para la creación de la aplicación de una

[7] OOPS!: http://oops.linkeddata.es/
[8] Node.js: https://nodejs.org/en/
[9] Tutorial de React.js: https://reactjs.org/tutorial/tutorial.html
[10] Create React App: https://github.com/facebook/create-react-app

sola página. Create React App ofrece la posibilidad de crear aplicaciones de una sola página con React de una forma rápida y sencilla y crea un entorno de desarrollo para la construcción y ejecución de dicha aplicación. Para todo ello, Create React App hace uso de las herramientas *Babel*[11] y *Webpack*[12], pero esto es transparente al usuario por lo que no es necesario tener conocimiento de dichas herramientas.

Hay distintas posibilidades de instalar y utilizar el entorno Create React App, pero en este caso se hará uso del administrador de paquetes *npm*[13].

Cabe mencionar que, para poder instalar Create React App correctamente, es necesario tener instalada la versión más reciente de Node.js.

Una vez comprobado esto último, ejecutando lo siguiente en la línea de comandos se instalará el entorno:

```
npm install -g create-react-app
```

Así mismo, para crear una aplicación React se ejecutará el siguiente comando en la ubicación deseada:

```
create-react-app proyecto-tfg
```

Esto creará una carpeta llamada *proyecto-tfg* y que contendrá todos los archivos y librerías necesarias para el desarrollo de una aplicación de una sola página con React.

Por último, para mantener siempre una copia de seguridad del proyecto, se ha decidido inicializar un repositorio *git* en la carpeta principal del proyecto. Para ello, se ejecuta el comando `git init` dentro de la carpeta principal del proyecto con la ayuda de la línea de comandos proporcionada por Git.

En la figura 6.3 se presenta finalmente la estructura de archivos resultante después de los pasos detallados. A continuación se explica brevemente el objetivo de cada una de las carpetas:

- **.git**. Representa el repositorio *git* asociado al proyecto.
- **build**. Es la carpeta destino en la que se construye la aplicación en modo de producción.

[11] Babel: http://babeljs.io/
[12] Webpack: https://webpack.js.org/
[13] npm: https://www.npmjs.com/

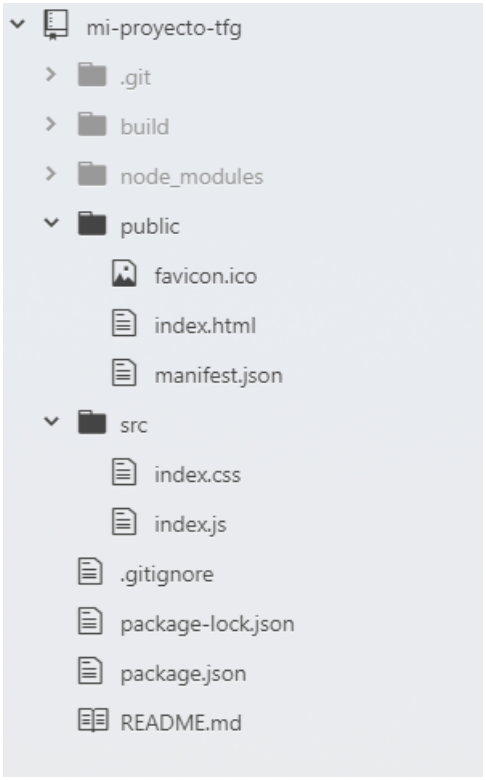

Figura 6.3: Estructura de archivos resultante tras la creación de la aplicación React

- **node_modules**. En esta carpeta se encuentran las diferentes librerías y módulos de Node.js que la aplicación podrá utilizar.

- **public**. En esta carpeta se encuentran los archivos estáticos de la aplicación. Dentro de esta carpeta se posiciona el fichero `index.html`, el cual consta únicamente de la plantilla de la aplicación a desarrollar. Este fichero podrá utilizar únicamente de los archivos contenidos en esta carpeta y es necesario que exista para la correcta construcción de la aplicación con Create-React-App.

- **src**. En esta carpeta se encuentra todo el código implementado de la aplicación. El fichero principal de la misma será `index.js`, el cual será el punto de entrada del código JavaScript. El fichero index.css contendrá el código CSS principal de la aplicación. Dentro de esta carpeta además, se podrán posicionar todos los archivos JavaScript y CSS adicionales deseados, sin necesidad de seguir una estructura de ficheros determinada.

Una vez que la aplicación ha sido correctamente creada, el entorno Create React App ofrece una serie de comandos integrados para la construcción y ejecución de la aplicación:

- **`npm start`**
  Ejecutando dicho comando en la carpeta principal del proyecto, el entorno construye y ejecuta la aplicación en modo de desarrollo. La aplicación estará por tanto visible en:

  `http://localhost:3000`

  Ejecutando la aplicación en este modo, todos los cambios efectuados en el código de la aplicación (todos los cambios realizados en los ficheros de la carpeta src) serán automáticamente recompilados y recargados en la página del navegador.

- **`npm run build`**
  Ejecutando dicho comando en la carpeta principal del proyecto, el entorno construye la aplicación en modo de producción en la carpeta `build` del proyecto.

6.2.2. Principios del desarrollo con React

Para explicar el desarrollo realizado, ya que una explicación paso a paso de todo el código realizado es imposible, en esta sección se detallan los principios básicos de la implementación con React. De esta manera, se podrá tener una idea general de las características y técnicas utilizadas en la implementación de la aplicación.

React se basa en la implementación de distintas clases o componentes que representan distintas partes de la interfaz. Después, de manera dinámica durante la ejecución de la aplicación, se decide cuáles de dichos componentes son renderizados dependiendo de ciertas condiciones.

En la implementación, estos componentes se definen como subclases de la clase ya definida `'React.Component'`. Estas clases pueden tener como entrada distintos parámetros mediante el objeto `props` (que proviene de propiedades) y devuelven, a través del método `render`, la descripción de lo que se quiere visualizar por pantalla.

Para la simplificación a la hora de implementar estas estructuras en React, React presenta la sintaxis *JSX*[14], una extensión de JavaScript adaptada a las características concretas de React. Esta sintaxis permite la creación de elementos React, objetos de JavaScript que pueden almacenarse en variables y/o pasarse como parámetro de las distintas funciones. Además, JSX engloba todo el potencial de JavaScript ya que cualquier expresión de JavaScript puede expresarse especificándose entre *llaves* ('{' y '}') dentro del código JSX.

[14]Introducción a la sintaxis JSX: https://reactjs.org/docs/introducing-jsx.html

En la figura 6.4 se presenta una parte simplificada de un componente de React usado en la aplicación descrito con JSX. Dicho elemento utiliza, entre otros, dos parámetros de entrada llamados `infoSensors` y `selectedSensors`. Estos parámetros, como ya se ha explicado, los recibe a través del objeto `props` del componente. En el ejemplo concreto simplificado, estos parámetros sólo se almacenan en dos variables con el mismo nombre, pero pueden ser utilizados y tratados como cualquier objeto de JavaScript.

```
class InformationQueryForm extends React.Component{
    ...

    render(){
        const selectedSensors = this.props.selectedSensors;
        const infoSensors = this.props.infoSensors;
        ...

        return(
                <div>
                    ...
                </div>
        )
    }
}
```

Figura 6.4: Parte simplificada del componente React *InformationQueryForm* descrito con la sintaxis JSX.

Para renderizar un elemento de React, simplemente se debe expresar el nombre del componente deseado dentro de una etiqueta HTML, indicando los parámetros que utiliza como propiedades de la etiqueta.

En la figura 6.5 se muestra un ejemplo concreto, en el que se renderiza el elemento mostrado en la figura 6.4 dentro de otro componente llamado `SelectQueryTabs`. En este ejemplo, los parámetros pasados al componente `InformationQuery` corresponden con variables JavaScript inicializadas anteriormente.

```
class SelectQueryTabs extends React.Component{
    ...

    render(){
        let selectedSensorsVar = ["idSensor1", "idSensor1", ...]
        let infoSensorsVar = [{"id":"idSensor1", "name":"Nombre Sensor 1", ...}, ...]
        ...

        return(
            <div>
                ...
                <InformationQueryForm
                    selectedSensors={selectedSensors}
                    infoSensors={infoSensors}
                    ...
                />
                ...
            </div>
        )
    }
}
```

Figura 6.5: Parte simplificada del componente React *SelectQueryTabs*, el cual utiliza el componente descrito en la figura 6.4.

Por otro lado, los componentes de React pueden tener asignado un estado. Dicho estado está expresado como un objeto JSON y puede ser utilizado y/o cambiado dentro de las diferentes funciones y/o métodos del componente. Para la modificación de este estado se ha de hacer uso del método `setState()`, el cual puede ejecutarse de forma asíncrona y recibe como parámetro un objeto con la variable del estado que se quiere modificar y su nuevo valor. React incorporará el objeto recibido en dicho método al estado actual del componente, de manera que, aunque se tenga definido un estado con más de una variable, solo es necesario indicar la variable modificada en el método `setState()`, pudiendo omitir el resto.

En la figuras 6.6 y 6.7 se muestra parte simplificada de la implementación de un componente React con un estado definido en el mismo. Dicho componente corresponde a la página principal de la aplicación, donde el usuario puede decidir qué funcionalidad desea realizar, y tiene las siguientes características:

- En el método **`constructor`** del componente se inicializa su estado, de manera que inicialmente el valor de la variable `selectedPage` del estado será `'preguntas'`.

- Las funciones **`mostrarPreguntas`** y **`mostrarTraductorDatos`** definidas en el componente, las cuales simplemente cambian el valor de la variable `selectedPage` definida en el estado.

- En el método **`render`**, se tiene en cuenta el estado del componente para inicializar la variable `content`. Si el valor del estado corresponde con *preguntas*, la variable contendrá el componente `QueriesPage`, y de lo contrario contendrá el componente `DataPage`. Además, en el componente se define una variable `sideNav`, que hace referencia al menú lateral de la aplicación, donde se podrá seleccionar qué página mostrar. En algún punto de la definición de esta variable, se indica que al pulsar el botón de la opción deseada se ejecute la función adecuada definida en el componente (`mostrarPreguntas()` o `mostrarTraductorDatos()`). Por último, se devuelven todas las variables inicializadas mediante el método `return()`, de manera que dependiendo del estado del componente se mostrará el contenido adecuado en la página (`QueriesPage` o `DataPage`).

```
class SelectedPage extends React.Component {
    constructor(props){
        super(props);
        this.state = {
            selectedPage: 'preguntas',
            ...
        };
    }
    ...

    mostrarPreguntas(){
        this.setState({
            selectedPage: 'preguntas',
        });
    }

    mostrarTraductorDatos(){
        this.setState({
            selectedPage: 'datos',
        });
    }
```

Figura 6.6: Parte de un componente de React con un estado definido y descrito con JSX - Parte 1.

```
    render(){
        const content = (this.state.selectedPage === 'preguntas')
            ? (<QueriesPage ... />)
            : (<DataPage ... />);

        const sideNav =
            (...<li>
                    <button onClick={() => this.mostrarPreguntas()}>
                        Consultar datos
                    </button>
                </li>
                <li>
                    <button onClick={() => this.mostrarTraductorDatos()}>
                        Insertar datos
                    </button>
                </li>...);
        ...

        return(
            <div>
                <div className='navBar'>
                    ...
                    {sideNav}
                </div>
                {content}
            </div>
        )
    }
}
```

Figura 6.7: Parte de un componente de React con un estado definido y descrito con JSX - Parte 2.

Por otro lado, otra de las claves de la implementación con React es el ciclo de vida de los componentes. Cuando un componente es renderizado por primera vez se dice que el componente ha sido "*montado*"(*mounted* en inglés). Cuando el componente deja de estar renderizado sin embargo, quitándolo del DOM, se dice que el componente ha sido "*desmontado*"(*unmounted* en inglés).

En los componentes de React se pueden entonces declarar métodos especiales para que se ejecute cierto código siguiendo el ciclo de vida del mismo:

- `componentDidMount()`. Código a ejecutar cuando el componente ha sido *montado*.

- `componentDidUnmount()`. Código a ejecutar cuando el componente ha sido *desmontado*.

En la figura 6.8 se muestra un ejemplo de un componente React de la aplicación en el que se ha utilizado la función `componentDidMount()`. En esta función se recopila la información necesaria sobre las máquinas que se tienen en la organización correspondiente a partir del parámetro `idOrg`, para después almacenarla en el estado del componente y así utilizarla en el resto del código del mismo.

```
export class QueriesPage extends React.Component {
    constructor(props){
        super(props);
        this.state = {
            machines: {},
            ...
        };
    }

    componentDidMount(){
        const idOrg = this.props.idOrganization;

        const maquinas = // Recopilar información de las máquinas de la
                         // organización cuyo id es 'idOrg'

        this.setState({
            machines: maquinas,
        });
    }

    ...
}
```

Figura 6.8: Ejemplo de utilización de la función *componentDidMount()* en un componente de React.

Por último, en la aplicación a desarrollar en este TFG los formularios son un parte muy importante de la misma, ya que todas las funcionalidades implementadas recogen cierta información proporcionada por el usuario a partir de formularios. Los formularios de HTML por defecto dirigen la información introducida por el usuario a una nueva página una vez que los datos han sido enviados, pero esto sin embargo difiere directamente de los principios de las aplicaciones de una sola página. Por ello, React propone una técnica

llamada ”componentes controlados” (*controlled components* en inglés)[15] para manejar la información de los diferentes formularios siguiendo el paradigma de las aplicaciones de una sola página.

Dicha técnica consiste en controlar directamente cada elemento del formulario con la ayuda de uno o varios métodos definidos en el componente y guardando la información de cada uno en el estado del componente. Para ello se define que, con la ayuda del evento `onChange`, cada vez que cambie el valor de uno de los elementos del formulario se guarde el nuevo valor en una variable de estado asignada a dicho elemento, de manera que en el estado del componente estará definida en todo momento la información introducida en el formulario.

Para tratar el envío del formulario, se define otro método en el componente que implemente el código deseado, para así poder ejecutar dicho método cuando ocurra el evento deseado, como podría ser hacer click en un botón o enlace personalizado, etc. Si para ejecutar dicho método se hace uso del evento `onSubmit` del formulario, el cual se ejecuta al hacer *click* sobre la etiqueta `<input type="submit>`, se deberá hacer referencia a `event.preventDefault();` en el código del nuevo método creado, para así evitar que se ejecute lo definido por defecto en dicho evento.

En las figuras 6.9 y 6.10 se muestra un ejemplo de la implementación de un formulario utilizando la técnica de los *componentes controlados*. Este ejemplo corresponde con parte del componente `InformationQueryForm` de la aplicación, en el que se define el formulario de consultas de información general. En este, para controlar los valores que introduce el usuario para filtrar las fechas, se han creado las variables de estado `fechaInicio` y `fechaFin`, donde se almacenará dicha información introducida. Para almacenar la información se utilizan los métodos `handleFechaInicio()` y `handleFechaFin()` en las etiquetas del formulario correspondientes. Por último, con el método `handleSubmit()` se ejecuta el método correspondiente al envío del formulario utilizando la información almacenada en el estado del componente.

[15] Documentación de la utilización de formularios en React: https://reactjs.org/docs/forms.html

```
class InformationQueryForm extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            fechaInicio: '',
            fechaFin: '',
            ...
        };
    }

    handleFechaInicio(event, value){
        ...
        this.setState({
            fechaInicio: value,
            ...
        });
    }

    handleFechaFin(event, value){
        ...
        this.setState({
            fechaFin: value,
            ...
        });
    }

    handleSubmit(){
        const fechaInicio = this.state.fechaInicio;
        const fechaFin = this.state.fechaFin;
        ...
        this.props.getInformationQuery({'startDate':fechaInicio,'endDate':fechaFin,...}, ...);
    }
```

Figura 6.9: Ejemplo de implementación de un formulario con la técnica *componentes controlados* - Parte 1.

```
    ...

    render(){
        ...
        return(
                ...
                <li>
                    <div ...>
                        Filtrar resultados por fechas
                    </div>
                    <div ...>
                        <input type='date' label="Desde..."
                            onChange={(e, value) => {this.handleFechaInicio(e, value);}}/>
                    </div>
                    <div>
                        <input type='date' label="Hasta..."
                            onChange={(e, value) => {this.handleFechaFin(e, value);}}/>
                    </div>
                </li>
                ...
                <li>
                    <button  onClick={() => {this.handleSubmit();}}>
                        Consultar
                    </button>
                </li>
                ..
        )
    }
}
```

Figura 6.10: Ejemplo de implementación de un formulario con la técnica *componentes controlados* - Parte 2.

Para la subida de ficheros en los formularios sin embargo, la etiqueta `<input type="file">` de HTML es únicamente de lectura, de manera que su valor sólo puede ser cambiado por el usuario, por lo que ésta no puede ser tratada con la técnica explicada. Esta etiqueta pertenece a los llamados "componentes descontrolados" (*uncontrolled components* en inglés)[16] de React.

Para hacer frente a esto, React propone hacer uso del atributo `ref`, que permite hacer referencia al elemento que lo contiene como si se tratase de un identificador del mismo. En la figura 6.11 se muestra parte de la implementación del componente `InsertDataform` de la aplicación, el cual corresponde con el formulario de subida de ficheros de datos para su inserción y en el que se hace uso del atributo `ref` mencionado.

[16]Documentación sobre los componentes descontrolados en React: https://reactjs.org/docs/uncontrolled-components.html

```
class InsertDataForm extends React.Component {
    constructor(props) {
        super(props);
        this.fileInput = React.createRef();
        this.state = {
            ...
        }
    }

    handleInsertData(){
        const selectedFile = this.fileInput.files;
        const fileName = selectedFile[0].name;
        ...
    }

    render(){
        ...

        return(
            <div>
                ...
                Fichero
                <input type="file" ref={input => {this.fileInput = input;}}/>
                ...
                <button onClick={() => this.handleInsertData()}>
                    Comenzar con la inserción de datos.
                </button>
            </div>
        )
    }
```

Figura 6.11: Parte de la implementación de un formulario con subida de ficheros.

Partiendo de estas bases, se han desarrollado con React las interfaces y gran parte de la lógica de la aplicación web de este TFG, utilizando para ello la sintaxis JSX comentada junto con el código JavaScript necesario para las diferentes funcionalidades.

6.2.3. Añadiendo estilo con Materialize

Para el diseño del estilo de nuestro sistema web, como bien se ha explicado anteriormente, se ha utilizado el marco de trabajo o *framework* Materialize.

Materialize es un *framework* de CSS que está basado en el concepto de *Material Design* de Google y que ofrece una serie de estilos, animaciones y diseños ya definidos y listos para utilizar. Todo esto está creado utilizando HTML, CSS y JavaScript, por lo que es

fácil de usar y entender. En la página web oficial de Materialize[17] se pueden encontrar todos los componentes, estilos y métodos disponibles y la documentación necesaria para su utilización.

La instalación de Materialize puede hacerse de distintas maneras, pero en nuestro caso, se ha utilizado la herramienta npm. Para ello se ha ejecutado el siguiente comando en la carpeta principal del proyecto:

```
npm install materialize-css
```

Una vez instalado, basta con importarlo al comienzo del archivo de JavaScript deseado para poder utilizarlo:

```
import M from 'materialize-css'
```

En la figura 6.12 se muestra un ejemplo de cómo crear un botón añadiendo estilo a una etiqueta `a` de HTML utilizando las clases proporcionadas por Materialize, junto con la visualización correspondiente del componente. Dicho ejemplo corresponde con el botón de consultar de los diferentes formularios de consulta de la aplicación.

```
<a className='btn blue darken-3' ...>
    Consultar
    <i className='material-icons right'>
        bar_chart
    </i>
</a>
```

Figura 6.12: Ejemplo de utilización de Materialize para crear un botón con una etiqueta `a` de HTML.

Para facilitar el uso de Materialize en las aplicaciones creadas con React, la librería *react-materialize*[18] proporciona una serie de componentes React ya definidos basados en Materialize, de manera que al importar los diferentes componentes éstos ya tienen el estilo de Materialize asociado. En la figura 6.13 se muestra la definición del mismo botón de consulta anterior pero utilizando la librería react-materialize.

[17] Documentación de Materialize: https://materializecss.com/
[18] react-materialize: https://react-materialize.github.io/#/

```
<Button className='blue darken-3' ... >
    Consultar
    <Icon right>
        bar_chart
    </Icon>
</Button>
```

Figura 6.13: Ejemplo de utilización de los componentes React `Button` y `Icon` de la librería react-materialize.

Para la utilización de dicha librería, al igual que con Materialize, basta con importar los componentes que se utilizarán en el fichero JavaScript correspondiente. Para el ejemplo presentado en la figura 6.13 se importarían los componentes `Button` e `Icon`:

```
import {Button, Icon} from 'react-materialize'
```

En el desarrollo de este TFG se ha hecho uso de Materialize mediante ambas opciones, decidiendo la utilización de cada una dependiendo del problema concreto a abordar.

Además, en el fichero `index.css` de la carpeta src, se han añadido ciertos estilos propios y pequeñas modificaciones a los estilos proporcionados por Materialize, como pueden ser cambios en ciertos colores predefinidos, tamaños de márgenes, etc.

6.3. Configuración de Virtuoso OpenLink en local

Para el almacenamiento de los datos RDF que serán necesarios en el sistema web, como bien se ha comentado antes, se ha decidido la utilización de Virtuoso OpenLink.

Virtuoso OpenLink, la versión *Open-Source* o *de código libre* de Virtuoso, tiene una extensa documentación[19] disponible en la *web*, donde se detallan la infinidad de posibilidades y características que ofrece la herramienta. En el caso de este TFG, sólo se utilizarán los servicios relacionados con el almacenamiento y la consulta de datos RDF.

Para la instalación de Virtuoso OpenLink, Virtuoso ofrece en su documentación distintas guías dependiendo del sistema operativo utilizado en la máquina destino de la instalación. En este caso, el sistema operativo de la máquina es Windows, por lo que se ha seguido la guía de instalación en Windows[20].

[19] Documentación de Virtuoso OpenLink: http://vos.openlinksw.com/owiki/wiki/VOS/

[20] Guía de instalación de Virtuoso OpenLink en Windows: http://vos.openlinksw.com/owiki/wiki/VOS/VOSUsageWindows

Para poder utilizar los servicios ofrecidos sin embargo, es necesario crear una instancia del Servicio Windows de Virtuoso en nuestro equipo.

Una vez que la instancia ha sido creada, ésta se puede administrar a través del panel Administrador de Servicios de Windows estándar o a través de los comandos CMD/DOS proporcionados por Virtuoso (start, stop, delete, etc.)

6.3.1. Administración y configuración de Virtuoso

Una vez que la instancia del servicio está iniciada, Virtuoso ofrece una herramienta accesible desde el navegador HTTP para la administración de dicha instancia.

Esta herramienta, llamada Virtuoso Conductor, está disponible en la URL:

```
http://<direccion-IP-del-servidor-Virtuoso>:<puerto-HTTP>/conductor
```

En este caso concreto, la URL con la que accederemos al Conductor será:

```
http://localhost:8890/conductor
```

Una vez que se ha accedido a dicha herramienta (Fig. 6.14), proporcionando la contraseña y usuario asignados por defecto podremos configurar características de los parámetros de Virtuoso, administrar los datos almacenados en el mismo, configurar opciones del servicio web ofrecido, etc.

