Trabajo Fin de Máster
Máster Universitario en Ingeniería Aeronáutica

Gestión de ontologías e instanciación en modelos de fabricación

Autor: Alberto Gómez Alonso
Tutor: Jesús Racero Moreno
Daniel Gómez Reyes

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería

Sevilla, 2019

Trabajo Fin de Máster
Máster Universitario en Ingeniería Aeronáutica

Gestión de ontologías e instanciación en modelos de fabricación

Autor:
Alberto Gómez Alonso

Tutor:
Jesús Racero Moreno
Profesor titular
Daniel Gómez Reyes
Personal investigador

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Máster: Gestión de ontologías e instanciación en modelos de fabricación

Autor: Alberto Gómez Alonso

Tutor: Jesús Racero Moreno

Daniel Gómez Reyes

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres el regalo que me hicieron dándome posibilidad de estudiar aquello que elegí y la confianza que siempre depositan en mí.

A mi tutor, que siempre me ha atendido y ayudado cuando lo he necesitado.

Y a mis amigos en general, pero a Dani en particular que ha sido mi compañero de fatigas los últimos años.

Alberto Gómez Alonso

Sevilla, 2019

Resumen

En la actualidad, las empresas viven en un entorno cada vez más competitivo en el que no es suficiente con ser eficaz, sino que hay que ser eficiente si se quiere permanecer en el mercado. Hoy en día, el hecho de entregar un producto de calidad a tiempo empieza a estar al alcance de cualquiera, si se quiere marcar la diferencia para competir en el mercado se tiene que ser eficiente, es decir, se tiene que hacer optimizando el uso de recursos. Esto no va en relación con el número de recursos en sí, sino con cómo funcionan y si se les saca el máximo rendimiento. No importa cuántos recursos tengas. Si no los sabes usar, nunca serán suficientes.

Con el conocimiento sucede lo mismo. No es cuánto tengas, sino cómo lo usas y para qué. Está claro que es necesario adquirir conocimiento, pero de nada vale tener información que no sirve o información que sirve, pero no se sabe interpretar ni transmitir adecuadamente.

Históricamente, siempre se ha empleado mucho tiempo en el diseño de nuevos productos y sus procesos de fabricación. Además, cada vez se requieren productos más avanzados y customizados, por lo que el desarrollo se ha vuelto más complicado, pero si se quiere ser competitivo hay que ser eficiente. Hay que introducir los productos en el mercado de manera más rápida, por eso es importante tener una buena base que permita no tener que empezar de cero cada vez que se requiera desarrollar el diseño o la fabricación de un producto. Por este motivo, el hecho de extraer todo tipo de información para nuevos productos se convierte en una necesidad, pero de nada vale tener información si ésta no se almacena, transmite e interpreta correctamente.

De esto último trata la interoperabilidad, de compartir conocimiento sin que se pierda información. La interoperabilidad es clave en un sector en el que entra en juego una cadena de suministro. Cuantos más agentes entren a formar parte de la cadena más veces se tiene que compartir la información y mayor es la probabilidad de que ésta no se transmita correctamente.

El objetivo de este TFM es desarrollar una herramienta que permita la comunicación entre aplicaciones diferentes y demostrar que es posible transmitir el conocimiento entre ellas a través de ontologías.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

En la actualidad, se emplea mucho tiempo en el diseño del producto y su proceso de fabricación. Por este motivo, el hecho de extraer todo tipo de información para nuevos productos se convierte en una necesidad, pero no sólo la correspondiente al diseño o su proceso de fabricación, sino toda la información relevante relacionada con el ciclo de vida del producto.

No obstante, de nada vale tener información si ésta no se almacena, transmite e interpreta correctamente.

Pero hay más, no solo vale ser capaz de transmitir esa información entre varios departamentos dentro de la misma empresa, hay que saber transmitirla o compartirla entre diferentes empresas que usan sistemas y herramientas distintos y tener la certeza de que no se pierde información en el trasvase porque es lo que demanda la realidad actual.

Hoy en día, existe una descentralización provocada por la alta necesidad de recursos que obliga a solicitar los servicios de varias empresas externas para conseguir el producto final, dando lugar a la cadena de suministro.

La cadena de suministro está constituida por todos los agentes que forman parte del proceso de fabricación del producto, desde el proveedor de materia prima hasta aquellos que no añaden valor al producto, pero son igualmente importantes en la cadena como pueden ser los transportistas.

El propietario del producto puede subcontratar, no sólo su fabricación sino también su diseño. Aquí entra en juego un nuevo concepto, el de **ingeniería concurrente**.

Es muy importante realizar concurrencias para diseñar productos fabricables, de nada sirve tener una buena idea que no se puede hacer realidad.

Esto implica que la colaboración que antes podía realizarse entre los departamentos de diseño y fabricación de la misma empresa ahora vaya más allá y obligue a realizar esa misma concurrencia entre varias empresas añadiendo un grado de complejidad.

La complejidad está ligada al número de participantes en el flujo de fabricación y, consecuentemente, al número

de veces que se tiene que intercambiar información entre ellos. Por este motivo, cobra tanta importancia gestionar dicho intercambio.

Por ejemplo, en la industria aeronáutica un caso como este entraría dentro de la normalidad:

La empresa A desarrolla un producto propio, subcontrata el diseño a la empresa B y la fabricación a la empresa C. Este hecho, implica una concurrencia vital entre las empresas B y C, pero la empresa C no se dedica a fabricar, es especialista en montaje. Esta empresa C va a subcontratar la fabricación de todas las piezas. Como se dedica a esto, desde el departamento de ingeniería de subcontratación de la empresa C serán capaces de realizar esa concurrencia con la empresa B, pero para ello deben tener un conocimiento exhaustivo de su cadena de suministro. No obstante, habrá geometrías o tolerancias que sean difíciles de conseguir y sea necesario incluir en esa concurrencia a la empresa D a la que se le vaya a subcontratar la fabricación de una pieza concreta. En resumen, es fácil tener a 4 empresas independientes participando en el desarrollo del producto y entre todas ellas, debe fluir la información de manera total.