El objetivo principal de Virtuoso en nuestro caso es el almacenamiento de datos en RDF, y dichos datos estarán basados en la ontología anteriormente desarrollada y explicada. Por ello, como primer paso se ha almacenado dicha ontología en un nuevo grafo, de manera que este será el grafo en el que se almacenarán los datos a utilizar en la aplicación mediante peticiones HTTP.

Para la administración de los datos, Virtuoso ofrece la herramienta *Virtuoso Interactive SQL (iSQL)*, basada como su nombre indica, en un SQL interactivo, el cual contiene una serie de bases de datos y métodos ya incorporados. Esta herramienta está disponible a través del Conductor o de la línea de comandos (Fig. 6.15). En este caso, se utilizará la línea de comandos para su utilización.

Para la inserción de ficheros RDF a un grafo determinado mediante iSQL, en la documentación especificada anteriormente de Virtuoso podemos encontrar distintas guías que abordan este tema[21].

[21] Documentación seguida en este caso para la inserción de datos RDF en un grafo: http://vos.openlinksw.com/owiki/wiki/VOS/VirtBulkRDFLoader

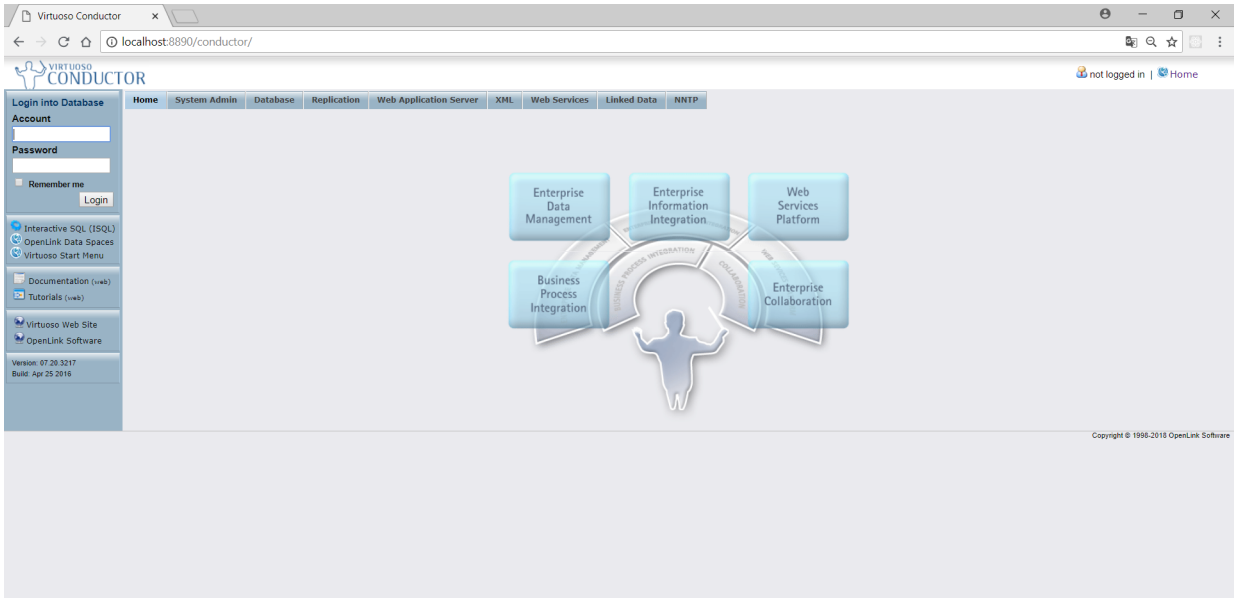

Figura 6.14: Página de incicio de la herramienta Virtuoso Conductor.

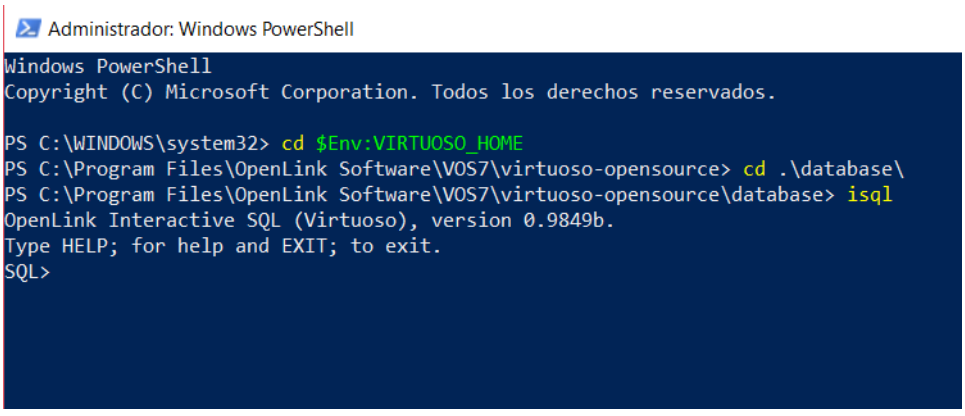

Figura 6.15: Acceso a la herramienta iSQL a través de la línea de comandos.

Finalmente, los pasos que se han seguido para la creación de un nuevo grafo con la ontología desarrollada son los siguientes:

1. Crear una carpeta en uno de los directorios permitidos en el parámetro `DirsAllowed` del fichero de configuración `virtuoso.ini`.

2. Crear en dicha carpeta un fichero *global.graph* con la URI del grafo a la que queremos subir todos los datos. Aunque en nuestro caso solo tengamos un fichero RDF a insertar, de esta manera se define que aquellos ficheros RDF a insertar que no tengan un fichero de extensión *".graph"* asociado tendrán como destino la URI indicada en *global.graph*. En este caso, la URI seleccionada para nuestro grafo es:

   `http://bdi.si.ehu.es/bdi/ontologies/extrusion/sensors#`

3. Añadir también a la carpeta los datos RDF que queremos subir. En este caso, se trata del fichero de la ontología creada, especificada en formato Turtle (extensión .ttl).

4. Acceder a la herramienta iSQL y ejecutar los comandos necesarios para cargar los datos mencionados en el grafo especificado. (Fig. 6.16)

```
PS C:\Program Files\OpenLink Software\VOS7\virtuoso-opensource\database> isql
OpenLink Interactive SQL (Virtuoso), version 0.9849b.
Type HELP; for help and EXIT; to exit.
SQL> ld_dir('ontologia', '*.ttl', 'http://bdi.si.ehu.es/bdi/ontologies/extrusion/sensors#');
Connected to OpenLink Virtuoso
Driver: 07.20.3217 OpenLink Virtuoso ODBC Driver

Done. -- 1047 msec.
SQL> rdf_loader_run();

Done. -- 7391 msec.
SQL>
```

Figura 6.16: Insertar los ficheros de extensión *.ttl* de la carpeta *datos_ontologia* a un nuevo grafo con la URI especificada.

Después de esto, accediendo al Conductor se puede comprobar cómo el nuevo grafo especificado ha sido creado correctamente. (Fig. 6.17)

Por otro lado, Virtuoso también ofrece un servicio SPARQL Endpoint para la realización de consultas de datos RDF, al que se puede acceder mediante la ruta reservada `/sparql` del servicio web.

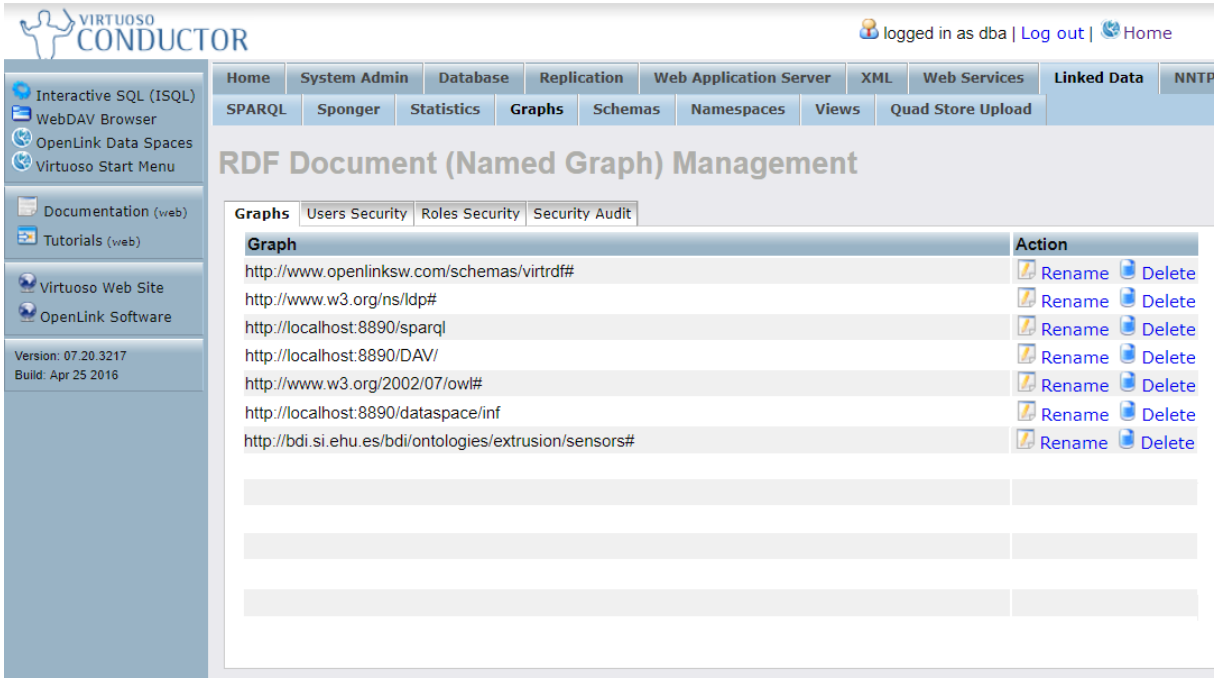

Figura 6.17: Grafos disponibles en la Instancia de Virtuoso creada de manera local.

En nuestro caso, se hará uso de dicho servicio de consultas SPARQL mediante peticiones HTTP. En la documentación de Virtuoso se encuentra un apartado que detalla la utilización de este servicio de consultas SPARQL mediante el protocolo HTTP[22], especificando los métodos, parámetros, formatos de respuesta disponibles, etc.

Por último, para que dichas peticiones HTTP que se hacen desde la aplicación al servidor web de Virtuoso funcionen correctamente, es necesario realizar ciertas configuraciones:

- **Modificar el Cross-Origin Resource Sharing**. Virtuoso utiliza *Cross-Origin Resource Sharing (CORS)*[23] como control de acceso HTTP. Por defecto sin embargo, no se especifica ningún dominio al que permitir el acceso, por lo que si no se modifica, ninguna petición HTTP generada desde un origen distinto al del propio servidor de Virtuoso será aceptada. En este caso, modificaremos el CORS correspondiente al servicio de consultas SPARQL de Virtuoso, ya que será el servicio utilizado, y especificaremos "*" en el mismo, permitiendo así el acceso a este servicio desde cualquier origen.

- **Aumentar el número máximo de tripletas a devolver**. La configuración por defecto de Virtuoso tiene especificado que el número máximo de tripletas a devolver al utilizar el servicio de SPARQL Endpoint es 10000. Esto sin embargo no es suficiente para la cantidad de datos que tratamos en nuestra aplicación, por lo que se ha aumentado dicho valor máximo a 800000 tripletas. Esta modificación se puede hacer directamente en el fichero de configuración virtuoso.ini o mediante el Virtuoso Conductor.

- **Habilitar la inserción de datos en la ruta /sparql** En la ruta `/sparql`, como bien se ha especificado antes, es posible hacer distintos tipos de consultas SPARQL, pero sin embargo, por razones de seguridad, Virtuoso tiene deshabilitada la opción de insertar datos a través de dicha ruta.
  En nuestro caso, una de las funcionalidades del sistema web es la inserción de datos, por lo que se ha buscado una manera de poder hacer inserciones de datos a través de peticiones HTTP para poder seguir con el mismo diseño de la arquitectura del sistema web. Esto ha dado un gran número de problemas dada a la escasa información encontrada sobre el tema en los documentos oficiales de Virtuoso, pero

[22] Documentación del servicio de consultas SPARQL de Virtuoso: http://vos.openlinksw.com/owiki/wiki/VOS/VOSSparqlProtocol

[23] El *intercambio de recursos de origen cruzado* o *CORS (Cross-Origin Resource SHaring)* es un mecanismo que utiliza encabezados adicionales HTTP para permitir que una petición solicite un recurso desde un dominio distinto, un protocolo o un puerto diferente al del documento que la generó.

estos problemas se detallarán más adelante (concretamente, en la sección 8.2). Finalmente, se ha conseguido habilitar la opción de inserción de datos en la ruta `/sparql` ejecutando la siguiente sentencia en la herramienta iSQL de Virtuoso:

```
GRANT EXECUTE ON DB.DBA.L_O_LOOK TO "SPARQL";
```

Esta sentencia provee los permisos necesarios a los usuarios que utilicen la ruta `/sparql` del servicio Virtuoso, permitiendo realizar las inserciones de datos a través de peticiones HTTP.

6.4. Dotando de información al sistema

Una vez que la interfaz principal del sistema web y el servicio de Virtuoso están en funcionamiento, se pueden empezar a añadir las diferentes funcionalidades a la aplicación web. Sin embargo, como paso previo a esto, es necesario dotar primero de información sobre los sensores de la máquina a tratar en el sistema, ya que esta información es imprescindible para el funcionamiento de las funcionalidades a desarrollar.

De esta manera, para cualquiera de las dos funcionalidades del sistema (inserción y consulta de datos), cuando el usuario seleccione la máquina extrusora con la que desea trabajar, el sistema recogerá desde Virtuoso la información de los sensores contemplados en la misma. Esta información es la representada en la ontología desarrollada, y de momento, sólo se tiene información sobre la máquina *Extrusora de Cuatro Zonas* de Urola Solutions.

Para la obtención de dicha información, el sistema genera la sentencia SPARQL correspondiente para la consulta de datos. Después, esta sentencia se envía al servicio de SPARQL Endpoint de Virtuoso través de una petición HTTP. Los resultados de la petición contendrán la información que se tiene en la ontología sobre todos los sensores contemplados en la máquina.

En la figura 6.18 se muestra la sentencia SPARQL concreta utilizada para la obtención de dicha información. En la parte del `WHERE` de la sentencia se pueden observar distintas propiedades definidas como `optional`, lo que indica que no todos los sensores pueden tener definida la propiedad en cuestión. Concretamente, estas propiedades son las que varían entre un tipo de sensor y otro.

```
PREFIX sosa: <URI_asociada_al_prefijo_sosa>
...

SELECT ?sensorId ?name ?class ?sensorType ?observationType ?valueType ?zone
       ?observedProperty ?measureUnit ?minValue ?maxValue
FROM <URI_del_grafo_a_consultar>
WHERE {
    ?metaSensorType rdf:type owl:Class ;
                    rdfs:subClassOf sosa:Sensor .
    ?sensorType rdfs:subClassOf ?metaSensorType .
    ?metaSensorType rdfs:subClassOf [ rdf:type owl:Restriction ;
                                      owl:onProperty sosa:madeObservation ;
                                      owl:allValuesFrom ?observationType ] .
    ?observationType rdfs:subClassOf [ rdf:type owl:Restriction ;
                                      owl:onProperty sosa:hasSimpleResult ;
                                      owl:allValuesFrom ?valueType ] .
    ?sensorType rdfs:subClassOf [ rdf:type owl:Restriction ;
                                owl:onProperty sosa:observes ;
                                owl:hasValue ?observedProperty ] .
    optional {
        ?observedProperty qu:unit ?measureUnit .
    }
    optional {
        ?sensorType rdfs:subClassOf [ rdf:type owl:Restriction ;
                                      owl:onProperty :maxValue ;
                                      owl:hasValue ?maxValue ] ,
                                      [ rdf:type owl:Restriction ;
                                      owl:onProperty :minValue ;
                                      owl:hasValue ?minValue ] .
    }
    ?sensorName rdf:type ?sensorType ;
                rdf:type owl:NamedIndividual ;
                :indicatorId ?sensorId ;
                :sensorName ?name .
    optional {
        ?sensorName :zone ?zone .
    }
}
ORDER BY asc(?name)
```

Figura 6.18: Sentencia SPARQL para la obtención de la información necesaria sobre los sensores de la máquina.

Para la realización de todas las peticiones HTTP del sistema web desarrollado se ha hecho uso del paquete *axios*[24], el cual está basado en la característica *promise* de JavaScript y ofrece un cliente HTTP para navegadores y Node.js. Axios ofrece una librería de funciones y un amplio número de opciones posibles para personalizar las peticiones. En este caso, se realizan peticiones *POST* y en la figura 6.19 se muestra la sintaxis utilizada para las mismas.

Una vez recibida la respuesta de la petición, el sistema trata la información obtenida y genera un objeto JSON específico que será utilizado por las funcionalidades para adaptarlas a las características concretas de la máquina seleccionada.

En dicho objeto JSON, está definida la información concreta de cada sensor contemplado, teniendo especificado un valor vacío (" ") en aquellas propiedades que no son aplicables en cada sensor (las definidas como opcionales en la sentencia SPARQL). En la figura 6.20 se muestra parte del objeto JSON en cuestión.

6.5. Dotando de funcionalidad al sistema: Inserción de datos

Una vez que la interfaz principal del sistema web y el servicio de Virtuoso están en funcionamiento, se puede empezar a añadir funcionalidad a la aplicación web. En esta sección se detalla la funcionalidad de inserción de datos, en la que el sistema se encarga de insertar los datos proporcionados por el usuario al grafo creado a través de Virtuoso y que contiene la ontología diseñada.

Para poder realizar esta inserción, los datos deberán pasar por un proceso de transformación y anotación de los mismos en RDF. A continuación explicaremos este proceso de transformación de datos y los pasos que se llevan a cabo para insertar dichos datos mediante peticiones HTTP.

6.5.1. Anotando los datos en RDF

Los datos proporcionados por los usuarios están inicialmente definidos en formato CSV, y éstos siguen un proceso que concluye con la anotación de los mismos en RDF, concretamente, en formato Turtle. En la figura 6.21 se representa gráficamente un resumen de dicho proceso, el cuál se detalla a continuación.

[24]axios: https://github.com/axios/axios

```
axios.post(usedURL, {
    query: query
})
.then((response) => {
    // Handle the response
})
.catch((error) => {
    // Handle the error
});
```

Figura 6.19: Ejemplo de utilización de Axios para la realización de una petición POST.


```
[
    {
        "indicatorId":"2F1KT7",
        "name":"Temperatura Zona 1",
        "class":"temp1",
        "sensorType":"TemperatureSensor",
        "observationType":"TemperatureObservation",
        "resultType":"DoubleValueResult",
        "zone": "1",
        "observedProperty":"temperature",
        "measureUnit":"ºC",
        "minValue":0,
        "maxValue":250
    },
    ...
]
```


Figura 6.20: Estructura del objeto JSON con información de los diferentes sensores de la máquina.

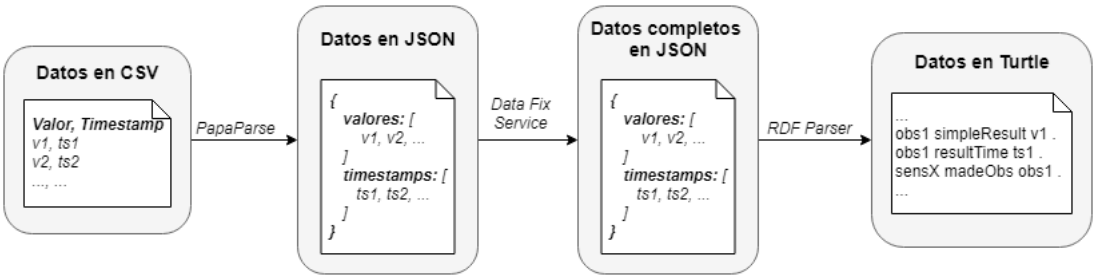

Figura 6.21: Proceso de transformación llevado a cabo en los datos proporcionados.

Antes de seguir con la explicación, es importante aclarar el uso del módulo de *corrección de timestamps* desarrollado por el grupo de investigación BDI, el cual está proporcionado en el servicio *Data Fix Service*. La utilización de este módulo se debe a que, en los archivos de datos definidos en CSV que posee el usuario, hay ciertos momentos de tiempo que no están representados en el mismo (ya sea porque el sensor ha estado apagado, estropeado, etc.), lo que provoca que existan amplios saltos de tiempo entre ciertos valores contiguos. Estos saltos pueden afectar al futuro procesado de los datos en las consultas, por lo que antes de insertar los datos en Virtuoso se ha decidido realizar un pequeño preprocesado para la corrección de los mismos, utilizando para ello el servicio comentado.

Este servicio recibe como entrada los valores y marcas de tiempo iniciales, y realiza la corrección de dichos datos insertando las marcas de tiempo necesarias en los saltos de tiempo encontrados, especificando como 'NA' el valor asociado a dichas marcas de tiempo insertadas. Para esto, el servicio se ajusta al periodo de un valor por segundo, de manera que la serie corregida resultante tendrá una marca de tiempo y valor especificados por cada segundo.

Para la utilización de dicha función del servicio *Data Fix Service*, tanto la entrada como la salida de la misma están definidas como un objeto JSON, por lo que es necesario que los datos a insertar pasen por estar definidos en formato JSON antes de su anotación en modelo RDF.

Por ello, la primera transformación realizada en los datos consiste en el paso del formato CSV proporcionado por el usuario a un objeto JSON. Para esta transformación, se decide el uso de la librería *PapaParse*[25]. PapaParse es una librería que permite la transformación de datos representados en CSV a formato JSON y viceversa dentro del propio navegador, y ofrece gran número de funciones y opciones específicas para adaptarse a las necesidades concretas de cada proceso. En el caso de este TFG, es especialmente interesante la

[25]PapaParse: https://www.papaparse.com/

posibilidad de tratar directamente los datos CSV definidos en un objeto de tipo *File* de JavaScript y la característica de definir la transformación paso a paso de los mismos para evitar el bloqueo del navegador.

Mediante PapaParse y una función definida para obtener el resultado deseado, se consigue entonces la definición de los datos en un objeto JSON que contendrá, por una parte, un *array* con los valores de los sensores, y por otra parte, un *array* con las marcas de tiempo asociadas a cada uno, de manera que ambos *arrays* estén asociados mediante la posición que ocupan los datos en los mismos. (Fig. 6.22)

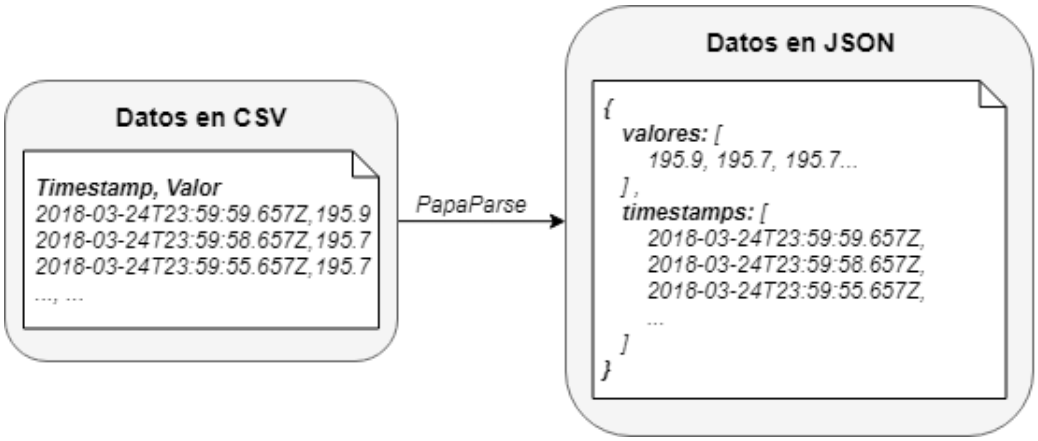

Figura 6.22: Transformación de los datos contenidos en CSV a un objeto JSON.

Una vez que los datos están representados en este objeto JSON, se hace uso del servicio *Data Fix Service*, el cual, como se ha comentado en esta misma sección, devuelve otro objeto JSON con la corrección de los datos realizada. Este objeto JSON tiene además el mismo formato que el objeto JSON de entrada. (Fig. 6.23)

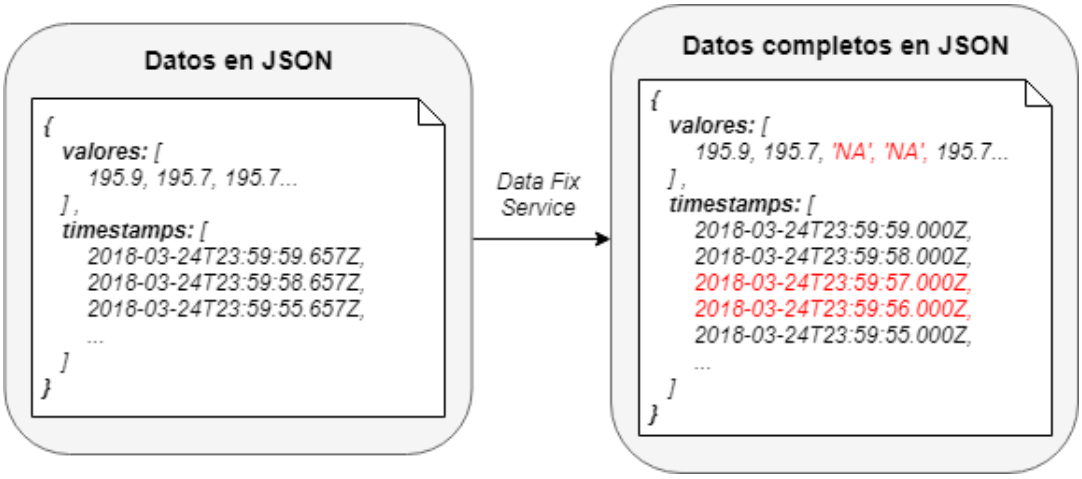

Figura 6.23: Corrección de los datos iniciales utilizando el servicio *Data Fix Service*.

Con el uso del servicio de corrección de datos, éstos ya están listos para su anotación en RDF, ya que serán los datos definitivos a insertar en Virtuoso. Para esto, se ha desarrollado un módulo que realiza dicha anotación de datos.

Este módulo, por cada valor y marca de tiempo asociada, genera cierto número de tripletas RDF que definen una observación compuesta por dicho valor y marca de tiempo. Además, en estas tripletas también se añade información del tipo de valor que se trata y del sensor al que se asocia la observación. Esta información adicional, la cual no está presente en los ficheros CSV, se define a partir de la información definida en la ontología desarrollada, utilizando el identificador del sensor al que pertenecen los datos en cuestión. Este identificador está representado en el nombre del fichero CSV inicialmente subido por el usuario, por lo que el sistema sabe en todo momento a qué sensor pertenecen los datos a insertar.

Por otro lado, tal y como está definido en la ontología, los valores de los sensores deben ser todos de tipo *xsd:boolean* o todos de tipo *xsd:double*, dependiendo del tipo de sensor al que pertenezcan los mismos. Sin embargo, a partir de la corrección realizada mediante el servicio *Data Fix Service*, en el objeto JSON devuelto puede aparecer el valor 'NA' representando la falta de valores en ciertos momentos. Este valor no corresponde a ninguno de los tipos de datos mencionados, por lo que cuando en el objeto JSON aparezca el valor 'NA', el módulo lo transformará en 'NaN' en caso de datos de tipo *xsd:double* y en 'xsi:nil' en caso de datos de tipo *xsd:boolean*.

En la figura 6.24 se muestra un ejemplo de dicha anotación de los datos en formato Turtle, suponiendo que los datos anotados pertenecen al sensor con identificador *X* y cuyos datos son de tipo *double*.

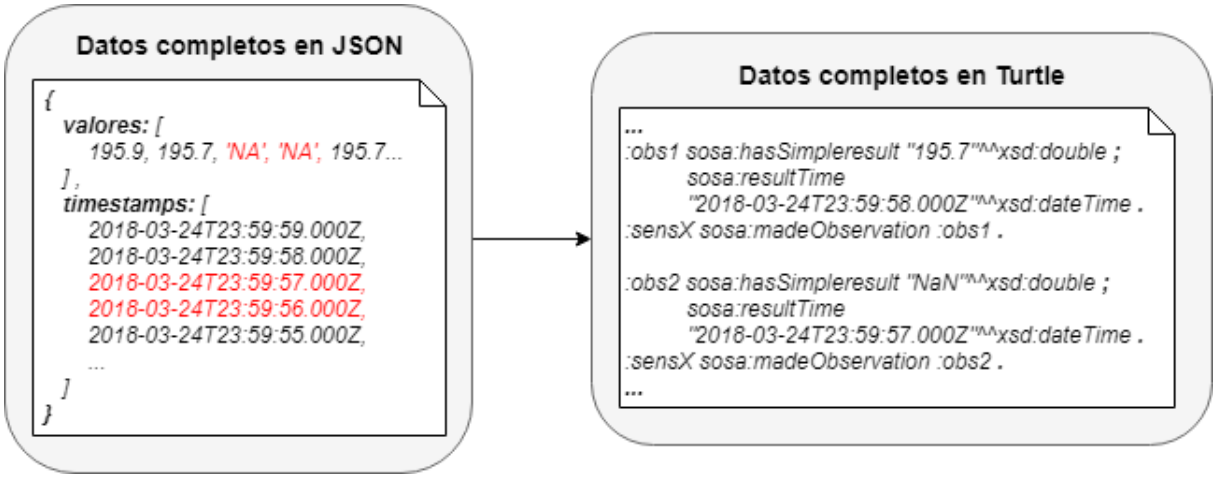

Figura 6.24: Anotación de los datos completos en JSON en formato Turtle.

6.5.2. Insertando datos a Virtuoso desde la aplicación web

Una vez explicadas las diferentes transformaciones realizadas sobre los datos proporcionados por el usuario, a continuación se detalla el proceso de inserción de dichos datos en Virtuoso. Este proceso de inserción se realiza mediante peticiones HTTP y se hace en conjunto con el proceso de anotación RDF.

Para empezar, para poder realizar la inserción en Virtuoso, es necesario formular la sentencia SPARQL adecuada, en la cual se debe especificar la *URI* del grafo destino de la inserción y los propios datos a insertar.

En la figura 6.25 se representa la estructura principal de una sentencia *Insert* utilizando SPARQL 1.1, de manera que los datos a insertar (*datos_a_insertar* en la figura) están definidos en formato Turtle. Dicha estructura es la utilizada en el módulo de datos encargado de la inserción de datos.

```
PREFIX p: <URI_asociada_al_prefijo_p>
...

INSERT DATA
{
   GRAPH <URI_del_grafo_destino>
   {
      ... datos_a_insertar ...
   }
}
```

Figura 6.25: Estructura de una consulta de inserción de datos en SPARQL 1.1.

Por otro lado, Virtuoso tiene un límite por defecto en cuanto al tamaño de información que recibe por peticiones HTTP. El tamaño resultante de una consulta SPARQL generada con los datos a insertar a partir de un fichero CSV habitual supera por mucho dicho límite. Como bien se explica más adelante (en la sección 8.2), dicho límite no ha podido ser modificado, por lo que la información a enviar se trocea de manera que las consultas que se generan no lo superen.

Por esto, una vez que los datos han sido corregidos y se tiene el objeto JSON con la información definitiva a insertar, el sistema lee cierto número definido de valores y marcas de tiempo asociadas y las anota en RDF. Después, se genera la consulta SPARQL asociada a dichos datos y se envía la misma mediante una petición HTTP al servicio de SPARQL Endpoint de Virtuoso (ruta `/sparql` de Virtuoso). Cuando dicha petición ha sido aceptada por Virtuoso, se pasa al siguiente conjunto de valores y marcas de tiempo, realizando el mismo proceso hasta que se hayan insertado todos los datos contenidos en el objeto JSON.

En un principio, este proceso no se realizaba de forma síncrona, de manera que el sistema no esperaba a recibir la aceptación de Virtuoso para continuar con la siguiente porción de datos a insertar. Esto sin embargo resultaba en una sobrecarga de Virtuoso o del cliente al ejecutar tantas peticiones al mismo tiempo. Por esto, a pesar del tiempo elevado necesario para ejecutar todas las peticiones de manera síncrona, se ha decidido optar por dicha opción para poder asegurar la correcta inserción de todos los datos.

Por último, a medida que los datos van siendo anotados en RDF, esta información se guarda en una variable para después generar un fichero con extensión *.ttl* con dicha información. De esta manera, se permite al usuario la descarga de este fichero a través del navegador, el cual contendrá los mismos datos que los especificados en el fichero CSV proporcionado al principio pero definidos en formato Turtle y con la corrección de los *timestamps* ya realizada.

6.6. Dotando de funcionalidad al sistema: Consulta de datos

La otra funcionalidad principal que el sistema ofrece, es la consulta de la información que ha sido insertada en Virtuoso mediante el proceso explicado en la sección anterior.

En esta sección se detalla el desarrollo de dicha funcionalidad, explicando los diferentes procesos llevados a cabo para conseguir el objetivo final: la visualización de una gráfica con los resultados de la consulta realizada.

6.6.1. Generando sentencias SPARQL

Para poder realizar consultas sobre los datos contenidos en Virtuoso, los cuales están representados en RDF, será necesario crear distintas sentencias SPARQL que permitan preguntar por los datos deseados. Para ello, se han desarrollado diferentes funciones en las que a partir de cierta información proporcionada se genera una sentencia SPARQL. Antes de continuar, es necesario puntualizar que las consultas generadas por estas funciones siguen las especificaciones de SPARQL 1.1.

Concretamente, se han desarrollado tres funciones distintas para la obtención de tres tipos de sentencias SPARQL:

1. Para la consulta de información sobre un sensor de manera aislada.
2. Para la consulta de información sobre un sensor definiendo una relación del mismo con otro u otros sensores.
3. Para la búsqueda de anomalías en los datos de ciertos sensores.

Sentencias SPARQL para consultar información sobre un sensor de manera aislada.

Con este tipo de consultas, se permite preguntar por toda la información disponible sobre un sensor concreto, filtrando dicha información por ciertos campos como fechas, horas, mediante funciones agregadas, etc.

Para la generación de las sentencias, la función desarrollada tiene como entrada un objeto JSON que contiene información sobre:

- El identificador del sensor a consultar
- Un valor *booleano* que represente si se desea realizar agrupación de datos o no. En caso afirmativo, también se especifican el campo utilizar para la agrupación y las funciones agregadas a realizar en las mismas.
- Un valor *booleano* que represente si se desean realizar filtros o no. En caso afirmativo, también se especifican el tipo de filtros a realizar y los valores, fechas, horas, etc. entre los que filtrar.
- Un valor *booleano* que represente si se desea ordenar los resultados o no. En caso afirmativo, también se especifican el campo a tener en cuenta en la ordenación y el sentido de la misma (ascendente o descendente).

En la figura 6.26 se muestra la estructura general de una sentencia `SELECT` con ordenación y agrupaciones en SPARQL 1.1, suponiendo que el sensor del que se preguntan los datos tiene el identificador *sensorX*. A continuación se explican las diferentes características de dicha sentencia presentada en la figura.

```
BASE <URI_del_grafo_base>
PREFIX sosa: <URI_grafo_prefijo_sosa>
... mas_prefijos ...

SELECT campos_a_consultar
FROM <URI_grafo_a_consultar>
WHERE {
    ?sensorName sosa:madeObservation ?obsName .
    ?obsName sosa:hasSimpleResult ?resultValue .
    ?obsName sosa:resultTime ?resultTime .
    filter( ?sensorName = <#sensorX> ) .
    ...
}
GROUP BY campos_a_agrupar
ORDER BY desc/asc(campo_a_ordenar)
```

Figura 6.26: Estructura principal de una sentencia `SELECT` con agrupaciones y ordenaciones.

La parte del `WHERE` detallada en la figura muestra la estructura básica que tienen las consultas de información general sobre un sensor, de manera que en las variables `?resultValue` y `?resultTime` obtenemos el valor y el timestamp de cada observación hecha por el sensor deseado respectivamente. Para asegurarnos de que estos datos pertenecen al sensor adecuado, mediante la función `filter()` filtramos los resultados de manera que el sensor que haya hecho la observación (representado por la variable `?sensorName`) concuerde con el identificador del sensor deseado, el cual es *sensorX* en este caso.

A parte de estas tripletas básicas representadas en la parte del `WHERE` de la figura, las cuales siempre aparecerán en las sentencias de información general, dependiendo de las características de cada sentencia, también aparecerán más funciones para realizar distintas operaciones con los datos, ya sean `filter()`, `bind()`, funciones de tratamiento de strings, etc.

Para el uso de funciones agregadas, a partir del `GROUP BY` agrupamos los resultados por el campo que deseemos, y después ejecutamos las agrupaciones en la parte del `SELECT`.

Las funciones agregadas que se utilizan en esta aplicación concreta son:

- `MAX(`*campo_a_consultar*`)`: devuelve el máximo del campo especificado.
- `MIN(`*campo_a_consultar*`)`: devuelve el mínimo del campo especificado.
- `AVG(`*campo_a_consultar*`)`: devuelve la media aritmética del campo especificado.

En las figuras 6.27 y 6.28 se muestran dos ejemplos de sentencias SPARQL completas utilizando las diferentes características explicadas anteriormente.

```
BASE <URI_del_grafo_base>
PREFIX sosa: <URI_grafo_prefijo_sosa>
PREFIX xsd: <URI_grafo_prefijo_xsd>

SELECT ?resultValue ?resultTime
FROM <URI_grafo_a_consultar>
WHERE {
    ?sensorName sosa:madeObservation ?obsName .
    ?obsName sosa:hasSimpleResult ?resultValue .
    ?obsName sosa:resultTime ?resultTime .
    filter(
        ?resultValue >= valor_1^^xsd:double &&
        ?resultValue <= valor_2^^xsd:double &&
        ?sensorName = <#sensorX>
    )
    filter(
        (xsd:time(xsd:dateTime(?resultTime)) >= hora_inicio^^xsd:time) &&
        (xsd:time(xsd:dateTime(?resultTime)) <= hora_final^^xsd:time)
    )
}
ORDER BY asc(?resultTime)
```

Figura 6.27: Ejemplo de sentencia `SELECT` con ordenaciones y filtros en los valores y las horas.

```
BASE <URI_del_grafo_base>
PREFIX sosa: <URI_grafo_prefijo_sosa>
PREFIX xsd: <URI_grafo_prefijo_xsd>

SELECT ?sensorName ?resultHour (AVG(?resultValue) as ?avgValue)
FROM <URI_grafo_a_consultar>
WHERE {
    ?sensorName sosa:madeObservation ?obsName .
    ?obsName sosa:hasSimpleResult ?resultValue .
    ?obsName sosa:resultTime ?resultTime .
    filter( ?sensorName = <#sensorX> ) .
    bind(xsd:time(xsd:dateTime(?resultTime)) as ?time) .
    bind(substr(str(?time), 1, 2) as ?hour) .
    bind(concat(?hour, ":00:00") as ?resultHour) .
}
GROUP BY ?sensorName ?resultHour
ORDER BY asc(?resultHour) desc(?sensorName)
```

Figura 6.28: Ejemplo de sentencia `SELECT` con ordenaciones, agrupaciones y utilizando funciones agregadas.

Sentencias SPARQL para consultar información de un sensor definiendo su relación con otros.

Con este tipo de consultas, se permite preguntar por los valores de un determinado sensor estableciendo unas condiciones determinadas para otro u otros sensores. De esta manera, obtendremos los valores del sensor preguntado cuando se den las condiciones impuestas para el resto de sensores. Estas consultas además, se podrán filtrar entre determinadas fechas.

Este tipo de sentencias son utilizadas en las consultas de relación entre sensores, de manera que se especifica dicha relación directamente en la sentencia SPAQRL.

Para la generación de las sentencias, la función desarrollada tiene como entrada un objeto JSON que contiene información sobre:

- El identificador del sensor cuyos valores se desean consultar.
- Los identificadores de los sensores de los que se fijarán los valores, junto con dichos valores específicos o la función agregada para conseguirlos (”min” o ”max”).
- Un valor *booleano* que represente si se desean realizar filtros. En caso afirmativo, se definen también el tipo de filtros a realizar y los valores y/o fechas entre los que filtrar.

Para fijar el valor del sensor o sensores que crean la relación con el sensor cuyos datos se preguntan, como bien se ha mencionado, se puede especificar el valor concreto o también se puede especificar una función agregada para el cálculo de dicho valor.

En la figura 6.29 se muestra un ejemplo de una sentencia en la que se proporciona el valor concreto del sensor con valor fijo. En dicha figura, se consulta el valor del sensor **sensorA** cuando el sensor **sensorK1** tenga el valor **valor_K1**. Para poder especificar la relación entre ambas observaciones, se especifica que la marca de tiempo de ambas observaciones debe ser la misma (`?resultTime` en ambos casos).

Para el caso de valores calculados, en la figura 6.30 se muestra un ejemplo de una sentencia en la que no se proporciona el valor concreto a fijar, si no que se calcula mediante una función agregada (la cuál puede ser MIN, MAX o AVG). En dicha ficgura, se consulta el valor del sensor **sensorA** cuando el sensor **sensorK1** tenga el valor **?calculatedValue1**. Para el cálculo de este valor, se hace una sentencia `SELECT` anidada en la que se calcula el valor con la fucnión agregada deseada y se almacena en la variable `?calculatedValue1`.

```
BASE <URI_del_grafo_base>
PREFIX sosa: <URI_grafo_prefijo_sosa>
... mas_prefijos ...

SELECT ?sensorName ?resultValue ?resultTime
FROM <URI_grafo_a_consultar>
WHERE {

    <#sensorK1> sosa:madeObservation ?knownObs1 .
    ?knownObs1 sosa:hasSimpleResult valor_K1^^xsd:double .
    ?knownObs1 sosa:resultTime ?resultTime .

    ...

    <#sensorA> sosa:madeObservation ?askedObs .
    ?askedObs sosa:hasSimpleResult ?resultValue .
    ?askedObs sosa:resultTime ?resultTime .
    bind(<#sensorA> as ?sensorName)
}
ORDER BY desc/asc(campo_a_ordenar)
```

Figura 6.29: Ejemplo de sentencia `SELECT` para la relación entre sensores mediante un valor concreto.

```
BASE <URI_del_grafo_base>
PREFIX sosa: <URI_grafo_prefijo_sosa>
... mas_prefijos ...

SELECT ?sensorName ?resultValue ?resultTime
FROM <URI_grafo_a_consultar>
WHERE {
    {
        SELECT (MIN/MAX/AVG(?calResultValue) as ?calculatedValue1)
        WHERE {
            <#sensorK1> sosa:madeObservation ?calObs1 .
            ?calObs1 sosa:hasSimpleResult ?calResultValue .
        }
    }
    <#sensorK1> sosa:madeObservation ?knownObs1 .
    ?knownObs1 sosa:hasSimpleResult ?calculatedValue1 .
    ?knownObs1 sosa:resultTime ?resultTime .

    ...

    <#sensorA> sosa:madeObservation ?askedObs .
    ?askedObs sosa:hasSimpleResult ?resultValue .
    ?askedObs sosa:resultTime ?resultTime .
    bind(<#sensorA> as ?sensorName)
}
ORDER BY desc/asc(campo_a_ordenar)
```

Figura 6.30: Ejemplo de sentencia `SELECT` para la relación entre sensores mediante un valor calculado.

Al igual que en el caso anterior, la relación entre ambas observaciones se realiza mediante la variable `?resultTime`.

De esta manera, por cada sensor con valor fijado se irá añadiendo una de las dos formas de especificar su valor dentro de la parte `WHERE` de la sentencia, relacionando finalmente todas las observaciones de los distintos sensores mediante la marca de tiempo, asegurando así que pertenezcan todos los valores al mismo momento de tiempo.

En la figura 6.31 se muestra un ejemplo uniendo ambas formas de fijar el valor de los sensores de valores conocidos, de manera que se consulta el valor del sensor **sensorA** cuando el sensor **sensorK1** tenga el valor **valor_k1** y cuando el sensor **sensorK2** tenga el valor **?calculatedValue1**.

Sentencias SPARQL para la búsqueda de anomalías en los datos de ciertos sensores.

Un cierto tipo de consultas que se permiten realizar en la aplicación web es la búsqueda de anomalías en los datos de los sensores. Para ello, el usuario proporciona una relación entre ciertos sensores que debería cumplirse en condiciones normales. Cuando esta relación no se cumple, se considera que ha habido una anomalía.

Dichas relaciones para la búsqueda de anomalías definen la tendencia que deberían tener los valores de los sensores entre si, es decir, son relaciones del tipo ”cuando los valores del sensor *X* aumenten, los valores del sensor *Y* deberían aumentar también, etc.”. Por ello, la información a mostrar al usuario es únicamente sobre aquellos momentos en los que la relación no se ha cumplido.

Este tipo de consultas sin embargo no se han podido realizar directamente en SPARQL, ya que implican la comparación entre datos contiguos en el tiempo para poder seleccionar únicamente aquellos momentos y datos que incumplen la relación definida.

Por ello, para la búsqueda de anomalías, las sentencias SPARQL utilizadas corresponden con las sentencias de consulta de información general sobre cada de sensor. De esta manera, se consulta toda la información disponible sobre los sensores que forman parte de la relación de anomalía definida, y una vez tenida toda la información, se utiliza una función que se encarga de la búsqueda de anomalías en la misma.

Esta función de búsqueda de anomalías se detalla en el siguiente apartado, ya que entra dentro del proceso de consulta de datos específico del tipo de preguntas en cuestión.

```
BASE <URI_del_grafo_base>
PREFIX sosa: <URI_grafo_prefijo_sosa>
... mas_prefijos ...

SELECT ?sensorName ?resultValue ?resultTime
FROM <URI_grafo_a_consultar>
WHERE {

    <#sensorK1> sosa:madeObservation ?knownObs1 .
    ?knownObs1 sosa:hasSimpleResult valor_K1^^xsd:double .
    ?knownObs1 sosa:resultTime ?resultTime .

    {
        SELECT (MIN/MAX/AVG(?calResultValue) as ?calculatedValue1)
        WHERE {
            <#sensorK2> sosa:madeObservation ?calObs1 .
            ?calObs1 sosa:hasSimpleResult ?calResultValue .
        }
    }
    <#sensorK2> sosa:madeObservation ?knownObs2 .
    ?knownObs2 sosa:hasSimpleResult ?calculatedValue1 .
    ?knownObs2 sosa:resultTime ?resultTime .

    ...

    <#sensorA> sosa:madeObservation ?askedObs .
    ?askedObs sosa:hasSimpleResult ?resultValue .
    ?askedObs sosa:resultTime ?resultTime .
    bind(<#sensorA> as ?sensorName)
}
ORDER BY desc/asc(campo_a_ordenar)
```

Figura 6.31: Ejemplo de sentencia `SELECT` para la relación entre sensores mediante un valor concreto y otro calculado.