Pero no queda ahí la cosa, porque en lo que se refiere a la cadena de suministro, la empresa C repartirá piezas de mecanizado entre varias empresas "D" (además del precio y el leadtime, que dependerá de la capacidad de industrialización de cada proveedor, hay que valorar la experiencia que pueda tener cada uno con determinados materiales o métodos de arranque de viruta que no sean a través de máquinas de CNC de 3 y 5 ejes, como puede ser el torno) y varias empresas "E" especialistas en perfiles y chapas. Pero a su vez, estas empresas subcontratarán tratamientos térmicos o procesos finales comunes a otros proveedores "F", pero también habrá otros procesos finales no tan comunes que habrá que subcontratarlos a empresas "G". Y, para terminar, la empresa C tendrá que hacer montajes de subconjuntos intermedios para los que no tendrá medios o no estará certificado como puede ser la instalación de una rótula, dando entrada en la cadena de suministro a una empresa "H".

Y esto es sólo para metálicas, lo mismo pasaría con piezas de material compuesto, tuberías, sellos...

A todas estas empresas que participan en el proceso de fabricación hay que proporcionarles la documentación asociada a cada pieza y es indispensable que reciban esa información de manera completa porque de lo contrario podrían fabricar una pieza que no cumpliera todos los requerimientos solicitados de modo que se rechazara la compra y, consecuentemente, se tuviera que retrabajar o fabricar de nuevo provocando un sobrecoste al proveedor y un retraso al cliente.

Cuando se requiere un intercambio de información entre dos personas o entidades, es imprescindible que ambas se entiendan entre sí y, además, sepan de lo que hablan. Por ejemplo:

- Dos personas que hablan el mismo idioma y se mueven el mismo ámbito, podrán intercambiarse información sin problema.
- Dos personas que hablan el mismo idioma, pero se mueven en un ámbito diferente, probablemente tendrán que aclarar el significado de algunos conceptos antes de poder hablar sobre ellos, ya sea por desconocimiento o porque al estar en ámbitos diferentes para cada uno puede ser distinto.
- Dos personas que hablan distintos idiomas, pero se mueven el mismo ámbito, tendrán que buscar un idioma común para comunicarse.
- Dos personas que hablan distintos idiomas y, además, se mueven en ámbitos diferentes, primero tendrán que buscar un idioma común y después, aclarar el significado de algunos conceptos.

En estos cuatro ejemplos, se ve como el intercambio de información entre personas lleva asociada una dificultad variable, que va desde un extremo en el que la dificulta puede ser inapreciable hasta otro en el que dicha dificultad puede ser tan alta que imposibilite el intercambio de información.

Con las aplicaciones pasaría lo mismo que con las personas, cada aplicación almacena los datos para sí misma, es decir, cada una los recibe, controla y emite como le conviene. Por este motivo, las aplicaciones no se hablan entre sí y, como consecuencia, se crean problemas de interoperabilidad entre ellas.

El objetivo de este TFM es desarrollar una herramienta que permita **la comunicación entre aplicaciones** diferentes a través de **ontologías** y demostrar que, haciendo uso de una sintaxis común, es posible relacionar elementos de distintas aplicaciones que tienen una **misma semántica**. Además, con las ontologías el usuario define su propio dominio y lo que considera de interés, por lo que a la hora de transmitir la información únicamente se envía y recibe lo que se quiere sin que se pueda producir una pérdida de significado.

La **interoperabilidad** es la habilidad de compartir datos, información y conocimiento de forma fluida y sin errores entre dos o más herramientas con la mínima intervención manual.[i] Favorecer la interoperabilidad evita la pérdida de información y la ineficiencia que esto conlleva.

La siguiente figura muestra la arquitectura básica a desarrollar y validar en este proyecto. La definición de ontologías permite a los usuarios definir una estructura semántica común, donde todo o parte de la información se comparte. La definición de la ontología queda expresada de forma colaborativa, es decir se amplía y gestiona en base a las necesidades de los sistemas a interoperar.

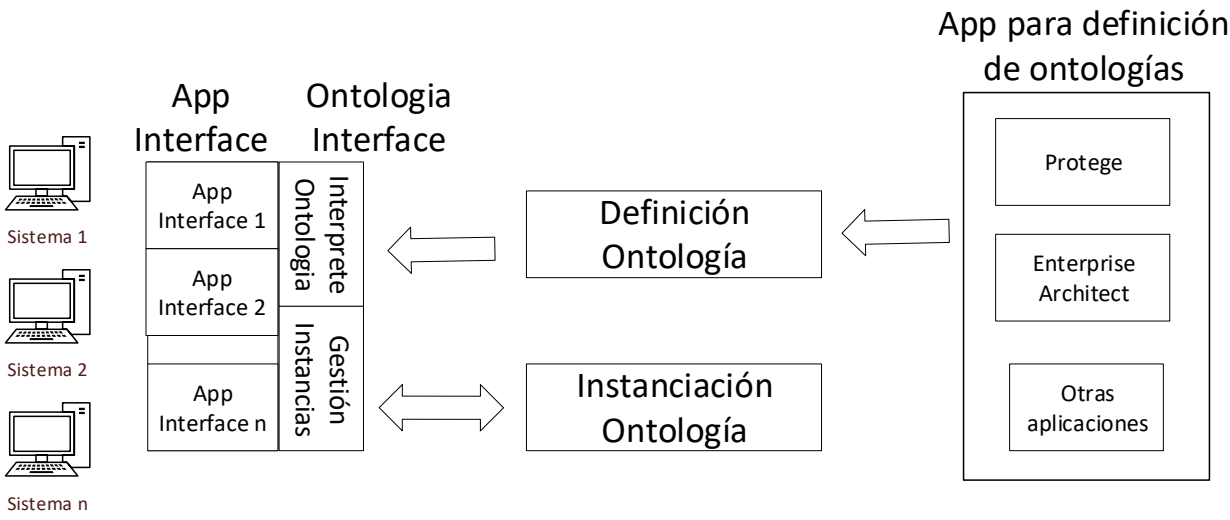

Figura 1.1 Arquitectura básica de instanciación de ontologías

El intercambio de información se realiza a través de la instanciación, una instanciación asociada a un sistema genera e importa los datos en base a las especificaciones de la ontología y la semática propia del sistema. Este proceso es posible gracias a las interfaces genéricas de gestión de ontologías que procesa la instancia. Finalmente, las interfaces (de los sistemas propietarios) conectan los datos de la instancia con las del sistema.