6.6.2. Consultando datos desde la aplicación web

Utilizando las sentencias SPARQL mencionadas en la sección anterior, podemos consultar los datos deseados desde la aplicación web mediante peticiones HTTP al SPARQL Endpoint de Virtuoso. Después, en la respuesta de estas peticiones se obtienen los resultados de la consulta realizada, datos que a continuación mostraremos visualmente mediante una gráfica.

Para este proceso, sin importar el tipo de consulta a realizar, lo primero es recoger la información introducida por el usuario en el formulario correspondiente y representarla adecuadamente en un objeto JSON para utilizar las funciones de creación de sentencias SPARQL comentadas.

Una vez que dicho objeto JSON está correctamente creado, por cada sensor a consultar se ejecutan una serie de pasos:

1. Se obtiene la sentencia SPARQL correspondiente con ayuda del objeto JSON creado y el identificador del sensor correspondiente.
2. Se envía dicha sentencia mediante una petición HTTP al servicio SPARQL Endpoint de Virtuoso.
3. Una vez que se ha recibido la respuesta con los resultados, éstos se reestructuran y se les aplica una reducción al mismo tiempo.
    - En el caso concreto de las consultas de búsqueda de anomalías, no se aplica la reducción de datos comentada.

Este proceso se repite tantas veces como sensores haya para consultar, y los resultados ya reestructurados y reducidos se van guardando en una variable de resultados finales. Cuando ya no hay más sensores por los que consultar, los resultados finales están listos para ser analizados y/o mostrados mediante gráficas.

La reestructuración que se hace sobre los datos viene de una necesidad de facilitar el manejo de los mismos para los distintos procesos posteriores. El formato de respuesta ofrecido por Virtuoso, por tratarse de datos en formato RDF, contiene más información de la necesaria para la funcionalidad concreta especificada. Por ello, mediante una función creada y con ayuda de la información de la consulta realizada previamente almacenada en un objeto JSON, se reestructuran los resultados de manera que solo se tienen en cuenta los valores de los sensores y sus marcas de tiempo asociadas.

```
[
    {
        "resultTime":{
            "datatype": "http://www.w3.org/2001/XMLSchema#dateTime",
            "type": "typed-literal",
            "value": "2018-03-24T12:00:00Z",
        },
        "resultValue":{
            "datatype": "http://www.w3.org/2001/XMLSchema#double",
            "type": "typed-literal",
            "value": "195.5",
        },
        ...
    },
    {
        "resultTime":{
            "datatype": "http://www.w3.org/2001/XMLSchema#dateTime",
            "type": "typed-literal",
            "value": "2018-03-24T12:00:01Z",
        },
        "resultValue":{
            "datatype": "http://www.w3.org/2001/XMLSchema#double",
            "type": "typed-literal",
            "value": "195.4",
        },
        ...
    },
    ...
]
```


Figura 6.32: Ejemplo de la estructura de los resultados recibidos en la petición HTTP a Virtuoso.

En la figura 6.32 se muestra un ejemplo de la estructura de los resultados recibidos en la respuesta de Virtuoso. Ésta está compuesta por un *array* de objetos. Cada uno de estos objetos corresponde a una observación, y están compuestos por un objeto por cada variable especificada en el `SELECT` de la sentencia consultada. Por último, por cada variable especificada en el `SELECT`, se especifican distintos valores de la misma: el tipo de dato de la misma (si es que está especificado), el tipo de la misma y, por último, el valor que contiene la variable.

Por ello, estos resultados se reestructuran mediante una función, a partir de la cual se obtiene un nuevo objeto JSON con los mismos resultados pero obviando la información que no es necesaria para la funcionalidad de consulta de datos. Esta reestructuración varía

ligeramente entre los casos en los que cada sensor solo tiene un valor por unidad de tiempo como resultado y los casos en los que cada sensor tiene más de un valor por unidad de tiempo como resultado. Esto último ocurre al utilizar más de una función agregada para más de un sensor a la vez, de manera que por cada unidad de tiempo se tiene el valor mínimo, máximo y medio del sensor, por ejemplo.

En la figura 6.33 se muestra un ejemplo de la estructura final que tendrán los resultados de los distintos sensores consultados en el caso de que solo se tenga un valor por unidad de tiempo.

En la figura 6.34 se muestra un ejemplo de la estructura final que tendrán los resultados de los distintos sensores consultados en el caso de que se tenga más de un valor por unidad de tiempo.

Por otro lado, después de la reestructuración de los resultados, en las consultas de información general y relación entre sensores, también se reducen dichos resultados antes de consultar los datos de un nuevo sensor. Esto se debe a dos razones principales:

- El rendimiento del gráfico de Google que se muestra a partir de dichos resultados empieza a bajar notablemente a partir de un cierto límite de datos introducidos en el mismo, resultando en un funcionamiento muy poco fluído y limitando la comprensión del mismo en gran medida. Por ello, para tener un buen equilibro entre la precisión de los datos mostrados y la fluidez del gráfico a manejar, se ha decidido que se introduzcan como máximo 4500 puntos en cada gráfico.

- Para evitar un exceso de datos almacenados en el cliente, esta reducción se hace antes de pasar al siguiente sensor consultado, sin esperar a tener almacenados los resultados de todos los sensores en cuestión. Los resultados de las consultas conllevan generalmente una inmensa cantidad de datos, por lo que se intenta adelantar lo máximo posible la reducción de los mismos para evitar el almacenamiento de tantos datos en el cliente.

Para esta reducción, se ha desarrollado una función que calcula la media aritmética de los valores de un rango determinado. Dicho rango se calcula dinamicamente a partir de la información que se tiene sobre la consulta realizada, de manera que los resultados finales obtenidos de todos los sensores no puedan sobrepasar los 4500 datos a introducir en la gráfica. Para las marcas de tiempo asociadas a dichos valores medios calculados, se tiene en cuenta el punto medio entre la primera y la última marca de tiempo de los valores

```
{
    "2F1KT7" : {
        "datetimes" : ["2018-03-24T12:00:00Z", "2018-03-24T12:00:01Z", "2018-03-24T12:00:02Z"...],
        "values" : [195.5, 195.5, 195.4...],
    },
    "79PWN7" : {
        "datetimes" : ["2018-03-24T12:00:00Z", "2018-03-24T12:00:01Z", "2018-03-24T12:00:02Z"...],
        "values" : [17965.0, 17791.0, 18210.0...],
    },
    ...
}
```


Figura 6.33: Ejemplo del resultado después de la reestructuración, con solo un valor como resultado.


```
{
    "2F1KT7" : {
        "datetimes" : ["12:00-13:00", "13:00-14:00", "14:00-15:00"...],
        "values" : {
            "avgValue": [195.52, 193.79, 195.56...],
            "maxValue": [195.8, 3276.7, 203.9...],
             ...
        }
    },
    "79PWN7" : {
        "datetimes" : ["12:00-13:00", "13:00-14:00", "14:00-15:00"...],
        "values" : {
            "avgValue": [18320.32, 9688.77, 17501.79...],
            "maxValue": [19568, 19752, 21070...],
             ...
        }
    },
    ...
}
```


Figura 6.34: Ejemplo del resultado después de la reestructuración, con más de un valor como resultado.

utilizados para la media. En caso de los valores que no tienen dato (especificado por "*NaN*" o "*xsi:nil*"), éstos no tomarán parte en la realización de la media.

Sin embargo, en el caso de las consultas de anomalías, la consulta que se realiza a Virtuoso es solamente para obtener todos los datos (filtrados entre fechas si el usuario lo especifica) sobre los sensores que toman parte en la relación de anomalía a analizar. Una vez que se obtienen todos los resultados de cada sensor, mediante una función desarrollada, se hace una búsqueda de las anomalías aparecidas en los mismos y se guardan los resultados en una nueva variable. Por esto, la reducción de los datos en este tipo de consultas no se puede realizar hasta no poseer los resultados finales de las posibles anomalías ocurridas.

Por último, dicha búsqueda de anomalías se realiza a través de una función desarrollada. Esta función recorre en paralelo la información obtenida sobre cada sensor, guardando siempre el valor anterior (si existe) al valor observado en cada momento. Los resultados están ordenados cronológicamente, por lo que de esta manera se van comparando los valores inmediatamente contiguos de cada sensor para así poder observar su tendencia. Cuando la tendencia de cada sensor ha sido determinada para un cierto momento, se comparan dichas tendencias con la relación de anomalía especificada, teniendo tres tipos de resultados posibles:

- Todos los sensores han tenido la tendencia indicada en la relación de anomalía. Esto indica que no ha habido ninguna anomalía, por lo que se descartan los datos actuales y se sigue al siguiente valor de cada sensor.

- Sólo algunos de los sensores han tenido la tendencia indicada en la relación. Esto indica que se ha encontrado una anomalía, por lo que se guardan los datos actuales antes de continuar con el siguiente valor de cada sensor.

- Ninguno de los sensores han tenido la tendencia indicada en la relación de anomalía. Esto indica que no ha habido ninguna anomalía, por lo que se descartan los datos actuales y se sigue al siguiente valor de cada sensor.

Cuando todos los datos de los sensores han sido analizados, se aplica finalmente la reducción de datos a los resultados de anomalías obtenidos utilizando para ello la misma función comentada anteriormente.

6.6.3. Generando gráficas

Una vez que se han recibido y procesado los resultados de todos los sensores consultados, los datos pueden ser mostrados en la o las gráficas correspondientes.

Como bien se ha mencionado anteriormente, para la generación de las gráficas se ha decidido utilizar las *gráficas de Google*[26]. Esta herramienta proporciona una manera sencilla e intuitiva de crear gráficas interactivas y estilizadas.

Por otro lado, con el fin de facilitar la integración de dichas gráficas de Google con el desarrollo en React de la aplicación, se ha decidido utilizar la librería *React Google Charts*[27]. React Google Charts ofrece una manera sencilla de integrar las gráficas de Google con React, permitiendo utilizar todas las características y opciones asociadas a las mismas.

Con dicha librería, las gráficas de Google se crean haciendo referencia al componente de React `Chart`, introduciendo diferentes características mediante propiedades de dicho componente. (Fig. 6.35)

```
<Chart
     chartType={chartType}
     data={data}
     options={finalOptions}
     width="100%"
     height="400px"
     chartEvents={chartEvents}
 />
```

Figura 6.35: Ejemplo de inicialización de una gráfica de Google con el componente `Chart` ofrecido por React Google Charts.

Para introducir los datos en las gráficas de Google, éstos tiene que tener un formato específico. Este formato consiste en un *array* de *arrays* los cuales forman una especie de tabla, de manera que cada uno de los *arrays* corresponde con cada fila de la tabla y las posiciones en los mismos indican la columna indicada. En la figura 6.36 se muestra un ejemplo de esta estructura de datos comentada, suponiendo que para cada momento de tiempo indicamos un valor de cada sensor consultado.

[26] Gráficas de Google: https://developers.google.com/chart/
[27] React Google Charts: https://github.com/rakannimer/react-google-charts

```
[
    ["Fecha y hora", "Temperatura Zona 1", "Consumo Motor" ...],
    ["2018-03-24T12:00:00Z", 195.5, 17969.0 ...],
    ["2018-03-24T12:00:01Z", 195.5, 17791.0 ...],
    ["2018-03-24T12:00:02Z", 195.4, 18210.0 ...],
    ["2018-03-24T12:00:03Z", 195.5, 18219.7 ...],
    ["2018-03-24T12:00:04Z", 195.6, 18320.95 ...],
    ...
]
```

Figura 6.36: Estructura de datos a introducir en las gráficas de Google.

A partir de los datos especificados siguiendo la estructura de la figura 6.36, obtenemos una gráfica del estilo de la mostrada en la figura 6.37, de manera que en una misma gráfica podemos observar los diferentes valores que tienen los sensores consultados para cada momento de tiempo.

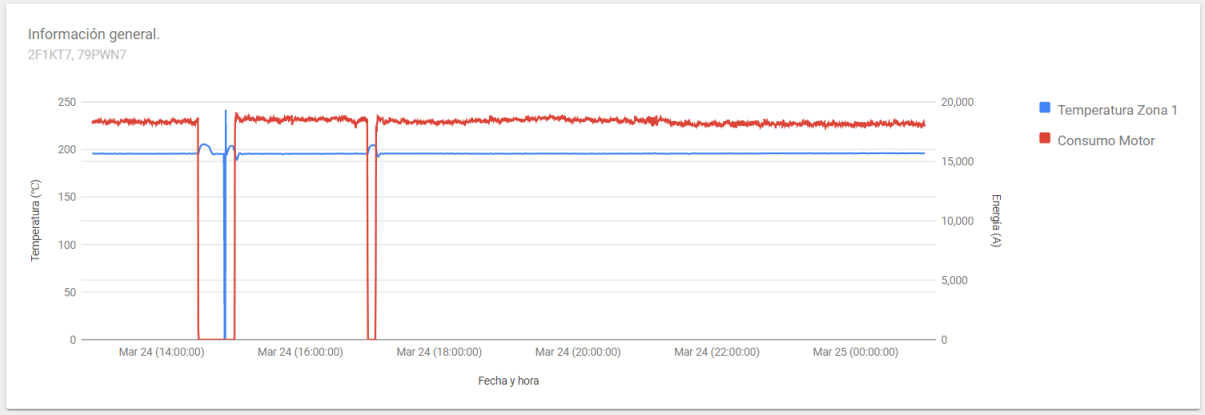

Figura 6.37: Gráfica resultante con un valor por unidad de tiempo en cada sensor.

En el caso de tener más de un valor a mostrar por cada sensor, como bien se ha explicado en el apartado de Diseño de la memoria (sección 5.2), se ha decidido separar dichos datos de cada sensor en diferentes gráficas, ya que de lo contrario, el gráfico mostrado resultaría ser menos intuitivo y entendible. En la figura 6.38 se muestra un ejemplo de esta separación de datos para diferentes gráficas, de manera que la gráfica destinada a cada sensor utilizaría el *array* de datos asociado al identificador de dicho sensor.

```
{
    "2F1KT7": [
        ["Hora", "Valor medio", "Valor máximo" ...],
        ["12:00-13:00", 195.52, 195.8 ...],
        ["13:00-14:00", 193.79, 3276.7 ...],
        ["14:00-15:00", 195.56, 203.9 ...],
        ...
    ],
    "79PWN7": [
        ["Hora", "Valor medio", "Valor máximo"...],
        ["12:00-13:00", 18320.32, 19568 ...],
        ["13:00-14:00", 9688.77, 19752 ...],
        ["14:00-15:00", 17501.79, 21070 ...],
        ...
    ],
    ...
}
```

Figura 6.38: Estructura de datos a introducir en las gráficas de Google, especificando los datos de cada sensor como distintas gráficas.

De esta manera, se obtendría una gráfica distinta para cada sensor, en la que se mostrarían los diferentes valores del mismo para cada unidad de tiempo. Un ejemplo de esto se muestra en la figura 6.39.

Por último, dependiendo de las características de las consultas y de los resultados mostrados, se personalizarán el tipo de gráficas mostradas y las opciones de las mismas para maximizar la comprensión de las mismas.

El tipo de concreto de cada gráfica (lineal, de barras, etc.) se define en la propiedad `type` del componente React `Chart`. En la documentación oficial de las gráficas de Google se puede encontrar una lista de todas los tipos de gráficas disponibles y el nombre que las identifica.

Las opciones que pueden ser personalizadas en cada gráfica sin embargo, se definen mediante un objeto JSON y se deben especificar en la propiedad `options` del componente React `Chart`. En la documentación oficial de las gráficas de Google se puede encontrar una lista de todas las opciones de personalización disponibles para cada tipo de gráfica. En la figura 6.40 se muestra un ejemplo de un objeto JSON que contiene dichas opciones, siendo éstas concretamente las opciones que dan resultado a la gráfica mostrada en la figura 6.37.

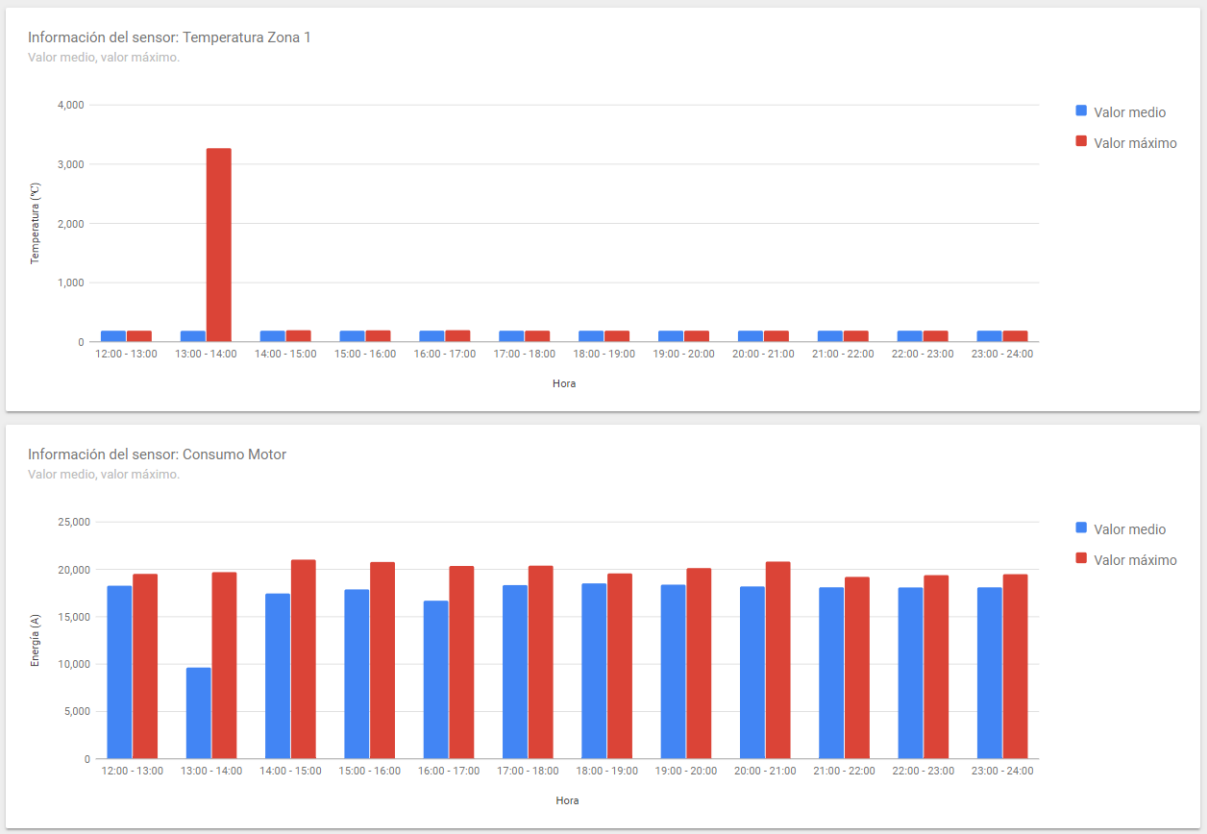

Figura 6.39: Gráficas resultantes con más de un valor por unidad de tiempo en cada sensor.

```
{
    "axes": {
        "y": {
            "temperature": {"label": "Temperatura (ºC)"},
            "power": {"label": "Energía (A)"}
        }
    },
    "chart": {
        "subtitle": "2F1KT7, 79PWN7",
        "title": "Información general"
    },
    "hAxis": {
        "format": "MMM dd (HH:mm:ss)"
    },
    "series": {
        "0": {"axis": "temperature"},
        "1": {"axis": "power"}
    },
    "vAxis": {
        "format": "decimal"
    }
}
```


Figura 6.40: Objeto JSON que representa las opciones utilizadas para la gráfica mostrada en la figura 6.37.

6.7. La fusión con I4TSPS

Por último, una vez que el sistema web desarrollado consta de todas las funcionalidades requeridas, para poder realizar la unión del mismo con la aplicación I4TSPS comentada es necesario realizar unos últimos cambios en el mismo.

A continuación se detallan los pasos dados para la preparación de dicha fusión.

6.7.1. Instalando Virtuoso en una máquina virtual Debian

Antes de pasar a realizar la unión de ambas aplicaciones, es necesario alojar una instancia de Virtuoso en la web, ya que si no no podremos utilizar los servicios que ofrece el mismo desde la aplicación I4TSPS.

La aplicación I4TSPS actualmente ya utiliza ciertos servicios instalados en una máquina virtual con sistema operativo Debian, por lo que se decide realizar una instalación de Virtuoso en una nueva máquina virtual de las mismas características que la mencionada,

para así comprobar el correcto funcionamiento de Virtuoso en dichas condiciones y evitar posibles errores que pudieran ocurrir durante la instalación.

Por lo tanto, se crea una nueva máquina virtual en la plataforma *Google Cloud*[28]. Google Cloud es una plataforma que reúne en un mismo sitio todas las aplicaciones de desarrollo web ofrecidas por Google. En este caso, se hará uso de los recursos informáticos que ofrece Google, concretamente de una máquina virtual. Estos servicios no son gratuitos, pero como se trata de una instalación de prueba, se utilizará el crédito de 300$ que ofrece Google al utilizar la plataforma Google Cloud por primera vez. De esta manera, una vez que este TFG haya finalizado, la instalación de Virtuoso podrá traspasarse a la máquina virtual definitiva utilizada por la aplicación I4TSPS.

Una vez creada la máquina virtual, mediante Google Cloud Platform podremos iniciarla, cambiar sus propiedades, detenerla, etc. (Fig. 6.41) Es importante saber que cada vez que se incie la máquina virtual, ésta tendrá una IP distinta. Para cambiar esto, se debe adquirir una IP estática mediante Google Cloud y asignarle la misma a la máquina virtual en cuestión. En nuestro caso sin embargo, al ser una máquina virtual temporal, se dejará que funcione con la IP cambiante, de manera que cada vez que se inicie la misma se modificará la IP asignada a Virtuoso en la aplicación.

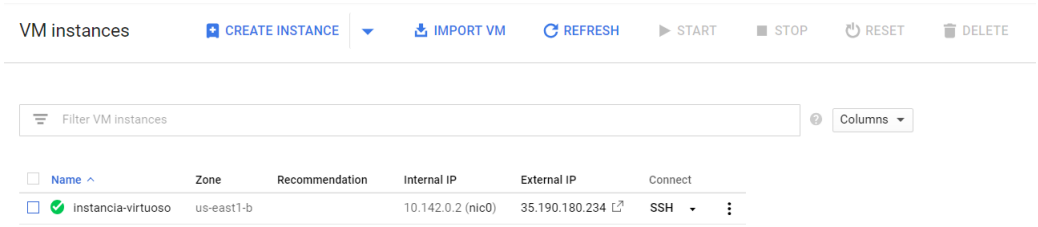

Figura 6.41: Máquina virtual creada en Google Cloud Platform y las opciones ofrecidas por la plataforma.

Para poder conectarse a la máquina virtual, la propia plataforma ofrece un acceso mediante SSH a la misma desde el navegador (Fig. 6.42). Existen otros métodos de conectarse a la misma, pero para la gran mayoría de operaciones en nuestro caso se utilizará dicho servicio SSH ofrecido por Google Cloud Platform.

[28]Plataforma Google Cloud: https://cloud.google.com/

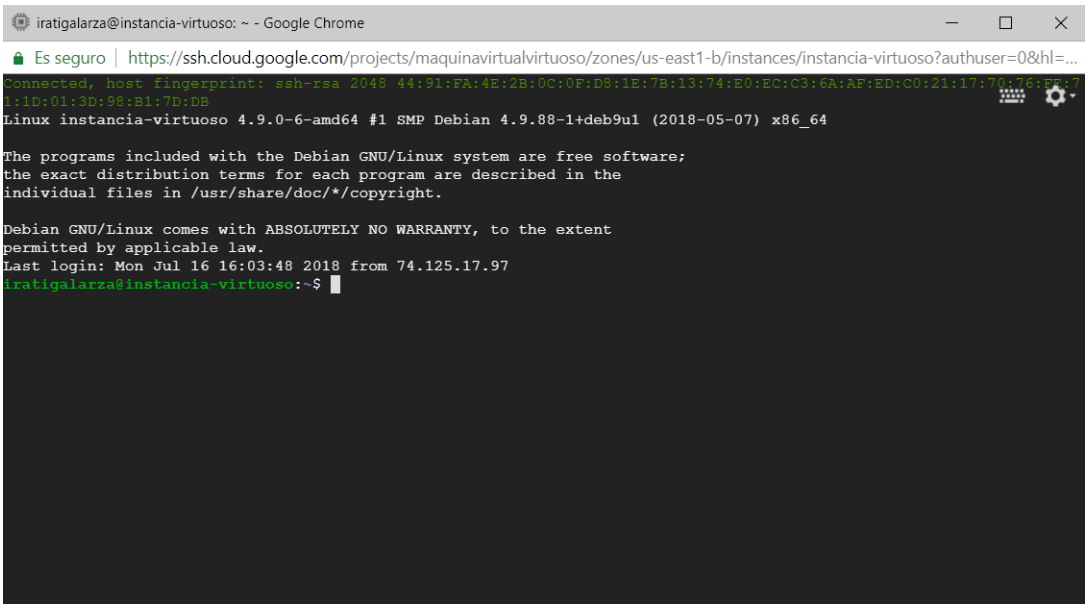

Figura 6.42: Conexión SSH a la máquina virtual ofrecida por Google Cloud Platform.

En cuanto a la instalación de Virtuoso en la máquina virtual, en la documentación oficial de Virtuoso OpenLink se puede encontrar una guía de instalación en sistemas operativos Debian[29].

En dicha documentación se explica que Debian posee de paquetes binarios pre-construídos de Virtuoso, por lo que la opción más sencilla es utilizar dichos paquetes para la instalación de Virtuoso. Sin embargo, uno de los problemas de utilizar dichos paquetes es la versión de Virtuoso que se ofrece. En la instalación de Virtuoso en local se posee de la versión 7.2.4 del mismo, y los paquetes de Virtuoso ofrecidos por defecto en Debian son de la versión 6.1.2, versión notablemente anterior.

Sin embargo, por problemas de compatibilidad con Debian aparecidos en el intento de instalar la última versión de Virtuoso construyéndolo desde código fuente, se decide finalmente utilizar dichos paquetes pre-construídos.

Una vez que Virtuoso OpenLink esté instalado correctamente, esta vez no es necesaria la creación de una instancia de Virtuoso para la utilización de los servicios. Para iniciar los servicios de Virtuoso en Debian basta con ejecutar el comando `virtuoso-t` en la carpeta correspondiente a la base de datos de Virtuoso.

[29] Documentación sobre la instalación de Virtuoso OpenLink en Debian:
http://vos.openlinksw.com/owiki/wiki/VOS/VOSDebianNotes

El resto de características de Virtuoso son muy similares a las explicadas para la instalación de Virtuoso en local, a excepción de las sentencias de inserciones.

En la versión 7.2.4 de Virtuoso, era posible la inserción de datos a través de una sentencia SPARQL 1.1 (Fig. 6.25), pero en la versión 6.1.2 instalada sin embargo, estas sentencias no son compatibles.

Para poder realizar inserciones a través de sentencias SPARQL al servicio de Virtuoso alojado en Debian, la estructura de la sentencia tiene que ser la especificada en la figura 6.43.

```
PREFIX: p: <URI_asociada_al_prefijo_p>
...

INSERT INTO GRAPH <URI_del_grafo_destino>
{
     ... datos_a_insertar ...
}
```

Figura 6.43: Sentencia SPARQL para inserción de datos en Virtuoso OpenLink 6.1.2.

6.7.2. Añadiendo información para la fusión

Por último, para efectuar la unión con éxito, es necesario definir la comunicación entre ambas aplicaciones, de manera que la información necesaria se pase correctamente entre ellas.

El sistema I4TSPS es el encargado de ofrecer al módulo la información sobre la organización a la que pertenece el usuario que inicie sesión. Esta información se tiene almacenada en el servicio de base de datos ofrecido por Firebase, y por tanto, dicha información está almacenada en formato de objetos JSON.

Para el correcto funcionamiento del módulo desarrollado en este TFG (*Semantic Module*), es necesario incluir en dicha base de datos cierta información adicional sobre las máquinas que posee cada organización. En la figura 6.44 se muestra la estructura de dicha información, en la que se especifica por cada organización las máquinas extrusoras que se tienen, el tipo de las mismas y las anomalías que se tienen predefinidas en ellas.

```
{
    "SemanticModule": {
        "Organizations": {
            "-L2PV1Ya30YR-SBlesmI": {
                "1086_WWN_BGY3MW_3":{
                    "type":"extrusora_4_zonas",
                    "anomalies": [
                        {"T4C3B9":"down", "84RATS":"down", "ParMotor":"down"},
                        {"T4C3B9":"up", "84RATS":"up", "ParMotor":"up"},
                        {"VMTKD6":"up", "84RATS":"down", "ParMotor":"down"},
                        {"VMTKD6":"down", "84RATS":"up", "ParMotor":"up"},
                        ...
                    ]
                },
                ...
            }
            ...
        }
    }
}
```

Figura 6.44: Estructura de la información del módulo *Semantic Module* en la base de datos de Firebase.

Por otro lado, el módulo debe ser capaz de acceder a la información comentada para poder hacer uso de la misma. Por ello, se deben modificar las llamadas para obtener información sobre las máquinas de la organización, las anomalías de cada máquina, etc.

Dichas llamadas modificadas se realizan con ayuda de un objeto `ref` definido por la aplicación I4TSPS y que representa la conexión con la base de datos en Firebase. En la figura 6.45 se muestra un pequeño ejemplo de llamada a la base de datos para obtener la información almacenada sobre la organización cuyo identificador corresponda con el contenido de la variable *idOrg*.

```
ref.child(`Modules/SemanticModule/Organizations/${idOrg}`).once('value')
    .then(snap =>{
        // Tratar la información obtenida mediante snap.val()
    });
```

Figura 6.45: Ejemplo de referenciar a la información de la base de datos de Firebase a través de la aplicación.

CAPÍTULO 7

Verificación y pruebas

A lo largo del desarrollo del sistema web se han ido ejecutando distintas pruebas para comprobar el correcto funcionamiento de todos los módulos. Una vez considerado que estos módulos funcionan como se esperaba, se han realizado una serie de pruebas finales para verificar que todas las posibilidades están bien cubiertas y que no existen errores inesperados.

En este capítulo se detallan los aspectos específicos tenidos en cuenta para cada funcionalidad así como un subconjunto de las pruebas finales realizadas en cada uno.

7.1. Pruebas de consulta de datos

Durante el desarrollo del módulo de consulta de datos se han tenido en cuenta tres aspectos principales para determinar que su comportamiento era correcto:

- Que los datos introducidos por los usuarios en los diferentes formularios se recojan correctamente y sin perder información.
- Que estos datos recogidos en los formularios sean correctamente tratados para realizar la consulta correspondiente a Virtuoso.
- Que los resultados de la consulta a Virtuoso sean correctamente tratados de manera que las gráficas muestren los datos reales.

Aparte de estos aspectos básicos que debe cubrir la funcionalidad de consulta de datos, para cada tipo de consulta se han tenido en cuenta otros criterios específicos adicionales para la función que se realiza en las mismas.

A continuación se especifican estos criterios adicionales de cada tipo de consulta y se presenta un *subconjunto* de las pruebas finales realizadas para descubrir posibles errores desconocidos.

7.1.1. Consultas de información general

En las consultas de información general se permite al usuario introducir una serie de restricciones y filtros para obtener unos resultados más específicos. Por este motivo, al realizar las pruebas de este tipo de consulta se han tenido especialmente en cuenta que todas las posibles combinaciones de dichos filtros se realicen correctamente, sin interferir unas con otras.

En la tabla 7.1 están representadas parte de las pruebas finales realizadas, representando aquellas que se considera que aportan información más relevante. En caso de que los resultados de las pruebas sean incorrectos se detallarán a continuación los pasos seguidos para su solución.

Tabla 7.1: Subconjunto de pruebas finales para la consulta de información general

Descripción	Código	Nº sens.	Salida esperada	Salida real	Observaciones
Consultar sin aplicar ningún filtro.	**1.1**	2	Una gráfica continua de todos los datos encontrados sobre los sensores seleccionados.	Una gráfica continua de todos los datos encontrados sobre los sensores seleccionados.	**Correcto.**
	1.2	Todos	Una gráfica continua de todos los datos encontrados sobre los sensores seleccionados.	Una gráfica continua de todos los datos encontrados sobre los sensores seleccionados.	**Mejorable.** Información ofrecida al usuario no completa.

Continuación de la tabla 7.1

Descripción	Código	Nº sens.	Salida esperada	Salida real	Observaciones
Consultar filtrando los resultados entre dos horas concretas.	**2.1**	1	Una gráfica no continua de los datos del sensor seleccionado entre las horas especificadas.	Una gráfica continua de los datos del sensor seleccionado entre las horas especificadas.	**Incorrecto.**
	2.2	Todos	Una gráfica no continua de los datos de los sensores seleccionados entre las horas especificadas.	Una gráfica no continua de los datos de los sensores seleccionados entre las horas especificadas.	**Correcto.**
Consultar filtrando los resultados entre dos fechas concretas.	**3.1**	2	Una gráfica continua de los datos de los sensores seleccionados entre las fechas seleccionadas.	Una gráfica continua de los datos de los sensores seleccionados entre las fechas seleccionadas.	**Correcto.**
	3.2	Todos	Una gráfica continua de los datos de los sensores seleccionados entre las fechas seleccionadas.	Una gráfica continua de los datos de los sensores seleccionados entre las fechas seleccionadas.	**Correcto.**

Continuación de la tabla 7.1

Descripción	Código	Nº sens.	Salida esperada	Salida real	Observaciones
Consultar filtrando los resultados entre dos valores concretos.	**4.1**	2	Una gráfica no continua de los datos de los sensores seleccionados que estén dentro del rango de valores especificado.	Una gráfica no continua de los datos de los sensores seleccionados que estén dentro del rango de valores especificado.	**Correcto.**
	4.2	Todos	Una gráfica no continua de los datos de los sensores seleccionados que estén dentro del rango de valores especificado.	Una gráfica no continua de los datos de los sensores seleccionados que estén dentro del rango de valores especificado.	**Correcto.**
Consultar filtrando los resultados entre dos horas, entre dos fechas y especificando un rango de valores concreto.	**5.1**	3	Una gráfica no continua de los datos de los sensores seleccionados teniendo en cuenta todos los filtros aplicados.	Una gráfica no continua de los datos de los sensores seleccionados teniendo en cuenta todos los filtros aplicados.	**Correcto.**
Consultar agrupando los resultados cada día y mostrando el máximo.	**6.1**	2	Una gráfica de barras con los datos máximos cada día de los sensores seleccionados.	Una gráfica de barras con los datos máximos cada día de los sensores seleccionados.	**Mejorable.** Formato de la fecha no adecuado.

Continuación de la tabla 7.1

Descripción	Código	Nº sens.	Salida esperada	Salida real	Observaciones
	6.2	Todos	Una gráfica de barras con los datos máximos de los sensores seleccionados cada día.	Una gráfica de barras con los datos máximos cada día de los sensores seleccionados.	**Correcto.**
Consultar agrupando los resultados cada hora y mostrando la media.	**7.1**	1	Una gráfica de barras con los datos medios del sensor seleccionado cada hora.	Una gráfica de barras con los datos medios del sensor seleccionado cada hora, pero aparece una hora más de la debida.	**Incorrecto.**
	7.2	Todos	Una gráfica de barras con los datos medios de los sensores seleccionados cada hora.	Una gráfica de barras con los datos medios de los sensores seleccionados cada hora.	**Correcto.**
Consultar agrupando los resultados cada hora, mostrando la media, el mínimo y el máximo y aplicando filtros de hora y valor.	**8.1**	2	Una gráfica por cada sensor seleccionado en la que se muestran los valores medios, mínimos y máximos del mismo cada hora y teniendo en cuenta los filtros.	Una gráfica por cada sensor seleccionado en la que se muestran los valores medios, mínimos y máximos del mismo cada hora y teniendo en cuenta los filtros.	**Correcto.**

Pruebas mejorables:

- Prueba 1.3: Ciertos sensores no son mostrados en la gráfica puesto que no existen datos de los mismos, pero esta información sin embargo no se comunica al usuario. Para su solución en el código se han añadido distintas restricciones que comprueban si Virtuoso ha devuelto datos de un sensor antes de añadirlos en la gráfica. De esta manera, cuando no haya datos de un sensor se le informará por pantalla al usuario.
- Prueba 6.1: El formato de la fecha aparecido por defecto no es el adecuado para las consultas en las que se agrupan los resultados por días, ya que aparece también la hora, indicando medianoche. Para su solución, se especifica que dependiendo de las características de las consultas también se adapte el formato de la fecha y hora indicados en las opciones de la gráfica a mostrar.

Pruebas incorrectas:

- Prueba 2.1: Las gráficas se tratan como continuas teniendo un filtro de horas, por lo que de cara al usuario parece que no hay salto entre las distintas horas. Para solucionar esto se ha cambiado el tipo de gráfica de Google de tipo *Line* a *Scatter*, dejando de ser continua.
- Prueba 7.1: En las consultas en las que los resultados se agrupan por horas aparece un intervalo de hora adicional, de manera que el primer y último intervalo se repiten. El error está en la definición del filtro en la sentencia SPARQL encargada de realizar la consulta, por lo que para su solución se corrige dicha definición.

7.1.2. Consultas de relación entre sensores

En las consultas de relación entre sensores se permite al usuario indicar los sensores que forman parte en la relación de la manera deseada, por lo que al realizar las pruebas se tiene especialmente en cuenta que todas las combinaciones posibles de relación funcionen correctamente. Además, también se tienen en cuenta el funcionamiento de los filtros de valor y fechas disponibles en este tipo de preguntas.

En la tabla 7.2 están representadas parte de las pruebas finales realizadas, representando aquellas que se considera que aportan información más relevante. En caso de que los resultados de las pruebas sean incorrectos, se detallan a continuación los pasos seguidos para su solución.

Tabla 7.2: Subconjunto de pruebas finales para la consulta de relación entre sensores

Descripción	Código	Val. fijo	Salida esperada	Salida real	Observaciones
Consultar un sensor con el valor de otro fijado.	**1.1**	Espec.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene el valor especifico fijado.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene el valor especifico fijado.	**Correcto.**
	1.2	MAX	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene su valor máximo.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene su valor máximo.	**Correcto.**
	1.3	MIN	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene su valor máximo.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene su valor máximo.	**Correcto.**

Continuación de la tabla 7.2

Descripción	Código	Val. fijo	Salida esperada	Salida real	Observaciones
	1.4	MAX entre rango valores	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene el valor máximo de entre el rango de valores indicado.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene el valor máximo entre el rango de valores indicado.	**Correcto.**
	1.5	MIN entre rango valores	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene el valor mínimo de entre el rango de valores indicado.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que el sensor fijado contiene el valor mínimo de entre el rango de valores indicado.	**Correcto.**
Consultar más de un sensor con el valor de otro fijado.	**2.1**	Espec.	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene el valor especifico fijado.	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene el valor especifico fijado.	**Correcto.**

Continuación de la tabla 7.2

Descripción	Código	Val. fijo	Salida esperada	Salida real	Observaciones
	2.2	MAX	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene su valor máximo.	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene su valor máximo.	**Correcto.**
	2.3	MIN	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene su valor mínimo.	No se muestra la gráfica.	**Incorrecto.**
Consultar un sensor con el valor de más de uno fijado.	**3.1**	Espec. en todos los sens.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que los sensores fijados contienen los valores especificos fijados.	Gráfica no continua con los valores del sensor preguntado en los momentos en los que los sensores fijados contienen los valores especificos fijados.	**Correcto.**

Continuación de la tabla 7.2

Descripción	Código	Val. fijo	Salida esperada	Salida real	Observaciones
Consultar más de un sensor con el valor de más de uno fijado.	**4.1**	Espec. en todos los sens.	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que los sensores fijados contienen los valores especificados.	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que los sensores fijados contienen los valores especificados.	**Correcto.**
Consultar más de un sensor con el valor de otro fijado y filtrando entre fechas.	**5.1**	MAX	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene su valor máximo entre las fechas especificadas.	Gráfica no continua con los valores de los sensores preguntados en los momentos en los que el sensor fijado contiene su valor máximo entre las fechas especificadas.	**Correcto.**

Pruebas incorrectas:

- Prueba 2.3: Cuando los datos de alguno de los sensores, tanto de los consultados o de los que se han especificado los valores, no están bien preprocesados, pueden haber problemas a la hora de realizar la consulta a Virtuoso, de manera que ésta no devuelve una respuesta, impidiendo la creación del gráfico. La solución a este problema trata simplemente en asegurar que todos los datos utilizados e insertados en Virtuoso estén bien preprocesados, separando el hecho de que no tengan valores en algunos momentos de los valores 0 como tal.

7.1.3. Consultas de anomalías

En las consultas de búsqueda de anomalías se permite al usuario personalizar completamente las relaciones de anomalía deseadas o seleccionar una de las anomalías predefinidas existentes. Por esto, al realizar las pruebas se tiene especialmente en cuenta el correcto funcionamiento de las relaciones de anomalía predefinidas (hasta ahora, ya que después el usuario podrá definir las que desee) a la vez que el correcto funcionamiento del mayor número de combinaciones posibles a la hora de personalizar las anomalías.

En la tabla 7.3 están representadas parte de las pruebas finales realizadas, representando aquellas que se considera que aportan información más relevante. En caso de que los resultados de las pruebas sean incorrectos, se detallan a continuación los pasos seguidos para su solución.

Tabla 7.3: Subconjunto de pruebas finales para la consulta de anomalías en los valores

Descripción	Código	Nº Sens.	Salida esperada	Salida real	Observaciones
Consultar anomalía personalizada	**1.1**	- Anom. predef. nº 1	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	**Correcto.**

Continuación de la tabla 7.3

Descripción	Código	Nº Sens.	Salida esperada	Salida real	Observaciones
	1.2	- Anom. predef. nº 2	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	**Correcto.**
	1.3	- Anom. predef. nº 3	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	Los datos mostrados en la gráfica no concuerdan con la relación de anomalía indicada.	**Incorrecto.**
	1.4	- Anom. predef. nº 4	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	Gráfica con los datos de los sessores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía.	**Correcto.**

Continuación de la tabla 7.3

Descripción	Código	Nº Sens.	Salida esperada	Salida real	Observaciones
Consultar anomalías personalizadas	**2.1**	2	Gráfica con los datos de los sensores que forman parte de la relación de anomalía personalizada, mostrando su valor cuando ocurre dicha anomalía.	Gráfica con los datos de los sensores que forman parte de la relación de anomalía personalizada, mostrando su valor cuando ocurre dicha anomalía.	**Correcto.**
	2.2	4	Gráfica con los datos de los sensores que forman parte de la relación de anomalía personalizada, mostrando su valor cuando ocurre dicha anomalía.	Gráfica con los datos de los sensores que forman parte de la relación de anomalía personalizada, mostrando su valor cuando ocurre dicha anomalía.	**Correcto.**

Continuación de la tabla 7.3

Descripción	Código	Nº Sens.	Salida esperada	Salida real	Observaciones
Consultar anomalías predefinidas con filtro de fechas	**3.1**	- Anom. predef. nº 1	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía entre las fechas indicadas.	Gráfica con los datos de los sensores que forman parte de la relación de anomalía seleccionada, mostrando su valor cuando ocurre dicha anomalía entre las fechas indicadas.	**Correcto.**
Consultar anomalías personalizadas con el cálculo del par motor incluído en la relación	**4.1**	3	Gráfica con los datos de los sensores que forman parte de la relación de anomalía personalizada, mostrando su valor cuando ocurre dicha anomalía.	Gráfica con los datos de los sensores que forman parte de la relación de anomalía personalizada, mostrando su valor cuando ocurre dicha anomalía.	**Correcto.**

Pruebas incorrectas:

- Prueba 1.3: Cuando se empezó a utilizar el servicio de corrección de timestamps *Data Fix Service*, los datos mostrados en las gráficas de búsqueda de anomalías no concordaban con los datos reales que debían ser mostrados. Esto se debe a no tener en cuenta la falta de valores (valores "*NaN*") en la función de búsqueda de anomalías, por lo que para solucionarlo se debe cambiar dicha función, obviando los momentos en los que no se tengan datos.

7.2. Pruebas de inserción de datos

Durante el desarrollo del módulo de inserción de datos se han tenido en cuenta dos aspectos principales para determinar que su comportamiento era correcto:

- Que las transformaciones de los datos subidos por el usuario en CSV se realizan correctamente y sin perder información en el proceso. Dichos procesos de transformación son:
  1. El paso de los datos del archivo en CSV a JSON. Aquí es importante que la información contenida en el archivo no varíe de un formato a otro, simplemente debe cambiar la estructura.
  2. El uso de la función de corrección de timestamps. Se da por supuesto que la función funciona correctamente, solo es necesario comprobar el correcto uso de la misma.
  3. El paso del JSON resultante de la función a tripletas definidas en Turtle. En este proceso es importante comprobar que la información añadida al crear las tripletas en Turtle sea la correcta (el sensor al que pertenecen los datos, los tipos del mismo, etc.), y además, dicha información tiene que representar las mismas observaciones y valores representadas en el JSON.
- Que los datos sean correctamente insertados en Virtuoso, prestando especial atención a su inserción en el grafo correcto y de la posterior disponibilidad de dichos datos. Para comprobar esto, se han hecho también diferentes consultas de los datos a partir de la interfaz web que ofrece Virtuoso para su SPARQL Endpoint, evitando así confundir posibles errores de la aplicación web a hacer las consultas.