Para la consecución del objetivo es necesario desagregarlo en objetivos parciales, que se iran desarrollando durante el trabajo. Los objetivos parciales son:

- Revisión de la literatura sobre interoperabilidad y técnicas aplicadas para el intercambio de información entre sistemas.
- Revisión de la literatura sobre ontologías, descripción de herramientas y métodos de definición. Esta revisión se complementa analizando ontologías sobre gestión de datos de producto y fabricación.
- Definición de la ontología de ensamblado, basada en un modelo de datos de una empresa aeronaútica.
- Diseño y desarrollo de la arquitectura descrita en la figura anterior.
- Desarrollo de un caso de uso de interoperabilidad entre un sistema PLM (Aras Innovator) y herramientas de instanciación.

El documento está dividido en tres bloques, en el primer bloque se centra en estudiar el estado de la tecnología, centrándose en un análisis sobre la interoperabilidad en sistemas PLM y en concreto se analiza el uso de ontologías como mecanismo de intercambio de información entre sistemas. Este bloque finaliza describiendo ontologías enmarcadas en la gestión de datos de producto y sistemas PLM.

El segundo bloque, capítulo 5 y 6, se centra en el desarrollo de la arquitectura, la arquitectura pemite definir los procesos de definición e interfaces de comunicación entre sistemas y aplicaciones de diseño. Este bloque define el modelo de datos de un ejemplo de ensamblado. El diseño del sistema es decrito mediante metodología orientada a objetos UML (Unified Modeling Language).

El tercer bloque, correspondiente con el capítulo 7 y 8 define un caso de uso demostrativo donde se implementa la ontología de ensamblado definida en el capítulo 6.

2 SISTEMAS PLM E INTEROPERABILIDAD

En los últimos años, la colaboración transversal en toda la cadena de suministro y diseño de producto ha cobrado una mayor importancia siendo un aspecto clave en la investigación para el desarrollo de productos y procesos más flexibles.

La ingeniería colaborativa se caracteriza por proveer conceptos, soluciones, así como tecnologías para el desarrollo de productos integrando diversos equipos de ingeniería, procesos, fabricación, mantenimiento… Así mismo, la ingeniería colaborativa permite que diferentes equipos, de diversos campos, trabajen conjuntamente intercambiando información y recursos mejorando los tiempos de diseño y fabricación.

Pero, diferentes actores colaborando entre si conlleva el uso de una gran diversidad de sistemas y aplicaciones, que en muchos casos no son compatibles o no disponen de mecanismos de intercambio de información. La variedad de equipos y expertos que intervienen en el ciclo de vida del producto crea la necesidad de disponer de protocolos de intercambio de información requiriendo de métodos y herramientas para compartir e integrar el conocimiento en el propio producto (Penciuc, 2014).[ii]

2.1 Sistemas PLM

Los sistemas PLM, se definen como "un enfoque comercial estratégico que aplica un conjunto consistente de Soluciones empresariales en apoyo de la creación, gestión, difusión y uso colaborativo de información de definición de producto en toda la empresa extendida desde el concepto hasta el final de la vida útil - integrando personas, procesos, sistemas comerciales e información" (Amann, 2002).[iii] Los sistemas PLM se han extendido ampliamente en la última década y la interoperabilidad entre software propietario se ha convertido en una necesidad. Esto unido al concepto de empresa extendida, ha provocado el auge de los sistemas PLM como integrador del conocimiento.

Los sistemas PLM representan una visión integral de toda la información relativa al producto a lo largo de su ciclo de vida. Es un repositorio de toda la información que afecta a un producto y un medio de comunicación entre las partes interesadas en dicho producto.[iv]

Los conceptos de PLM fueron introducidos originalmente en sectores en los que la seguridad y el control siempre han sido factores determinantes, como es el caso de la industria aeroespacial. Estos sectores implementaron la disciplina de la gestión de la configuración (CM), que evolucionó a sistemas electrónicos de gestión de datos (EDMS), y, posteriormente, a gestión de datos del producto (PDM).

El entorno empresarial está evolucionando y se está volviendo cada vez más competitivo. En el mundo de la fabricación el coste se mide en tiempo, por eso cualquier ahorro se centra en una reducción del tiempo asociado a una operación.

Por este motivo, si se quiere ser más competitivo hay que ser más eficiente. Hay que introducir los productos en el mercado de manera más rápida, por eso es importante tener una buena base que permita no tener que empezar de cero cada vez que se requiera desarrollar el diseño o la fabricación de un producto. Además, cada vez se requieren productos más avanzados y customizados, o lo que es lo mismo, más complejos y esa complejidad conlleva que las empresas tengan que ser flexibles y capaces de adaptarse a los cambios con agilidad.

La gestión del ciclo de vida del producto (PLM) se puede definir como el proceso de gestión del ciclo de vida completo de un producto, incluyendo su diseño, fabricación, comercialización, mantenimiento y eliminación.

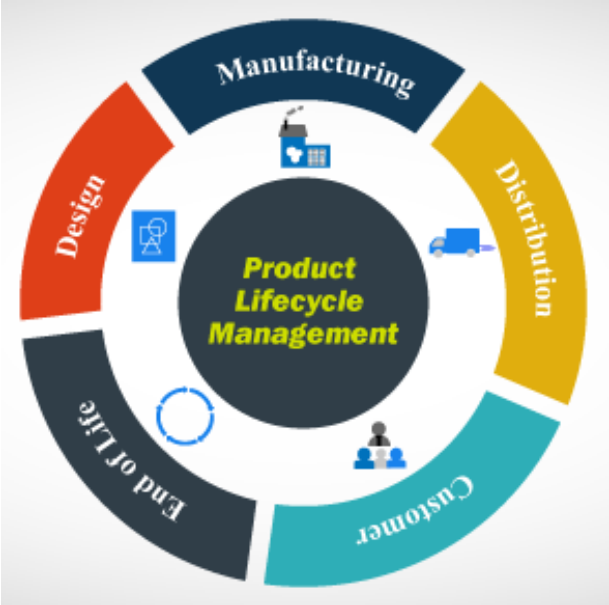

Figura 2.1 Ciclo de vida del producto

El sistema PLM permite tener un registro centralizado de todos los datos del producto que sin él estarían descentralizados, repartidos entre varios departamentos de la empresa o, incluso, entre varias empresas. Hechos como este, han provocado el auge de los sistemas PLM como integradores del conocimiento.