En este caso, el formulario de inserción de datos es considerablemente sencillo, solo ofreciendo la opción de selección del archivo a subir, por lo que para esta funcionalidad no se han realizado numerosas pruebas finales como las mostradas en puntos anteriores. Todas las pruebas realizadas han sido para comprobar los dos puntos básicos comentados en el párrafo anterior.

CAPÍTULO 8

Seguimiento y Control del Proyecto

Durante todo el desarrollo del proyecto se ha realizado el seguimiento y control del mismo, supervisando lo realizado con la planificación inicial e implantando los cambios necesarios para asegurar la correcta finalización del TFG.

En este capítulo se presentan, a modo de conclusión, los resultados finales obtenidos en este proceso de seguimiento continuo del proyecto. Para ello, se detalla el alcance final del proyecto realizado, el control de las incidencias surgidas durante el desarrollo y las diferencias entre la planificación inicial y la finalmente seguida.

8.1. Control del alcance

La base principal del alcance del proyecto no ha variado a lo largo del desarrollo del mismo, de manera que la Estructura de Descomposición de Tareas (E.D.T.). no ha variado en cuanto a su estructura inicial (Fig. 3.2 de la sección 3.3). Sin embargo, por problemas que se detallan en la sección 8.2 de este mismo capítulo, a finales de mayo se decidió que la fecha de finalización del TFG iba a ser retrasada hasta mediados de Septiembre. Aprovechando una pequeña parte del tiempo extra debido a la nueva fecha de finalización decidida, se decidió realizar una pequeña ampliación en el alcance del proyecto, creando nuevas tareas a abordar en ciertos paquetes de trabajo del E.D.T. mencionado.

Por un lado, se decidió dar un paso más en la unión del sistema web desarrollado en

este TFG con el sistema I4TSPS ya comentado en secciones anteriores. En un principio, la unión real de los sistemas no entraba dentro del alcance del proyecto, de manera que únicamente se preparaba el sistema para dicha futura unión, pero sin llevarla a cabo. Se decidió entonces llevar a cabo esta unión dentro del alcance del TFG, al menos de una manera superficial para comprobar el correcto funcionamiento de ambos sistemas en conjunto.

Por otro lado, la configuración de la herramienta Virtuoso OpenLink solo se había planteado como una tarea, sin diferenciar si la misma era en local o en una localización pública. Esto se debe a que no se tenía realmente definido cómo iba a ser dicho alojamiento de Virtuoso, pero se preveía que ambas configuraciones iban a ser parecidas si no la misma. Sin embargo, a causa de la unión con el sistema I4TSPS, a mediados del desarrollo del proyecto se llegó al acuerdo de alojar Virtuoso en una máquina virtual Debian proporcionada por Google. Esto se debe a que actualmente el servicio desarrollado por el grupo BDI para la aplicación web I4TSPS está alojado en una máquina virtual de estas características, por lo que previendo una futura adición de Virtuoso a dicho servicio, se decide llevar a cabo el mismo enfoque para comprobar su funcionamiento. Se observó que la configuración de Virtuoso sí que variaba notablemente dependiendo de en qué localización se instale, ya que se trata de dos sistemas operativos diferentes: Windows en local y Debian en la máquina virtual.

A continuación se citan las tareas finalmente añadidas, mencionando únicamente los paquetes de trabajo en los que se han producido cambios y detallando los objetivos de dichas nuevas tareas.

En el paquete de trabajo **Alojamiento de Datos (LD):**

- LD.1: Preparar y configurar Virtuoso de manera local para el alojamiento de los datos necesarios.
- LD.2: Implementar la automatización del proceso de alojamiento de los datos.
- **LD.3**: Preparar y configurar Virtuoso en una máquina virtual con sistema operativo Debian.

En el paquete de trabajo **Alojamiento Web (LSW):**

- *LSW.1: Configurar Firebase y el proyecto para el correcto alojamiento del Sistema Web.* Esta tarea ha sido eliminada.

- *LSW.2: Alojamiento final y continuo del proyecto, solucionando posibles errores que aparezcan una vez esté en línea.* Esta tarea ha sido eliminada.
- **LSW.1'**: Preparar la aplicación web para su alojamiento en la aplicación I4TSPS ya en desarrollo, alojada en Firebase.
- **LSW.2'**: Realizar la unión de la aplicación web desarrollada en este proyecto con I4TSPS.

En el paquete de trabajo **Pruebas Datos (PD)**:

- PD.1: Verificación de que la traducción de datos al formato Turtle es correcta.
- *PD.2. Esta tarea ha sido modificada y separada en dos nuevas tareas:*
  - **PD.2.1**: Verificación de que el alojamiento automático de los datos funciona correctamente en local.
  - **PD.2.2**: Verificación de que el alojamiento automático de los datos funciona correctamente en la máquina virtual.

En el paquete de trabajo **Pruebas Sistema Web (PSW)**:

- PSW.1: Verificación de que las interfaces creadas para el Sistema Web funcionan correctamente.
- PSW.2: Verificación del correcto funcionamiento de la generación de las consultas SPARQL a partir de los formularios.
- *PSW.3. Esta tarea ha sido modificada y separada en dos nuevas tareas:*
  - **PSW.3.1**: Verificación de que la conexión con Virtuoso en local es correcta.
  - **PSW.3.2**: Verificación de que la conexión con Virtuoso en la máquina virtual es correcta.
- PSW.4: Verificación de que las gráficas se muestran correctamente, correspondiendo con los datos iniciales.
- *PSW.5: Verificación de que el sistema web funciona correctamente después de su alojamiento en Firebase.* Esta tarea ha sido eliminada.

- **PSW.5'**: Verificación de que la unión con el sistema web de I4TSPS funciona correctamente.

Por último, en el paquete de **Conexión con Datos (CSW)** ha habido una adición momentánea de una tarea durante cierto tiempo del desarrollo del proyecto. Dicha tarea tiene como objetivo la creación de un servicio RESTful en Java, y su adición se debe a los problemas surgidos al implementar la conexión del sistema web con Virtuoso, de manera que para solucionarlos se creyó necesaria la creación de dicho servicio. Sin embargo, el problema fue finalmente solucionado pudiendo implementar la conexión como se tenía planeado desde un principio, por lo que dicha tarea se ha eliminado puesto que no tiene relevancia para el resultado final del proyecto. En la sección 8.2 de este mismo capítulo se detalla el problema mencionado y el impacto que tuvo en el seguimiento del desarrollo del TFG.

8.2. Control de incidencias

Durante el desarrollo del proyecto han ido apareciendo una serie de problemas e incidencias que han dificultado el cumplimiento del planteamiento inicial. Estas incidencias han correspondido en su mayoría a riesgos ya previstos al principio del proyecto pero que, sin embargo, no han podido ser evitados.

A continuación se detallan dichas incidencias surgidas, el impacto que han tenido sobre el desarrollo del TFG y las tareas de mitigación concretas llevadas a cabo en su aparición.

8.2.1. Planificación incorrecta

En este TFG, la gran mayoría de las herramientas utilizadas eran nuevas para mí, no las había utilizado con anterioridad. Esto requiere invertir una gran parte del tiempo del inicio del proyecto en la adquisición de conocimientos sobre dichas herramientas, cosa que ya había sido contemplado en la planificación inicial, pero la novedad de las herramientas también supone que el desarrollo con las mismas será considerablemente más lento que con herramientas ya conocidas. Ésto último no se tuvo en cuenta al realizar el planteamiento inicial, tomando erróneamente como referencia el tiempo habitual necesitado para desarrollar aplicaciones en otros proyectos sin considerar el grado de conocimiento sobre las tecnologías utilizadas.

Esto supuso una mala previsión de la dedicación y de los periodos de desarrollo de las tareas involucradas con dichas herramientas, incidiendo en definitiva en una incorrecta

planificación inicial del proyecto. Para poder hacer frente a los retrasos surgidos en dichas tareas en cuanto a los periodos planeados, se han tenido que realizar continuas modificaciones en la planificación, sobre todo en los primeros meses del TFG. Una vez lograda una familiarización con las herramientas que suponían una novedad, las dedicaciones de las tareas involucradas con las mismas disminuyeron, ajustándose más a lo previsto en el planteamiento inicial.

8.2.2. Problemas surgidos a causa de Virtuoso

Una de estas herramientas novedosas mencionadas en la sección anterior es Virtuoso, utilizada para el almacenamiento de datos en RDF y su posterior consulta, y cuyo uso ha generado una ralentización adicional en los periodos de desarrollo de las tareas involucradas con la misma.

Aparte del tiempo extra de desarrollo por ser una herramienta no conocida, el entendimiento de su funcionamiento ha sido mucho más costoso de lo esperado y planeado. El gran número de funcionalidades ofrecidas por Virtuoso la convierten en una herramienta poco intuitiva y, desde mi punto de vista, realmente compleja de comprender. A pesar de tener disponible una extensa documentación accesible por internet, resultó difícil encontrar en la misma las causas y soluciones de errores concretos que iban apareciendo durante la utilización de dicha herramienta.

De entre estos problemas y errores surgidos se destacan dos que han incidido especialmente en el desarrollo de este TFG:

- **Inserción de datos a través del servicio web de Virtuoso.** Como ya se ha explicado en la sección 6.3, Virtuoso ofrece un servicio de realización de consultas SPARQL a través de la ruta `/sparql` de su servicio web. En esta ruta sin embargo, está deshabilitada por defecto la posibilidad de realizar consultas de inserción de datos, funcionalidad necesaria para el desarrollo del sistema web de este TFG. En la documentación de Virtuoso no se ha encontrado ninguna mención sobre el problema de hacer inserciones de datos a un grafo mediante una petición HTTP, por lo que las fuentes para la búsqueda de soluciones han sido variadas. Para empezar, se investigaron otras rutas destinadas también a la realización de consultas SPARQL, como `/sparql-auth`, a la que se le ha de proporcionar información de usuario y contraseña para su utilización. Esta ruta puede ser accedida y utilizada sin problemas desde la interfaz web, pero no se ha conseguido hacer uso de dicha ruta mediante una petición HTTP. Abordando el problema desde otro enfoque, se

decidió implementar un servicio RESTful en Java, haciendo uso de las librerías que ofrece Virtuoso para Java. De esta manera, el servicio RESTful era el receptor de las peticiones HTTP del sistema web, haciendo de intermediario entre el sistema y Virtuoso, lo que soluciona el problema mencionado. Finalmente, se consiguió cambiar la configuración de Virtuoso para poder insertar datos mediante la ruta `/sparql` del servicio web como se tenía desde un principio planeado, eliminando el servicio RESTful.

- **Número de datos máximos a insertar a través del servicio web de Virtuoso.** Una vez solucionada la inserción de datos a través de la ruta `/sparql`, surgió otro problema relacionado: el límite de información a enviar en cada petición realizada a Virtuoso es demasiado pequeño. En el sistema web desarrollado en este TFG, se les da la opción a los usuarios de insertar en la aplicación datos de sensores contenidos en un fichero CSV, los cuales generalmente contienen información de una semana entera. Al traducir estos datos a RDF, se generan al rededor de 3.600.000 tripletas a partir de cada archivo, las cuales son después insertadas mediante peticiones HTTP a la ruta `/sparql` de Virtuoso. Sin embargo, las peticiones HTTP al servicio web de Virtuoso cuentan por defecto con un límite de información a recibir, lo que conlleva que solo se puedan insertar un máximo de 160 tripletas en cada petición, teniendo ésto un impacto directo en el tiempo necesitado para la operación. Aunque la inserción funcione correctamente, el tiempo de espera para la realización de la misma no es viable para una aplicación real. Este problema sin embargo, no ha conseguido ser solucionado en el desarrollo del proyecto, pero dentro del grupo de investigación BDI se seguirá trabajando en ello.

Los errores surgidos a partir de Virtuoso han supuesto retrasos en los periodos de cumplimiento de las tareas que abordan los mismos, lo que ha vuelto a conllevar pequeñas modificaciones continuas de la planificación, cambiando el orden de realización de tareas y dividiendo las dedicaciones previstas hasta conseguir solucionar los errores.

8.2.3. Problemas con el hardware utilizado

A finales de mayo surgió una incidencia inesperada que afectó directamente al desarrollo del proyecto: el ordenador portátil en el que se estaba trabajando dejó de funcionar.

El TFG en desarrollo pudo ser recuperado en su totalidad, pero el ordenador portátil utilizado sin embargo no ha podido seguir sirviendo de entorno de trabajo para el proyecto.

Esto supuso tener que volver a configurar todo el entorno necesario en un nuevo ordenador portátil para poder llevar a cabo el desarrollo del TFG.

Esta incidencia causó una semana de retraso en el desarrollo del proyecto en un momento crucial de la planificación, ya que después de distintas modificaciones en la misma, pequeños retrasos podían suponer la inviabilidad del proyecto. Por esto, se tuvo que modificar la planificación añadiendo grandes cambios a la misma para asegurarse el correcto cumplimiento del TFG para conseguir los objetivos impuestos.

8.3. Control de la planificación

Como se ha presentado en la sección anterior, distintas incidencias ocasionaron múltiples modificaciones de la planificación, alterando periodos de realización de tareas y modificando el orden de realización de las mismas. A partir de la incidencia en el hardware utilizado, se llevó a cabo la modificación más grande de la planificación, ya que se decidió retrasar la finalización del proyecto de julio a septiembre.

En la tabla 8.1 se muestra una comparación de las fechas de finalización previstas y las fechas de finalización reales de las distintas tareas definidas en el proyecto. Además, también se expresa una aproximación de la desviación habida entre las mismas, para una mejor visualización de la comparación.

Tabla 8.1: Tabla comparativa de las fechas de finalización previstas y las fechas de finalización reales de las tareas.

Tarea	Fecha fin prevista	Fecha fin real	Desviación
Gestión Conocimiento (GC)	**18/05/18**	**04/07/18**	**+2 meses**
Adquisición Competencias (AC)	24/01/18	04/07/18	+2 meses
AC.1: RDF y SPARQL	24/01/18	24/01/18	
AC.2: Protégé	04/02/18	04/02/18	
AC.3: React.js y Materialize	29/03/18	31/03/18	+2 días
AC.4: Firebase	29/03/18	04/04/18	+1 semana
AC.5: OpenLink Virtuoso	28/04/18	31/05/18	+1 mes
AC.6: Gráficas de Google	16/05/18	11/06/18	+1 mes
AC.7: Herramientas secundarias	18/05/18	04/07/18	+2 meses
Diseño (DI)	**18/05/18**	**30/07/18**	**+3 meses y 1/2**
Diseño Ontología (DIO)	13/04/18	30/07/18	+3 meses y 1/2
DIO.1: Primera versión ontología	09/02/18	20/02/18	+1 semana y 1/2

Table 8.1 continued from previous page

Tarea	Fecha fin prevista	Fecha fin real	Desviación
DIO.2: Mejorar ontología	13/04/18	30/07/18	+3 meses y 1/2
Diseño Interfaces (DII)	18/05/18	03/07/18	+1 mes y 1/2
DII.1: Primera versión interfaces	02/03/18	11/04/18	+1 mes
DII.2: Mejorar interfaces	18/05/18	03/07/18	+1 mes y 1/2
Datos (D)	**03/05/18**	**06/07/18**	**+2 meses**
Traducción de Datos (TD)	20/04/18	26/04/18	+1 semana
TD.1: Estructura datos en RDF	15/04/18	21/04/18	+1 semana
TD.2: Módulo anotación RDF	20/04/18	26/04/18	+1 semana
Alojamiento de Datos (LD)	03/05/18	06/07/18	+2 meses
LD.1: Configurar Virtuoso local	23/04/18	02/05/18	+1 semana
LD.2: Implementar alojamiento	03/05/18	26/06/18	+1 mes y 1/2
LD.3: Configurar Virtuoso MV	–	*06/07/18*	–
Sistema Web (SW)	**08/06/18**	**26/07/18**	**+1 mes y 1/2**
Desarrollo Interfaces (DSW)	25/05/18	23/07/18	+1 mes y 1/2
DSW.1: Desarrollo inicial	10/04/18	12/05/18	+1 mes
DSW.2: Cambios y mejoras	25/05/18	13/07/18	+1 mes y 1/2
Conexión con Datos (CSW)	15/05/18	31/05/18	+2 semanas
CSW.1: Preguntas SPARQL	05/05/18	18/05/18	+1 semana y 1/2
CSW.2: Conexión Virtuoso	11/05/18	21/05/18	+1 semana y 1/2
CSW.3: Recogida datos	15/05/18	31/05/18	+1 semana
Generar Gráficas (GSW)	23/05/18	01/07/18	+1 mes y 1/2
GSW.1: Añadir datos a gráficas	19/05/18	06/06/18	+2 semanas y 1/2
GSW.2: Características gráficas	23/05/18	01/07/18	+1 mes y 1/2
Alojamiento Sistema Web (LSW)	08/06/18	26/07/18	+1 mes y 1/2
Eliminada (LSW.1)	*05/04/18*	–	–
Eliminada (LSW.2)	*08/06/18*	–	–
LSW.1': Cambios para unión	–	*22/07/18*	–
LSW.2': Unión con I4TSPS	–	*26/07/18*	–
Pruebas (P)	**13/06/18**	**27/07/18**	**+1 mes y 1/2**
Pruebas datos (PD)	30/04/18	10/07/18	+2 meses y 1/2
PD.1: Pruebas traducción datos	27/04/18	02/05/18	+1 semana
Modificada (PD.2:)	*30/04/18*	–	–
PD.2.1: Pruebas inserción local	–	*03/07/18*	–

Table 8.1 continued from previous page

Tarea	Fecha fin prevista	Fecha fin real	Desviación
PD.2.2: Pruebas inserción MV	–	*10/07/18*	–
Pruebas Sistema Web (PSW)	13/06/18	27/07/18	+1 mes y 1/2
PSW.1: Pruebas interfaces	07/05/18	18/07/18	+2 meses y 1/2
PSW.2: Pruebas SPARQL	06/05/18	25/05/18	+3 semanas
Modificada (PSW.3)	*13/05/18*	–	–
PSW.3.1: Pruebas conexión local	–	*29/05/18*	–
PSW.3.2: Pruebas conexión MV	–	*10/07/18*	–
PSW.4: Pruebas gráficas	25/05/18	11/07/18	+1 mes y 1/2
Eliminada (PSW.5)	*13/06/18*	–	–
PSW.5': Pruebas unión I4TSPS	–	*27/07/18*	–
Gestión (G)	**06/07/18**	***No finalizado, nueva previsión: 14/09/18***	***+2 meses***
Planificación (PL)	14/03/18	31/03/18	+2 semanas
PL.1: Identificación requisitos	08/02/18	08/02/18	
PL.2: Planificación inicial	26/02/18	26/02/18	
PL.3: Actualización plan. inicial	14/03/18	31/03/18	+2 semanas
Seguimiento y Control (SyC)	06/07/18	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*
SyC.1: Reunión con directoras	06/07/18	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*
SyC.2: Diario tareas	17/01/18	17/01/18	
SyC.3: Hoja cálculo dedicaciones	03/02/18	03/02/18	
SyC.4: Recopilación información	06/07/18	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*
SyC.5: Contraste con planificación	06/07/18	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*

Table 8.1 continued from previous page

Tarea	Fecha fin prevista	Fecha fin real	Desviación
Documentación (DOC)	**06/07/18**	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*
Informe (INF)	22/06/18	05/09/18	+2 meses y 1/2
INF.1: Configurar LaTeX	24/02/18	24/02/18	
INF.2: Desarrollo memoria	22/06/18	05/09/18	+2 meses y 1/2
Defensa (DEF)	06/07/18	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*
DEF.1: Crear presentación	28/06/18	*No finalizado, nueva previsión: 10/09/18*	*+2 meses*
DEF.2: Repasar y mejorar	06/07/18	*No finalizado, nueva previsión: 14/09/18*	*+2 meses*

En cuanto al tiempo necesitado para la realización de dichas tareas, en la tabla 8.2 se recoge una comparación entre las dedicaciones previstas para cada fase del proyecto y las dedicaciones realmente invertidas en las mismas.

Tabla 8.2: Tabla comparativa de la dedicación prevista y la dedicación real del proyecto.

	Dedicación Prevista	Dedicación Real
Gestión de Conocimientos (GC)	38 horas	31 horas
Diseño (DI)	20 horas	41 horas
Datos (D)	45 horas	66 horas
Sistema Web (SW)	97 horas	145 horas 50 min
Pruebas (P)	30 horas	31 horas
Gestión	44 horas	26 horas 40 min
Documentación (D)	82 horas	137 horas
Total	**356 horas**	**478 horas 10 min**

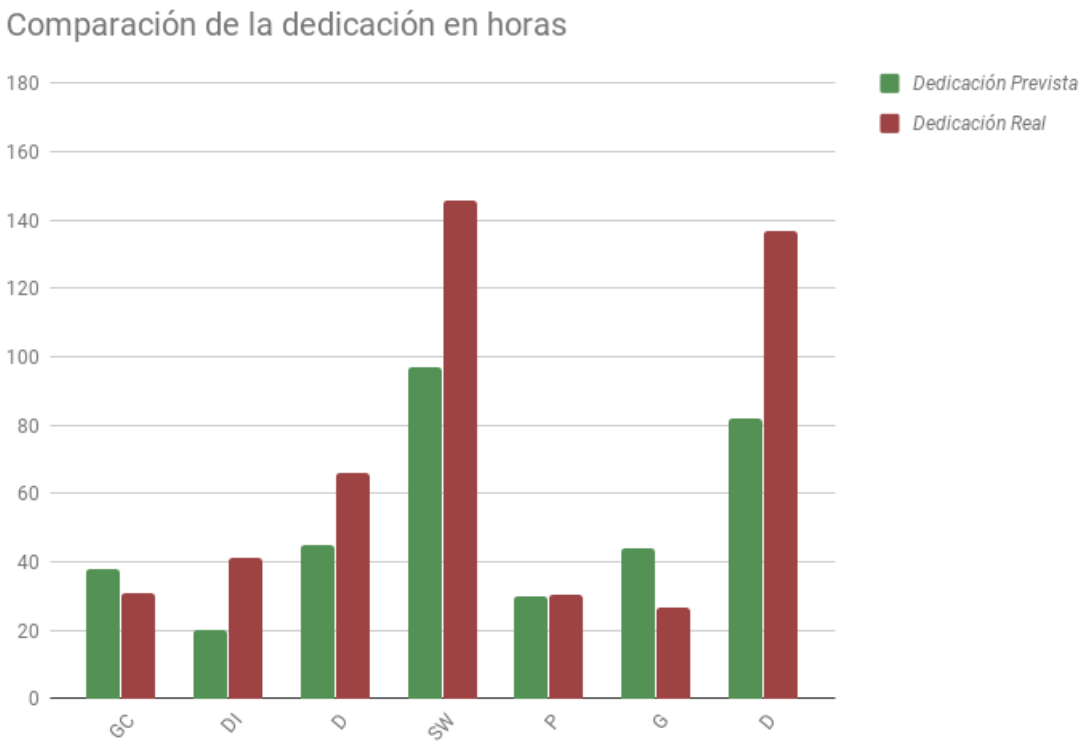

Figura 8.1: Gráfica comparativa de la dedicación prevista y la dedicación real del proyecto en horas.