Con un sistema PLM, las organizaciones pueden gestionar decisiones clave como cambios del producto a tiempo real. Además, con un sistema PLM, los fabricantes sólo tienen acceso a la única versión activa del producto en ese momento, pero la compañía tiene acceso al registro de cambios existentes entre cada una de las revisiones del producto. [v]

2.2 Interoperabilidad

Damjanović (2007) describe los principales retos que debe acometer la ingeniería colaborativa agrupándolo en tres grupos, tecnológico, social y económico-organizacional. Este ya describe en su trabajo la necesidad de definir una ontología colaborativa centrada en procesos colaborativos de mecatrónica, destacando la definición de ontologías con el objetivo de fomentar la interoperabilidad entre áreas.[vi]

El desarrollo de espacios compartidos basado en sistemas informáticos ha sido una de las principales orientaciones en el desarrollo de productos de forma colaborativa. La ingeniería colaborativa no solo se centra en el diseño del producto, sino que abarca todas las etapas del ciclo de vida, incluyendo el concepto de iDMU

(Industrialization Mock Up). Este nuevo paradigma precisa de nuevas metodologías de trabajo y herramientas PLM colaborativas (Mas, 2013).[vii]

La interoperabilidad es la capacidad de los sistemas, ya sean idénticos o radicalmente diferentes, para comunicarse sin ambigüedad y operar conjuntamente.[viii] En general, los autores distinguen tres enfoques para solventar los problemas de interoperabilidad: ontologías y web semántica, definición de estándares de comunicación y creación de servicios web (Penciuc, 2011).[ix]

- Ontologías y web semántica

La utilización de ontologías y web semántica permite mapear los datos entre softwares de diferentes propietarios a nivel conceptual. Siendo una técnica reciente, precisa que los propietarios del software se adapten a diseños ontológicos abiertos.

En la literatura existen estudios que implementan diferentes enfoques para el diseño de productos en ontologías como estándar para el intercambio de datos entre el diseño y otras actividades de ingeniería en tareas colaborativas.

En el campo PLM se propone ONTO-PLM como un modelo básico común para proporcionar una solución de interoperabilidad entre los datos del producto (encapsulados en PLM) y las aplicaciones empresariales que los administrarán, como ERP, CAD y MES (Panetto, 2012).[x] El marco conceptual consiste en la conceptualización de las normas existentes, principalmente ISO 10303 e IEC 62264, relacionadas con el modelado de datos técnicos del producto. Por ejemplo, el concepto de modelo de producto genérico es utilizado por Mun (2008) para el desarrollo de un modelo de datos neutral que respalde un almacén de datos que esté disponible como interfaz entre una base de datos de productos y un sistema ERP a lo largo del ciclo de vida de la planta de energía nuclear.[xi]

- Estandarización de intercambio de datos

Los métodos de estandarización se caracterizan por definir un lenguaje estandarizado para el intercambio de información, en la literatura se han definido procesos estandarizados. Entre los principales lenguajes se encuentra DXF para el intercambio de información CAD y en el ámbito de la definición de procesos destacan lenguajes como STEP. El Standard para intercambio de datos de producto (Standard Exchange of Product Model Data), es un estándar ISO que describe como representar e intercambiar datos de producto. STEP se ha convertido en la principal referencia en la especificación de datos de producto y como método de intercambio datos entre sistemas PLM (ISO, 1994, Mason, 2002).[xii y xiii]

El papel global y principal de los estándares de intercambio de información es reducir el número de protocolos de intercambio de una multitud inmanejable de intercambios uno a uno a un número finito de composiciones distintas y significativas de información coherente a través del tiempo, el espacio y múltiples disciplinas (Rachuri, 2008).[xiv]

Aunque el proceso de estandarización quedó fijado hace tiempo, la investigación se centra en la definición y automatización de herramientas para la definición de estándares, que por ejemplo no solo incluya productos sino también procesos. Por ejemplo, Padillo (2017) describe un procedimiento para definir mediante artefactos UML (Lenguaje de modelado Unificado) estructura de intercambio de información. Este método emplea como base un lenguaje de intercambio de datos denominado EXPRESS.[xv]

Los métodos de estandarización entre sistemas ERP y MES también han sido estudiado en la literatura. El método ISA-95 se define como el estándar internacional para la integración de sistemas empresariales y de control. Consiste en modelos y terminología que pueden usarse para determinar qué información debe intercambiarse entre sistemas de ventas, finanzas, logística, producción, mantenimiento y calidad.[xvi]

El problema de la estandarización en sistemas PLM ha cobrado gran importancia en empresas del sector de fabricación donde la cantidad y variedad de proveedores implica una mayor colaboración e intercambio de información. Esto ha obligado a empresas como AIRBUS a trabajar en la estandarización de la información en sistemas PLM, la propia empresa ha creado un grupo de estandarización al igual que se creado en otros ámbitos. Este grupo ha creado varios programas y proyectos de estandarización centrado en la normativa AP-242 (Managed Model based 3D Engineering) y AP-239 (Figay, 2015).[xvii]

Para terminar esta revisión es importante referenciar al MBSE (Model Based System engineering). La ingeniería

de sistemas basada en modelos es la aplicación formal de varios niveles de modelado (de 0D a 3D) para evaluar los requisitos del sistema, diseño, análisis, verificación y actividades de validación que comienzan en la fase de diseño conceptual y continúan durante todo el desarrollo y las fases posteriores del ciclo de vida. En su forma más directa, MBSE aplica un paradigma de modelado continuo (0D, 1D, 2D, 3D ...) para definir sistemas, pasando de la forma más simple (0D) a una representación 3D completamente definida, y a modelos de orden superior para comprender los problemas temporales.

Esto se realiza, además, en el contexto de los requisitos escritos y los diseños CAD 2D y 3D. Los modelos se utilizan para validar desde etapas muy tempranas que el sistema funcionará según lo concebido y definido por sus requisitos. Principalmente MBSE emplea como lenguaje SYSML, lenguaje que ofrece soporte para el detalle, evaluación, plan, verificación y aprobación de una amplia variedad de sistemas complejos.

- Servicios web

Los servicios web se han constituido como una alternativa en el intercambio de datos entre sistemas. En la última década el open data se ha vuelto muy popular, con el objetivo de poner a disposición de cualquiera la información facilitando el intercambio de datos.

La implantación de open data se extiende al ámbito de la interoperabilidad, con la salvedad de que son las empresas propietarias las que deciden la información que proporcionan. Los servicios web se han convertido en la herramienta empleada como vía de solicitud e intercambio de información.

En este ámbito se están desarrollando investigaciones donde la estandarización de los servicios y API (Application Interface) garantiza una estandarización.[xviii] La especificación de los Servicios PLM 1.0 define un Modelo Independiente de Plataforma (PIM) para los Servicios de Gestión del Ciclo de Vida del Producto. Esta especificación define un Modelo Específico de Plataforma (PSM) aplicable a una implementación de servicios web definida por una especificación WSDL, con un enlace SOAP y una especificación de esquema XML (Lukas, 2005).[xix] Ejemplo de ello son los sistemas PLM como ARAS que, a través de un servicio web, permite acceder y ampliar la información del sistema. Este nuevo enfoque permite una customización del propio PLM desde el webservice.

En el campo PLM, los "Habilitadores PLG OMG" basados en tecnologías de middleware y los "Servicios PLM" son tecnologías web desarrolladas para soportar la comunicación entre sistemas PLM y entre PLM y otras aplicaciones CAX (Wang, 2009).[xx]

Por ejemplo, en Choi (2010), un archivo neutral basado en XML se define al referirse a los servicios PLM para la construcción del integrador PLM como software que puede intercambiar información de productos, procesos y recursos entre sistemas PLM comerciales.[xxi] El integrador PLM consiste en el adaptador XML y el adaptador PLM para extraer información PPR (Recursos de Proceso del Producto) de un sistema comercial PLM o PDM, convertirlos en archivos XML y también, importar toda esta información PPR en otro sistema PLM o PDM. El estándar de servicios PLM es utilizado en otro trabajo por Gunpina (2008) para soportar el intercambio de datos entre dos sistemas PDM diferentes a través de Internet.[xxii] Se realiza una implementación para soportar la interoperabilidad entre dos PDM comerciales: SmartPDM y DynaPDM. En esta arquitectura, el módulo de transferencia de datos traduce los datos del producto dentro del sistema PDM al servidor de referencia de servicios PLM. El módulo de transferencia de datos captura los datos del sistema PDM mediante el uso de API y los traduce al formato que la implementación de referencia de los Servicios PLM puede leer y visualizar. Cualquier cliente XPDI permitido de Servicios PLM desde el lado remoto puede acceder a los datos PDM traducidos en el servidor de Servicios PLM. Otra forma universal de apoyar la colaboración entre empresas a lo largo del ciclo de vida del producto es enfocarse en las interacciones a nivel de flujo de trabajo para administrar el mecanismo de interoperabilidad del servicio web.

Por ejemplo, Jiang (2009) ha desarrollado un novedoso enfoque combinado de vista de proceso basado en redes de Petri y servicios web para permitir una mejor colaboración entre empresas a lo largo del proceso de desarrollo de productos.[xxiii]

En este enfoque, los modelos de flujo de trabajo individuales de las empresas participantes se asignan primero a los modelos de flujo de trabajo de vista de proceso. Luego, estos modelos de flujo de trabajo de vista de proceso se representan en el lenguaje de descripción de servicios web (WSDL).

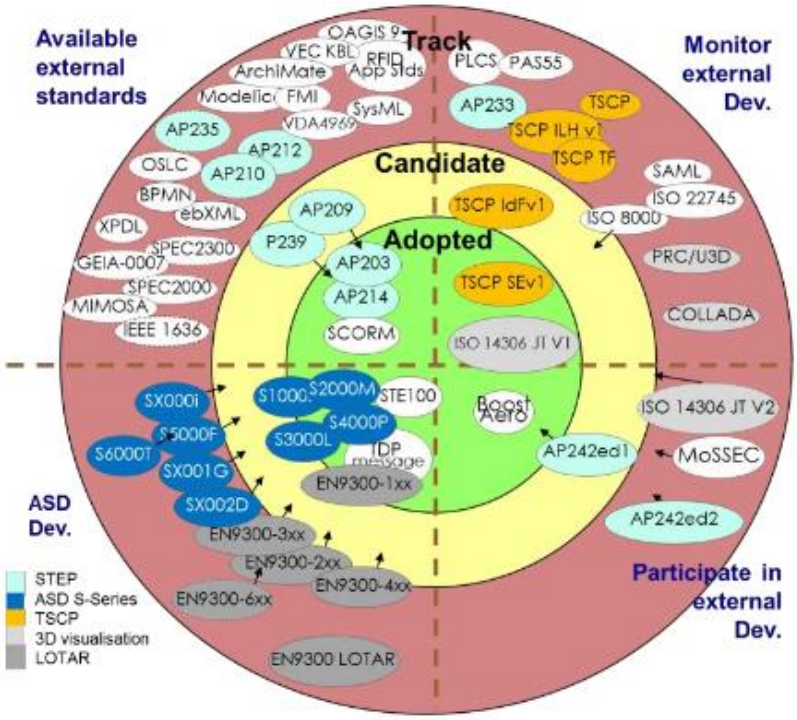

Figura 2.2 Evolución de las normas de estandarización

2.3 ISO 10303

El ISO 10303 denominado **integración y sistemas de automatización industrial** – intercambio y representación de datos del producto, también conocido como STEP (Standard for the Exchange of Product Model Data) es un estándar creado para describir las características asociadas a un producto a lo largo de todo su ciclo de vida, con independencia de los sistemas que puedan intervenir.

La necesidad de normas para intercambiar datos fue originalmente reconocida en los años 70's y condujo al desarrollo de especificaciones como el Graphics Inicial Exchange Specification (IGES) por el departamento de defensa de los Estados Unidos, un archivo informático que define un formato neutral de datos que permite el intercambio digital de información entre sistemas de diseño asistido por ordenador (CAD).

De manera similar, en Francia nació el D'Exchange Estándar et de Transfert, conocido como SET, y en Alemania el Automobilindustrie-Flachen-Schnittstelle Verband der (VDA-FS). Todas estas normas prescriben el uso de formatos estándar de archivos para el intercambio de datos.