De la misma manera, en la figura 8.1 se muestra una comparación gráfica de dichas dedicaciones.

Como se puede observar, la dedicación final del proyecto ha resultado ser superior a la inicialmente prevista. Gran parte de este aumento en la dedicación es una consecuencia directa de las incidencias detalladas en la sección anterior, de manera que las tareas de Datos y Sistema Web han aumentado de una manera muy considerable. Parte del aumento de las dedicaciones, sobre todo de las del Sistema Web, son causa de la pequeña adición de nuevas tareas comentadas en la primera sección de este capítulo.

El aumento de tiempo para la realización del proyecto también ha incidido en el aumento de la dedicación de Diseño, ya que se ha aprovechado el tiempo extra para perfeccionar y prestar más atención al diseño de la ontología y mejorar el diseño de la interfaz del sistema.

Por otro lado, la dedicación prevista para la Gestión del proyecto es, sorprendentemente, mayor a la dedicación real invertida en la misma. Esto ha sido en parte por una mala previsión del tiempo necesario para la gestión del proyecto y, por otro lado, también indica

una mejorable realización de la gestión del mismo.

CAPÍTULO 9

Conclusiones y líneas futuras

9.1. Conclusiones

Este TFG se planteó como una prueba de concepto que demostrara el potencial de la utilización de técnicas semánticas en un entorno de Industria 4.0, y una vez finalizado el proyecto se puede concluir que la prueba de concepto ha sido desarrollada con éxito, cumpliendo los objetivos planteados al comienzo del TFG y reforzando la cabida y el interés de las técnicas semánticas en el ámbito de la Industria 4.0.

El sistema web desarrollado se sustenta en nociones de la llamada *Web Semántica*, utilizando la información proporcionada por una ontología que describe el dominio concreto analizado y almacenando los datos del sistema en formato RDF y en base a la misma.

La incorporación de tecnologías de la Web Semántica a nuestro sistema web no ha interferido con el ámbito de Industria 4.0 en el que se ha enmarcado el proyecto, si no que lo ha enriquecido.

Las conclusiones finales obtenidas a partir del desarrollo de este TFG se han desarrollado teniendo en cuenta tres enfoques distintos: las conclusiones técnicas del proyecto, las correspondientes a la transferencia real del proyecto y, finalmente, las conclusiones personales del mismo.

A continuación se detallan dichas conclusiones finales diferenciándolas a partir de los tres

enfoques mencionados.

9.1.1. Conclusiones a nivel técnico

Como se ha mencionado en la introducción del capítulo, en este TFG se ha realizado el desarrollo de un sistema web basado en el uso de técnicas semánticas en el ámbito de la Industria 4.0.

Tomando un enfoque técnico del proyecto, este desarrollo ha implicado lo siguiente:

1. El **desarrollo de una ontología** como extensión de la ontología SSN y utilizando conceptos de la QUDT. En esta ontología se describen los sensores de la extrusora cuyos datos serán almacenados y analizados en el sistema. Para el desarrollo de dicha ontología se han seguido los pasos de la metodología NeOn, reutilizando la ontología SSN como base para la descripción de los sensores y la ontología QUDT para la representación de las propiedades observadas por dichos sensores y sus unidades.

2. La definicion, desarrollo e implementación del **módulo de anotación RDF** de las series temporales. Este módulo anota las series temporales de datos de los sensores en formato RDF para su posterior almacenamiento en el repositorio de datos. Dichas series temporales se contienen en un archivo en formato CSV y encuentran únicamente un valor y una marca de tiempo asociada al mismo en cada fila del documento. El módulo anota todos los datos contenidos en estos fichero en formato RDF siguiendo la ontología desarrollada.

3. La **identificación de tres tipos distintos de consultas** y su implementación. Para la consulta de los datos de la extrusora se identifican tres tipos distintos de preguntas: de información general sobre los sensores, de relación entre los valores de los sensores y de búsqueda de anomalías en los datos de los sensores. Se considera que cada una de éstas proporcionan un interés específico en los usuarios de la aplicación. Dichas consultas se implementan siguiendo las bases de los sistemas de consultas visuales, dirigiendo así el sistema a usuarios inexpertos en las técnicas de extracción y análisis de datos.

4. El diseño y desarrollo de un **módulo de generación de consultas SPARQL**. Éste módulo genera las sentencias SPARQL correspondientes a las consultas creadas por el usuario a través de la interfaz, posibilitando la extracción de los datos deseados en RDF.

5. El desarrollo de la **representación gráfica de los resultados**. Los resultados obtenidos en RDF se visualizan de manera gráfica, personalizando determinadas características dependiendo del tipo de consulta realizada. También se muestra en éstos gráficos información asociada a los sensores, como la representación visual de los *outliers* de los mismos, y un resumen de la pregunta realizada para un seguimiento eficaz de las consultas realizadas.

9.1.2. Conclusiones a nivel de transferencia real

Tomando un enfoque dirigido al nivel de transferencia real con el que se ha desarrollado este TFG, se pueden destacar ciertos puntos de esto:

- El manejo de **series temporales reales**. En el proyecto se ha hecho uso de series temporales con datos reales de los sensores descritos en el sistema. Estas series contienen los datos de una semana completa, teniendo un valor por cada segundo pasado a excepción de ciertos intervalos en los que los sensores no han recogido datos. El tamaño medio de dichas series es de 13.300KB.
- La **presentación a dos representantes de la industria**. Se ha realizado una presentación del sistema web desarrollado a dos representantes de las empresas Savvy y Urola, las cuales son dos de las empresas involucradas en el proyecto real de Industria 4.0 en el que se basa este TFG. Información concreta sobre dichas empresas y su rol en el proyecto real mencionado ya ha sido detallado en la sección 2.1.
- La **valoración positiva del sistema** de los representantes de la industria. Los representantes de la industria mencionados en el punto anterior han valorado positivamente el sistema web presentado, viéndolo como una herramienta útil para los directores de desarrollo de producto y con mucho potencial didáctico para el personal que tenga que ser formado. En cuanto a los directores de desarrollo del producto, se ven especialmente relevantes las consultas de relación entre los valores de los sensores y las de búsqueda de anomalías en los datos, mientras que las consultas informativas se ven mejor enfocadas a las tareas didácticas mencionadas.

9.1.3. Conclusiones a nivel personal

Por último, considerando un enfoque más personal de las conclusiones, éstas están directamente relacionadas con los objetivos definidos en el planteamiento inicial de este proyecto.

A continuación se hace una mención de dichos objetivos, detallando el nivel de cumplimiento de los mismos y las conclusiones sacadas de cada uno.

- *Creación de un sistema web que permita a un responsable de planta de fabricación la consulta de datos generados por sensores industriales*. Este era el objetivo principal del TFG, siendo el producto a desarrollar en el mismo, y como bien se ha expresado en la breve introducción de esta sección, dicho objetivo ha sido correctamente cumplido. El sistema web desarrollado cumple todos los requisitos detallados en el planteamiento inicial del mismo.

- *Aprendizaje de la utilización de tecnologías semánticas en un entorno de Big Data*. Las técnicas semánticas eran un concepto completamente nuevo antes del comienzo de este TFG, y una vez finalizado, se ha conseguido alcanzar un nivel de conocimiento medio sobre las mismas, cumpliendo con totalidad el objetivo impuesto. El conocimiento de estas técnicas ha supuesto una gran aportación a la formación personal, siendo éste un campo en constante auge y que cada vez toma más relevancia en el mundo de la informática. Además, como opinión personal, realmente me ha parecido un campo muy interesante, y cuyo uso aporta una gran cantidad de posibilidades y beneficios en una infinidad de ámbitos.

- *Aprendizaje del desarrollo de Sistemas Web mediante una herramienta basada en las aplicaciones de una sola página*. Al igual que las técnicas semánticas, las aplicaciones de una sola página eran un concepto totalmente nuevo y con el que no se había trabajado, y durante el desarrollo de este proyecto se ha conseguido alcanzar un nivel de conocimiento considerablemente alto sobre las mismas y sobre la herramienta específica utilizada: React.js. Estas aplicaciones se basan en unos conceptos que difieren completamente de los clásicamente asociados a los sistemas web y los que han sido aprendidos a lo largo de la carrera, por lo que aportan algo totalmente nuevo e interesante a la formación personal. Como opinión personal, considero que dichos conceptos de las aplicaciones de una sola página facilitan de una manera extraordinaria el desarrollo de los sistemas web, mejorando además la experiencia de los usuarios.

- *Conocer y utilizar un lenguaje de diseño de interfaces*. A pesar de tener un conocimiento previo muy básico sobre los lenguajes de diseño, esto no se había puesto en práctica en ningún proyecto previo a este TFG, y en el desarrollo de este proyecto se ha cumplido el objetivo en cuestión. Para ello se han puesto en práctica las bases

del *Material Design*, un lenguaje de diseño desarrollado por Google, creando así la interfaz de la aplicación web. Personalmente, esto me ha ayudado a establecer conceptos básicos sobre el diseño de las interfaces, entendiendo mejor la importancia del mismo para poder aspirar a ofrecer una buena experiencia de usuario.

- *Realizar y gestionar un proyecto de este calibre de principio a fin.* El TFG ha sido finalizado con éxito, consiguiendo la realización y gestión del proyecto. Por otro lado, también considero que la gestión del proyecto no ha sido la óptima, ya que las incidencias ocurridas durante el proceso junto con la errónea planificación inicial han resultado en el retraso de la fecha de finalización del proyecto. Sin embargo, el haber previsto y realizado el cambio de fecha de finalización del proyecto con tiempo suficiente para no alterar la viabilidad del mismo lo considero como un punto positivo a mencionar.

9.2. Posibles mejoras

En esta sección se expresan las posibles mejoras que podrían implementarse, en un futuro cercano, en el sistema web desarrollado, mejorando así su calidad:

- **Optimización de los tiempos de espera**. Dada la cantidad de datos que es tratada en el dominio concreto, una característica muy frecuente en ámbitos de industria 4.0, el tiempo de espera de ciertos procesos del sistema web es demasiado alto. Aplicar un mayor preprocesado de los datos antes de ser almacenados y una optimización en el tratamiento de los datos podría mejorar en gran medida este aspecto.
- **Añadir más posibilidades de personalización a los diferentes formulario de consulta**. Concretar las distintas consultas que son más útiles para los usuarios de la aplicación, añadiendo posibles mejoras en la personalización de dichas consultas.
- **Modularización del sistema web**. Extraer parte del proceso de tratamiento de datos a un servicio intermedio externo, evitando una sobrecarga de datos en el cliente.

9.3. Líneas futuras

En esta sección se detallan las posibles líneas futuras que podrán aplicarse al sistema web desarrollado. A diferencia de las mejoras representadas en la sección anterior, la implementación de las líneas futuras podrían tener lugar en un futuro más lejano a causa del incremento en el coste del desarrollo de las mismas.

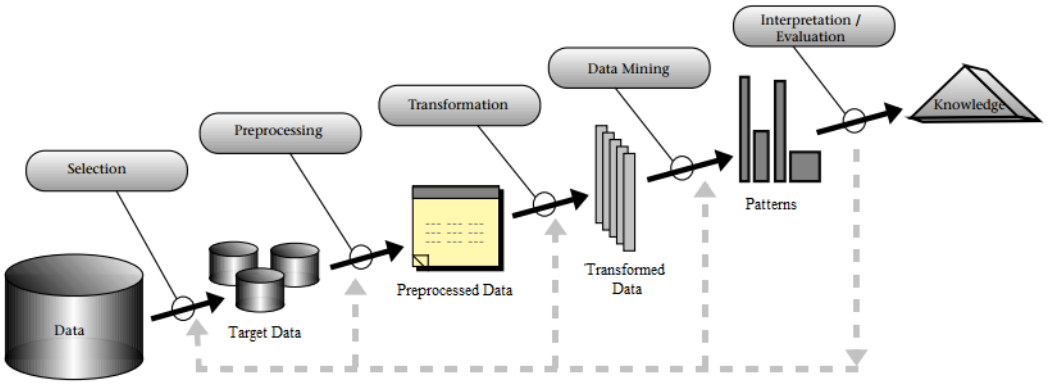

Figura 9.1: Visión general de las fases que componen el ciclo KDD.

- **Utilizar las técnicas semánticas en otras fases del proceso KDD**. Ahora mismo el proyecto utiliza las técnicas semánticas a nivel de anotación y extracción de datos. Haciendo referencia al ciclo *Knowledge Discovery in Databases (KDD)* (Fig. 9.1), se posicionaría en la fase de selección de datos, con una pequeña parte de preprocesado de los mismos, por lo que lo ideal sería poder seguir avanzando en las diferentes fases de dicho ciclo, aprovechando para ello el potencial de las técnicas semánticas.

- **Utilizar todo el potencial de Virtuoso**. Virtuoso es una herramienta enorme que ofrece una infinidad de opciones y posibilidades. Se podrían analizar dichas opciones utilizando así las más convenientes.

Anexos

ANEXO A

Diagramas de secuencia

A continuación se presentan los diagramas de secuencia asociados a las diferentes funcionalidades ofrecidas en la aplicación web desarrollada.

- **Consultar datos de sensores**: figura A.1.
- **Consultar información general sobre cada sensor**: figura A.2.
- **Consultar relación entre sensores**: figura A.3.
- **Consultar anomalías en sensores**: figura A.4.
- **Añadir relación de anomalía predefinida**: figura A.5.
- **Eliminar relación de anomalía predefinida**: figura A.6.
- **Insertar datos de sensores**: figura A.7.
- **Subir datos a Virtuoso**: figura A.8.
- **Descargar fichero Turtle**: figura A.9

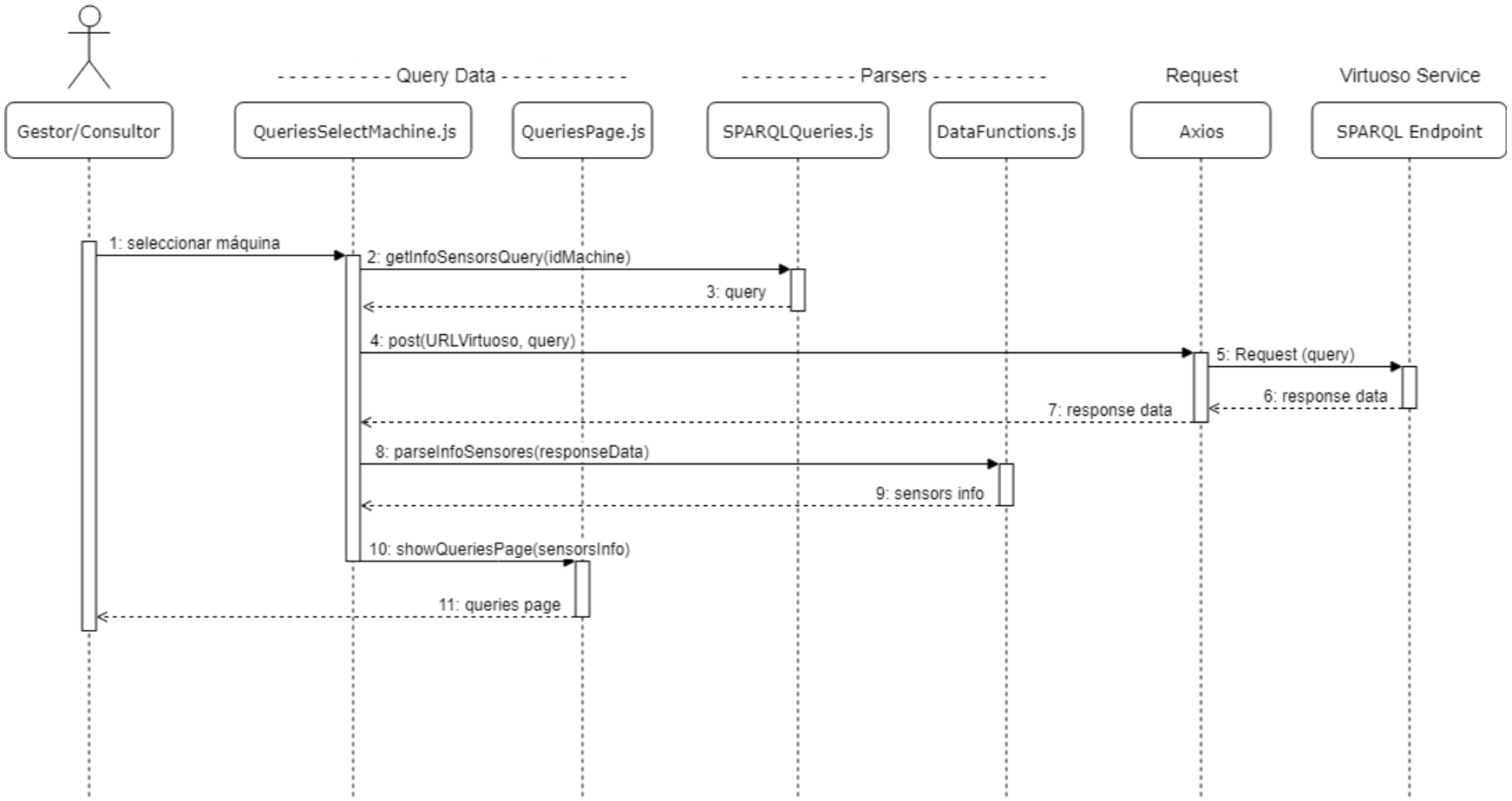

Figura A.1: Diagrama de secuencia de la funcionalidad *Consultar datos sensores*

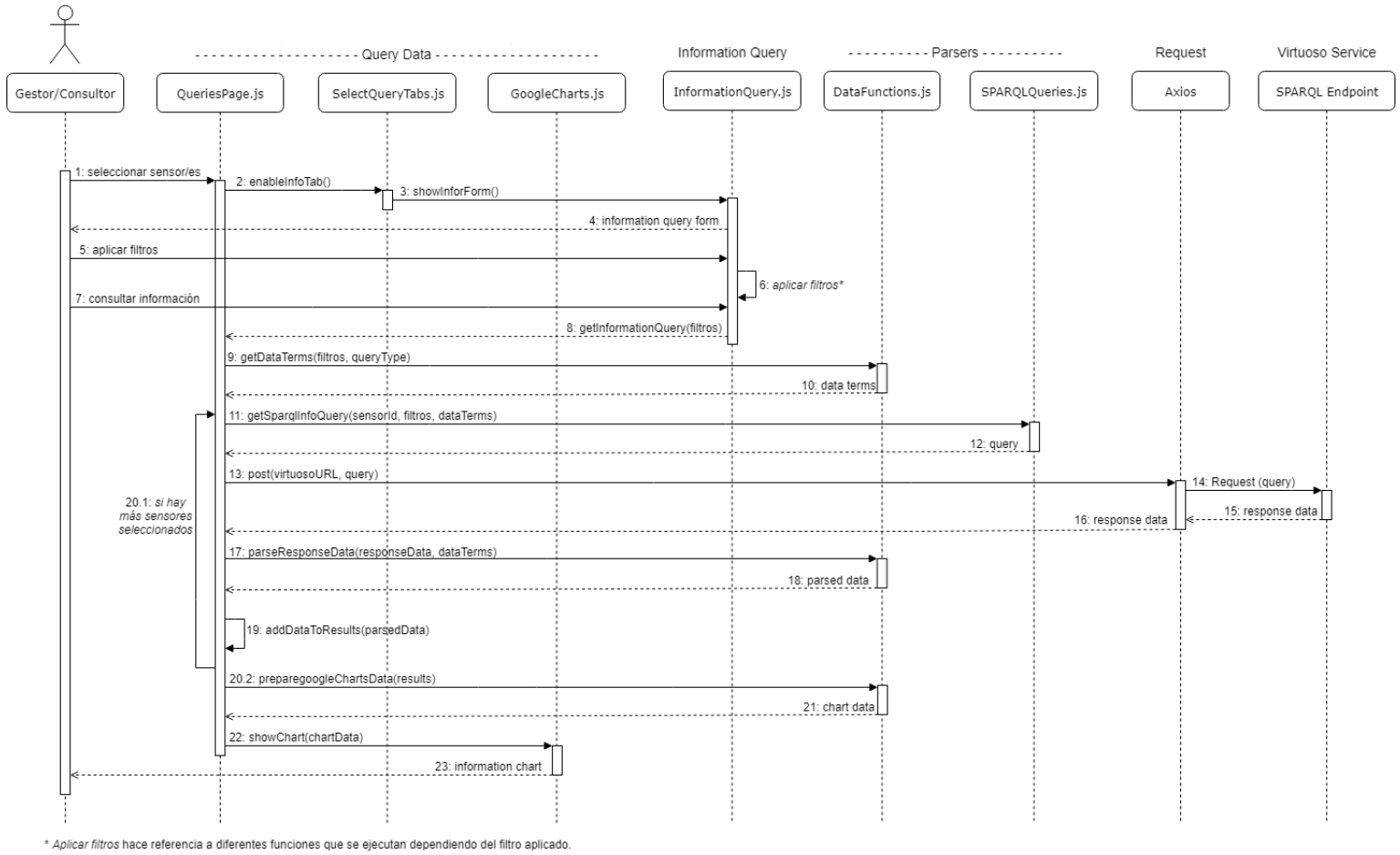

Figura A.2: Diagrama de secuencia de la funcionalidad *Consultar información general sobre cada sensor*

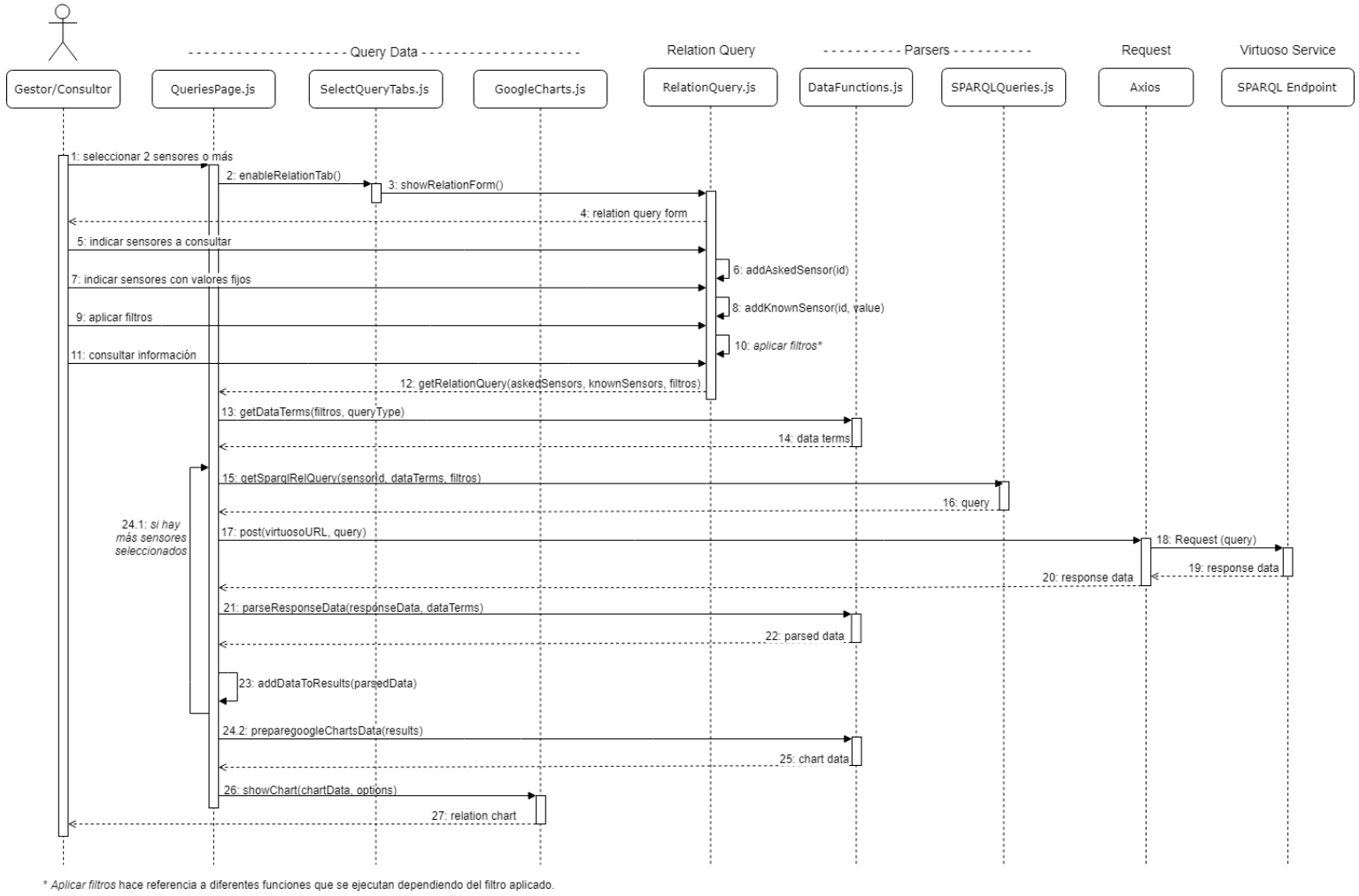

Figura A.3: Diagrama de secuencia de la funcionalidad *Consultar relación entre sensores*