A mediados de los años 80's las necesidades de las empresas hicieron que éstas se inclinaran a favor de una norma internacional que no solo mejorara las especificaciones del producto sino todo lo relacionado con su ciclo de vida dando lugar al formato STEP.

El proyecto STEP se inició en 1984, con los siguientes objetivos.

- La creación de un estándar internacional que cubriera todos los aspectos de datos relacionados con los sistemas CAD/CAM.
- La implementación y aceptación de este estándar en la industria reemplazando diversas normas y especificaciones nacionales.
- La estandarización de un mecanismo para describir datos del producto a lo largo de todo su ciclo de vida con independencia de los sistemas.
- La separación de la descripción de datos del producto de su implementación.

Un total de doce partes de STEP fueron publicado por la organización internacional de normalización (ISO) a inicios de 1995. Además de estas doce partes, más de sesenta partes adicionales del estándar fueron

desarrollándose por aquel entonces. Desde su inicio, no sólo ha sido ampliado, sino que además ha ido modificándose en función de las necesidades de la industria.

En 1984, los requisitos cruciales fueron la creación de un estándar que pudiera reemplazar los formatos IGES, SET, VDA-FS, etc. presentándose como mejor alternativa en términos de efectividad y eficiencia para el intercambio de datos del CAD/CAM. Hoy día, STEP está expandiéndose para reconducir otras necesidades de la industria como el manejo y control de los datos del producto en su ciclo vida completo, con independencia de los sistemas.

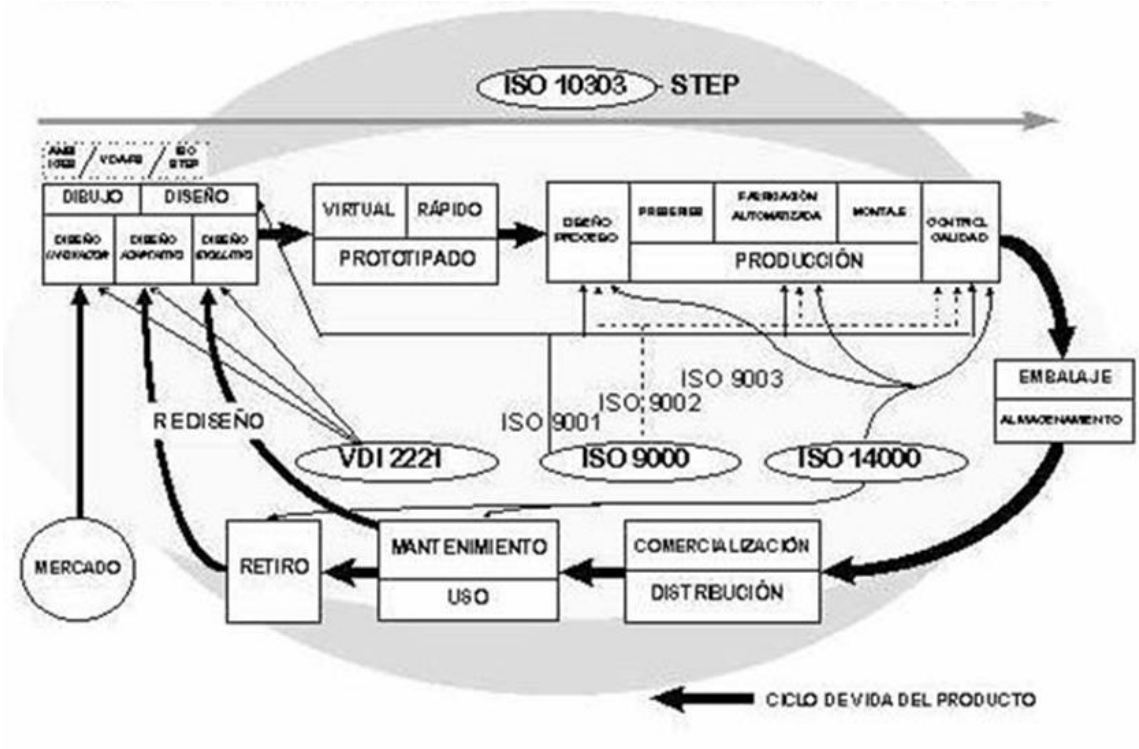

Figura 2.3 Estandarizción del ciclo de vida del producto

3 INTRODUCCIÓN A LAS ONTOLOGÍAS

La **Web Semántica** no es más que una red de datos con un marco común que permite que dichos datos se compartan y reutilicen a través de los límites de la aplicación. Para hacerla posible son necesarias dos cosas:

- Formatos comunes para la integración y combinación de datos extraídos de diversas fuentes.
- Lenguaje para registrar cómo se relacionan los datos con los objetos del mundo real permitiendo que una persona o una máquina comience en una base de datos y luego se mueva a través de un conjunto interminable de bases de datos que están conectadas, no por cables, sino por ser prácticamente lo mismo.

El Lenguaje de Ontologías Web (OWL)[xxiv, xxv, xxvi y xxvii] es un lenguaje web semántico diseñado para transmitir conocimiento sobre un dominio compuesto de elementos que puede ser interpretado tanto por humanos como por máquinas sin ambigüedades. OWL es un lenguaje computacional basado en la lógica que permite, entre otras cosas, hacer explícito el conocimiento implícito. Los documentos de OWL, conocidos como ontologías, pueden publicarse en la World Wide Web y pueden referirse a otras ontologías de OWL.

El OWL es un lenguaje con un significado formalmente definido. Las ontologías proporcionan **clases**, **propiedades**, **individuos** y **valores de datos** y se almacenan como documentos web semánticos, es decir, son vocabularios de términos formalizados, que a menudo cubren un dominio específico y son compartidos por una comunidad de usuarios.

El OWL está diseñado para ser usado en aplicaciones que no sólo representan información para los usuarios, sino que, además, necesitan procesar su contenido. OWL facilita un mejor mecanismo de interpretabilidad de contenido Web que los mecanismos admitidos por XML, RDF, y RDF Schema (RDF-S) proporcionando vocabulario adicional junto con una semántica formal.

La Web semántica es una visión del futuro de la Web donde la información está dando un significado explícito, permitiendo que las máquinas puedan procesar e integrar automáticamente la información disponible en la Web. La Web semántica se basará en la capacidad de XML para definir esquemas de etiquetas a medida y en el enfoque flexible de RDF para representar datos. El primer nivel requerido por encima de RDF para la Web semántica es un lenguaje de ontologías que pueda describir formalmente el significado de la terminología usada en los documentos Web. Si se espera que las máquinas hagan tareas útiles de razonamiento sobre estos

documentos, el lenguaje debe ir más allá de las semánticas básicas del RDF Schema.

OWL ha sido diseñado para cubrir esta necesidad de un lenguaje de ontologías Web. OWL forma parte de un conjunto creciente de recomendaciones del W3C relacionadas con la Web semántica.

- XML proporciona una sintaxis superficial para documentos estructurados, pero no impone restricciones semánticas en el significado de estos documentos.
- XML Schema es un lenguaje que se utiliza para restringir la estructura de los documentos XML, además de para ampliar XML con tipos de datos.
- RDF es un modelo de datos para objetos ("recursos") y relaciones entre ellos, proporcionando una semántica simple para éste. Este tipo de modelo de datos puede ser representado en una sintaxis XML.
- RDF Schema es un vocabulario utilizado para describir propiedades y clases de recursos RDF, con una semántica para la generalización y jerarquización tanto de propiedades como de clases.
- OWL añade más vocabulario para describir propiedades y clases: entre otros, relaciones entre clases, cardinalidad, igualdad, más tipos de propiedades, características de propiedades y clases enumeradas.

3.1 RDF y RDF-S

RDF Schema[xxviii] [y xxix] es una extensión de RDF[xxx] que proporciona mecanismos para describir clases, que pueden estar relacionadas entre sí, y relaciones entre dichas clases.

El sistema de clases y propiedades del RDF Schema es similar a los tipos de sistemas de lenguajes de programación orientados a objetos, pero difiere de ellos en que en vez de definir una clase en términos de las propiedades que puedan tener sus instancias, describe las propiedades en función de las clases a las que definen.

De este modo, es sencillo para un usuario modificar una ontología existente añadiendo o quitando propiedades de las clases definidas según le interese. Quizás haya una clase definida con dos atributos, pero el usuario quiera definirla con tres, en cuyo caso sólo tendría que añadir esa propiedad con un dominio que fuera la clase en cuestión o quizás simplemente quiera hacer la ontología compatible para dos softwares distintos que leen la misma propiedad, pero con distinto nombre, en cuyo caso sólo tendría que duplicarla y nombrarla de las dos formas para que ambas herramientas, leyeran únicamente la que reconocen, pero terminen interpretando la misma información.

Cuando las personas se comunican utilizan palabras que están destinadas a transmitir un significado. Este significado es fundamental para entenderse y, en el caso de las aplicaciones RDF, es vital para procesar la información correctamente. Para ello, es necesario que tanto emisor como receptor interpreten el mismo significado para los mismos términos. Por este motivo, hay que ser riguroso en la definición de cada uno de ellos.

El significado en RDF se expresa mediante la referencia a un esquema. Este esquema define los términos que se utilizarán en las declaraciones RDF y les da significados específicos.

En el esquema se documentan las definiciones y restricciones de uso de las propiedades. Para evitar confusiones entre definiciones independientes del mismo término, RDF utiliza el recurso de espacio de nombres XML. Los espacios de nombres son una forma de vincular un uso específico de una palabra con el esquema donde está definida. En RDF, cada propiedad utilizada en una declaración debe identificarse con un espacio de nombres o un esquema.

En definitiva, un espacio de nombres (namespace) no es más que un “contenedor” de elementos en el que dichos elementos son únicos. Esto se hace porque puede haber un usuario A que defina un elemento “Part” a partir de los atributos “Part Number” y “Description” y otro usuario B que defina otro elemento “Part” por los atributos “Part Number” y “Raw material” de modo que exista más de un elemento “Part”, pero si se está en el namespace del usuario A no hay duda de que “Part” es el elemento definido por “Part Number” y “Description” y si se está en el namespace del usuario B no la habrá de que es el definido por “Part Number” y “Raw material”.

El espacio de nombres XML se identifica mediante una referencia URI (Identificador de recursos uniforme) o

IRI (Identificador de recursos internacionalizados), que no es más que una extensión de la primera.

Prefijo del espacio de nombres	IRI del espacio de nombres	Vocabulario RDF
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	Vocabulario RDF incorporado
rdfs	http://www.w3.org/2000/01/rdf-schema#	Vocabulario de RDF Schema
xsd	http://www.w3.org/2001/XMLSchema#	Tipos XSD compatibles con RDF

Tabla 3.1 Prefijos de espacios de nombres e IRI's asociados

3.2 Ontologías

OWL puede usarse para representar explícitamente el significado de los términos y las relaciones que existen entre dichos términos definidos en vocabularios. Esta representación de términos y sus interrelaciones se llama ontología.

Las ontologías nacen para resolver los problemas que surgen de usar terminología diferente para referirse al mismo concepto o usar el mismo término para referirse a conceptos diferentes. Proponen un punto de vista del dominio, simple y entendible, desde el que se representa, explícitamente, el significado de diferentes términos o conceptos con sus relaciones.[xxxi]

Una ontología es una especificación explícita de una conceptualización compartida capaz de representar el conocimiento sobre un dominio y transmitir dicho conocimiento, haciendo posible la comunicación entre personas y aplicaciones.[x]

La definición de un dominio es la de las clases y relaciones que puede haber entre dichas clases en el ámbito en cuestión cuya caracterización se realiza a través de propiedades que se asocian a cada una de las clases o relaciones. Por tanto, una ontología de definición está compuesta por propiedades, que se crean para caracterizar clases y que a su vez se dividen en "Object Properties" y "Data Properties", y las propias "Classes".

- Object Properties: Se definen para caracterizar relaciones entre clases. Estas propiedades tienen como dominio una clase relación y como rango una clase de individuos que es la que van a relacionar. Como es obvio, para crear una relación entre dos miembros, será necesario instanciar dos "Object properties" con un mismo dominio, pero no necesariamente con un mismo rango.
- Data Properties: Son atributos que se definien para caracterizar clases. Estas propiedades tienen como dominio una clase cualquiera y como rango el tipo de elemento que será instanciado para asociar un valor a dicho atributo.
- Classes: Son las entidades que componen el dominio en cuestión y quedan definidas a través de las propiedades que les sean asociadas. Cada clase puede declarar una serie de atributos clave entre los que le han sido asignados de manera que no se pueda instanciar más de un miembro de dicha clase con los mismos valores de estos atributos. De esta manera, se hace posible la identificación de miembros de una clase a través de la búsqueda de sus atributos clave sin ambigüedades.