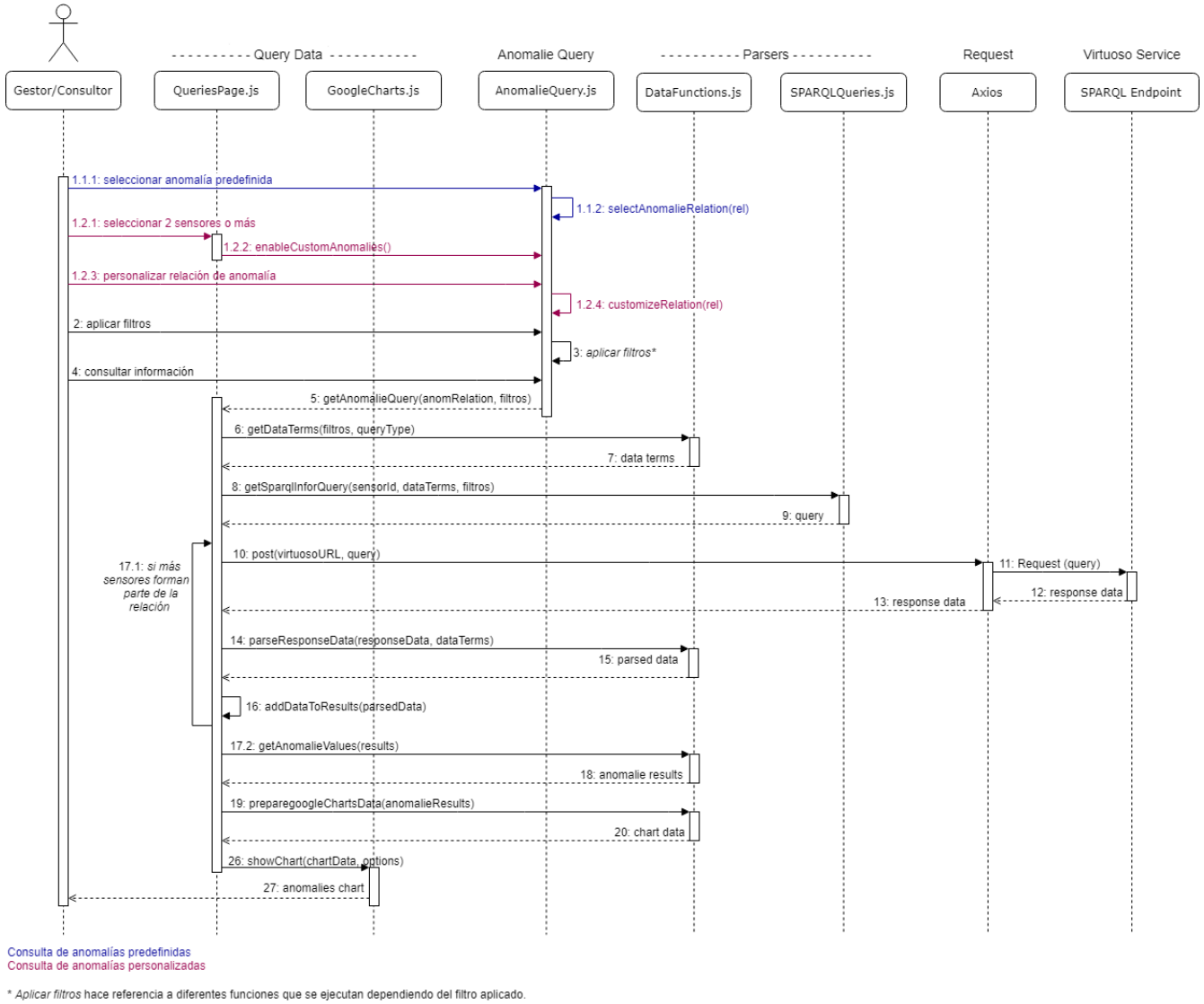

Figura A.4: Diagrama de secuencia de la funcionalidad *Consultar anomalías en sensores*

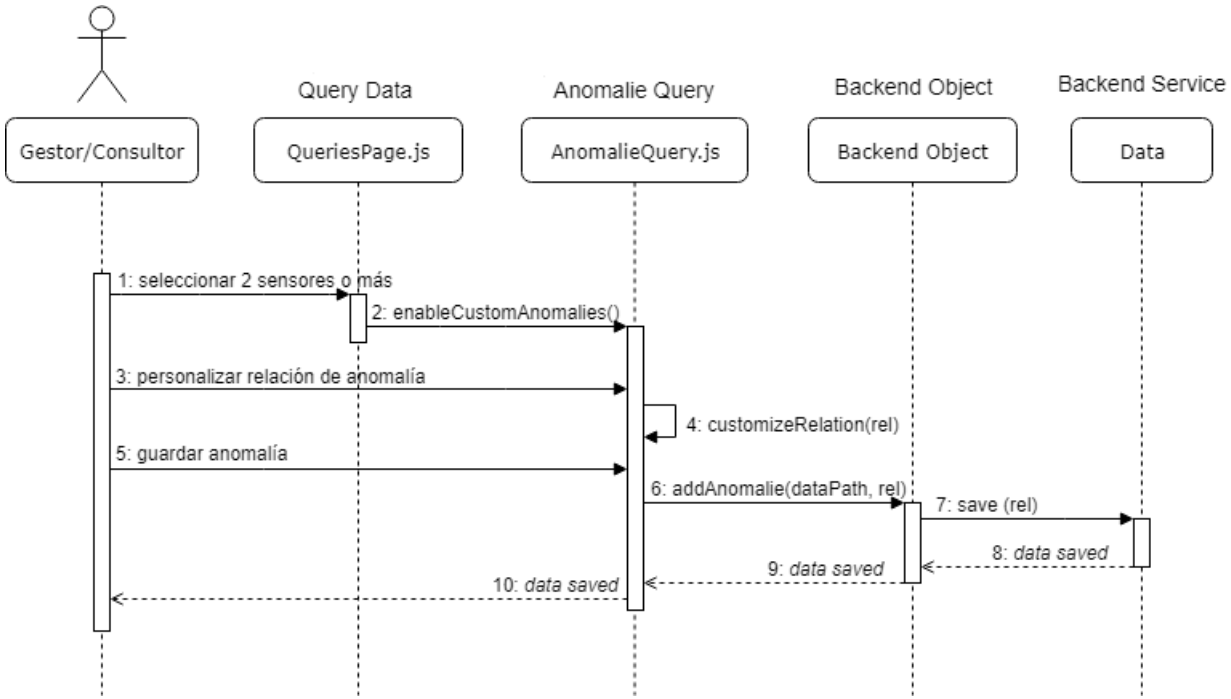

Figura A.5: Diagrama de secuencia de la funcionalidad *Añadir relación de anomalía predefinida*

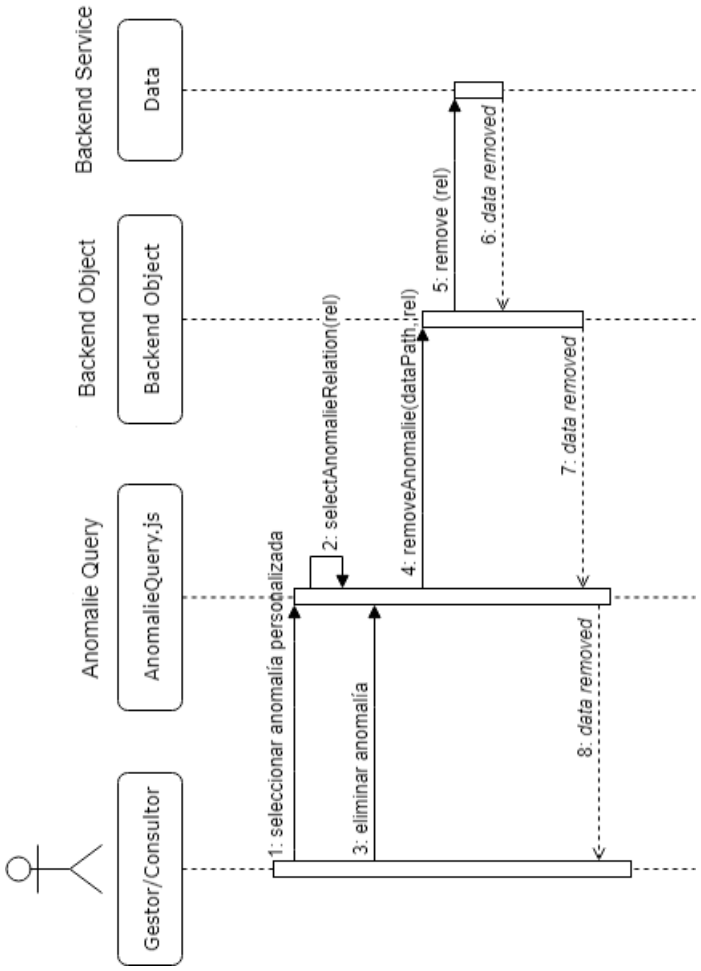

Figura A.6: Diagrama de secuencia de la funcionalidad *Eliminar relación de anomalía predefinida*

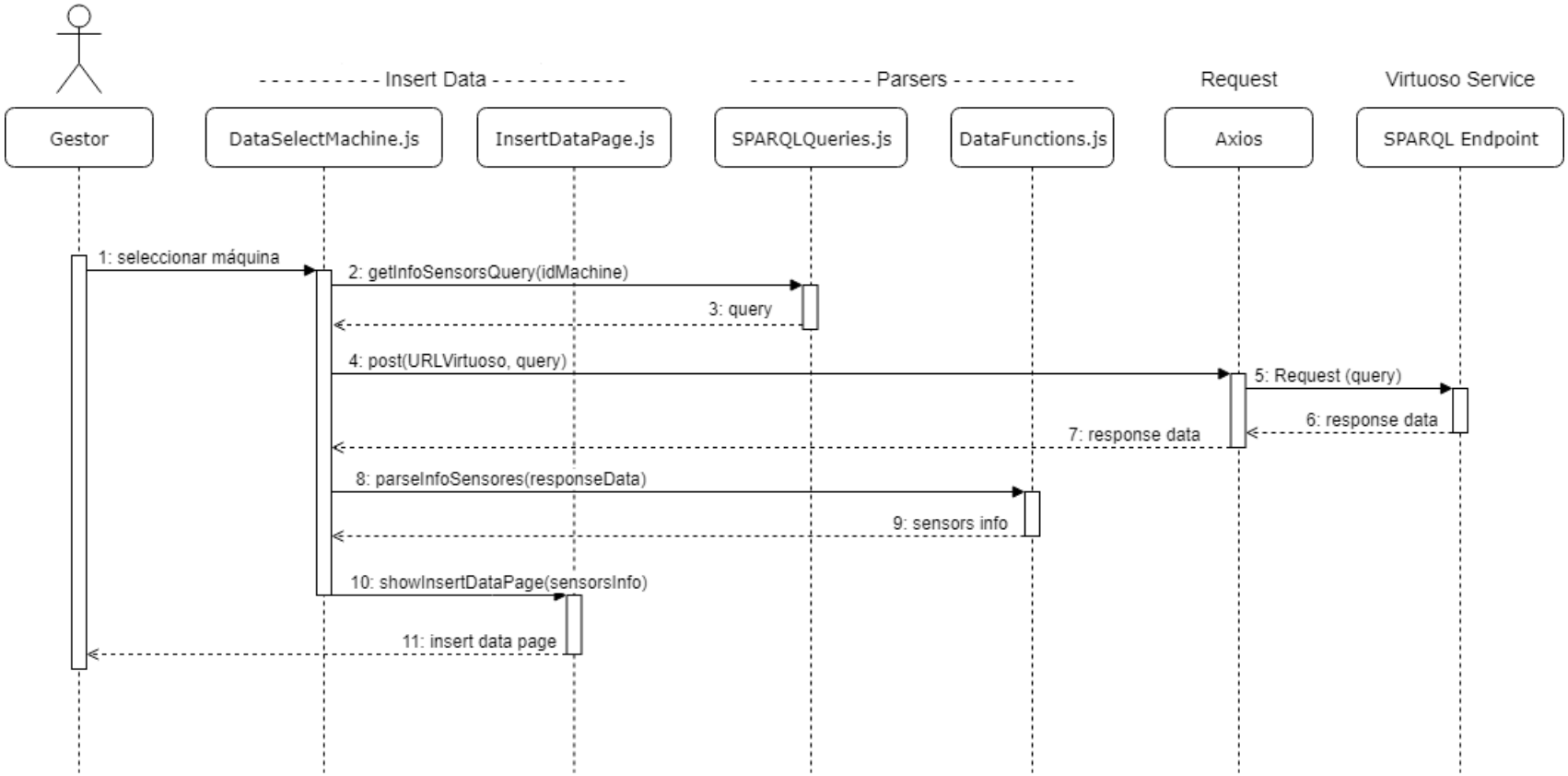

Figura A.7: Diagrama de secuencia de la funcionalidad *Insertar datos de sensores*

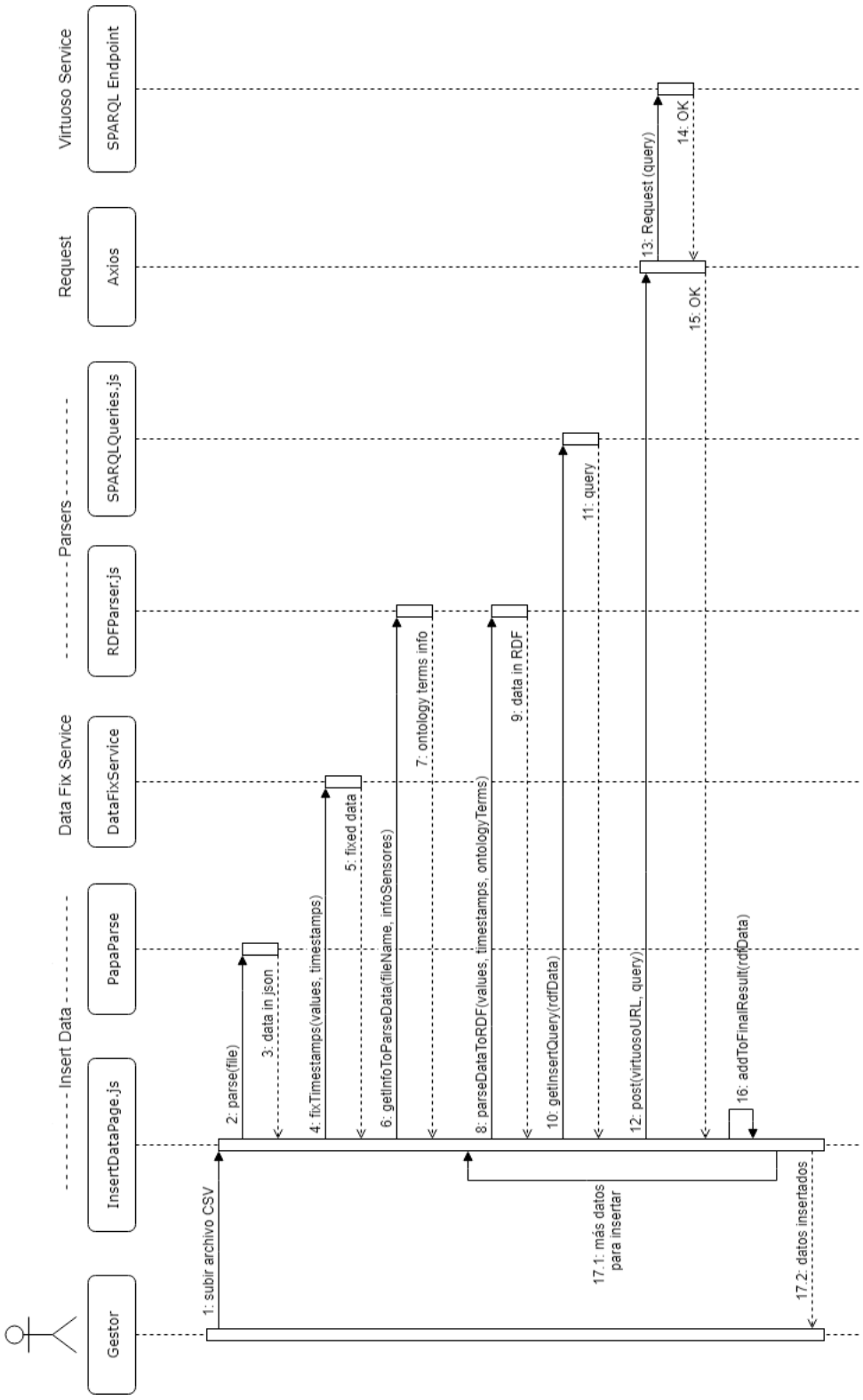

Figura A.8: Diagrama de secuencia de la funcionalidad *Subir datos a Virtuoso*

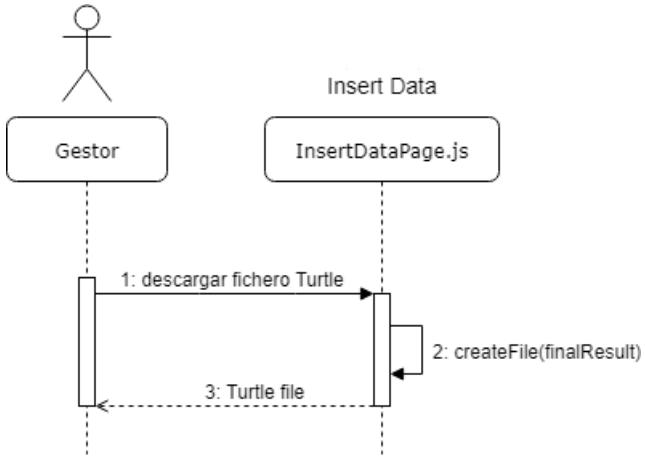

Figura A.9: Diagrama de secuencia de la funcionalidad *Descargar fichero Turtle*

Bibliografía

[Catarci et al., 1997] Catarci, T., Costabile, M., Levialdi, S., and Batini, C. (1997). Visual query systems for databases: A survey. 8:215–260.

[Cheng et al., 2016] Cheng, H., Zeng, P., Xue, L., Shi, Z., Wang, P., and Yu, H. (2016). Manufacturing ontology development based on industry 4.0 demonstration production line. In *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, pages 42–47.

[Compton et al., 2012] Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., and Taylor, K. (2012). The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25 – 32.

[Fox et al., 1995] Fox, M. S., Barbuceanu, M., and Gruninger, M. (1995). An organisation ontology for enterprise modelling: preliminary concepts for linking structure and behaviour. In *Proceedings 4th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '95)*, pages 71–81.

[Grangel-González et al., 2016] Grangel-González, I., Halilaj, L., Coskun, G., Auer, S., Collarana, D., and Hoffmeister, M. (2016). Towards a semantic administrative shell for industry 4.0 components. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pages 230–237.

[Kaufmann and Bernstein, 2007] Kaufmann, E. and Bernstein, A. (2007). How useful are natural language interfaces to the semantic web for casual end-users? In Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P.,

Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *The Semantic Web*, pages 281–294, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Lasi et al., 2014] Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., and Hoffmann, M. (2014). Industrie 4.0. *WIRTSCHAFTSINFORMATIK*, 56(4):261–264.

[Obitko and Jirkovský, 2015] Obitko, M. and Jirkovský, V. (2015). Big data semantics in industry 4.0. In Mařík, V., Schirrmann, A., Trentesaux, D., and Vrba, P., editors, *Industrial Applications of Holonic and Multi-Agent Systems*, pages 217–229, Cham. Springer International Publishing.

[Poveda-Villalón et al., 2014] Poveda-Villalón, M., Gómez-Pérez, A., and Suárez-Figueroa, M. C. (2014). Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation. *Int. J. Semant. Web Inf. Syst.*, 10(2):7–34.

[Schevers and Drogemuller, 2005] Schevers, H. and Drogemuller, R. (2005). Converting the industry foundation classes to the web ontology language. In *2005 First International Conference on Semantics, Knowledge and Grid*, pages 73–73.

[Shadbolt et al., 2006] Shadbolt, N., Berners-Lee, T., and Hall, W. (2006). The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101.

[Soylu et al., 2015] Soylu, A., Kharlamov, E., Zheleznyakov, D., Jiménez-Ruiz, E., Giese, M., and Horrocks, I. (2015). Optiquevqs: Ontology-based visual querying.

[Suárez-Figueroa et al., 2012a] Suárez-Figueroa, M. C., Gómez-Pérez, A., and Fernández-López, M. (2012a). *The NeOn Methodology for Ontology Engineering*, pages 9–34. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Suárez-Figueroa et al., 2012b] Suárez-Figueroa, M. C., Gómez-Pérez, A., and Fernández-López, M. (2012b). *The NeOn Methodology for Ontology Engineering*, pages 9–34. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Vega-Gorgojo et al., 2016a] Vega-Gorgojo, G., Giese, M., Heggestøyl, S., Soylu, A., and Waaler, A. (2016a). Pepesearch: Semantic data for the masses. *PLOS ONE*, 11(3):1–12.

[Vega-Gorgojo et al., 2016b] Vega-Gorgojo, G., Slaughter, L., Giese, M., Heggestøyl, S., Soylu, A., and Waaler, A. (2016b). Visual query interfaces for semantic datasets: An evaluation study. *Web Semantics: Science, Services and Agents on the World Wide Web*, 39:81 – 96.