Pero hay más, una ontología permite instanciar el dominio definido. Es decir, crear individuos de cada clase y relaciones entre dichos individuos asignando valores a sus atributos con el fin de poder transmitir información. O lo que es lo miemos, una ontología siempre va a estar compuesta por la definición del dominio (Object properties, Data properties y clases) y puede que por la instanciación del dominio, que quedaría formada por los "Individuals":

- Individuals: Son los miembros de una clase que ha sido instanciada. Esto no es más que la creación de entidades de algún tipo asciando valores a sus atributos para adquirir información y poder crear conocimiento.

Las ontologías se pueden crear de diversas maneras, pero para que exista interoperabilidad es necesario que toda esta información se pueda intercambiar en un documento con formato común y la sintaxis elegida, al igual que en los casos anteriores, ha sido la XML.

3.3 XML

El XML (Lenguaje de Marcado Extensible o Extensible Markup Language) es un formato de texto simple y muy flexible derivado de SGML (Lenguaje de Marcado Generalizado Estándar o Standard Generalized Markup Language). No es un lenguaje en sí mismo, sino más bien un estándar que permite definir lenguajes de marcado adecuados para usos determinados.

Esto quiere decir que no está predefinido, por lo que cada uno debe definir sus propias etiquetas con el fin de compartir datos a través de diferentes sistemas.

Un documento XML se compone de elementos y atributos. El documento se estructura de forma jerárquica de modo que aparecerá un nodo principal, denominado raíz, y el resto de los nodos como descendencia de éste, ya sea como hijos, nietos… Se denomina nodo a un elemento con sus atributos y todo lo que cuelga de él. Es decir, si se selecciona el nodo raíz, se está seleccionando todo el documento, pero si se selecciona otro nodo cualquiera, se estaría seleccionando ese elemento con sus atributos y los de su descendencia, si éste tuviera.

Cada elemento se caracterizará por una serie de atributos. Los atributos son pares nombre-valor que pueden asociarse a los elementos para definirlos.

A la hora de escribir el texto, hay que distinguir si el elemento tiene un valor asociado o no.

- Si lo tiene, el nodo sería "<Nombre elemento Nombre atributo=Valor atributo>Valor elemento</Nombre elemento>"
- Si no lo tiene, el nodo sería "<Nombre elemento Nombre atributo=Valor atributo/>"

A continuación, se muestra un ejemplo de un nodo "NamedIndividual" con un atributo que a su vez tiene cuatro nodos hijos, también con un atributo, tres de los cuales tienen un valor asociado al elemento.

```
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#CS214001-0001">
    <rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part"/>
    <item_number rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CS214001-0001</item_number>
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Front Spar Assy</name>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CS214001-0001</rdfs:label>
</owl:NamedIndividual>
```

Este nodo se leería como que existe un individuo de tipo "Part", que será una clase definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20, caracterizado por los atributos de tipo string "ítem_number" y "name" cuyos valores son CS214001-0001 y Front Spar Assy, respectivamente y una etiqueta, también de tipo string, cuyo valor será CS214001-0001.

3.4 Definición de una ontología. Caso de uso

El presente apartado se centra en mostrar la definición de una ontología para la gestión de una lista de materiales o BOM (Bill of Material).

Una lista de materiales (BOM) es una lista de las materias primas, normales, químicos, elementales, conjuntos y las cantidades necesarias de cada uno para fabricar un producto final.[xxxii]

Puede ser utilizado para comunicarse entre los participantes en la fabricación, o confinado a una única planta de fabricación.

Una lista de materiales puede definir productos como están diseñados (lista de materiales de ingeniería), como son pedidos (lista de materiales de ventas), como son fabricados (lista de materiales de fabricación), o como son mantenidos (lista de materiales de servicio). Los diferentes tipos de listas de materiales dependen de las

necesidades del negocio y el uso para el cual están destinados.

Las listas de materiales son de naturaleza jerárquica con el nivel superior representando el producto final que puede ser un subconjunto o un elemento terminado. Las listas de materiales que describen los subconjuntos se denominan listas de materiales modulares. Un ejemplo de esto es la lista de materiales NAAMS que se utiliza en la industria de la automoción para listar todos los componentes de una cadena de montaje. La estructura de la lista de materiales NAAMS es sistema, línea, herramienta, unidad y detalle.[xxxiii]

En la actualidad una ontología se describe mediante un fichero de texto, aunque existen herramientas que facilitan la definición. Entre las principales herramientas para la definición de ontologías destaca "Protege".[xxxiv] Protege es un entorno de código abierto para la definición de sistemas inteligentes mantenido por una amplía comunidad, gobiernos, académicos y usuarios corporativos.

3.4.1 Fases en la creación de una ontología.

El proceso de definición de una ontología en Protege puede resumirse en una serie de pasos:

1.- **Definir el dominio y alcance de la ontología**. El proceso de definir el domino corresponde a responder a varias preguntas:

- ¿Cúal es el dominio que la ontología debe cubrir?
- ¿Para qué se va a utilizar la ontología?
- ¿Qué tipos de preguntas debe responder la ontología?
- ¿Quién usará y mantendrá la ontología?

Como ejemplo de métodos de definición de una ontología, considérese que que se desea implementar una ontología que recoja la definición de la lista de materiales descrita en el apartado anterior. Si analizamos las respuestas podemos decir, que el dominio se centra en definir una lista de materiales de un producto o componente y cómo se estructura.

La ontología se empleará para definir la estructura de un producto y sus componentes y debe responder a una pregunta básica, ¿De qué componentes consta un producto? Finalmente, al ser un ejercicio práctico se mantendrá para este proyecto.

2.- Considerar reutilizar ontologías existentes. Existen librerías de ontologías que pueden ser reutilizadas, muchas de ellas están disponible en formato electrónico.

Particularmente, la definición de una ontología para definir una lista de materiales se realiza como ejemplo académico.

3.- Enumerar los términos mas importantes en la ontología. Es útil escribir una lista de todos los términos sobre los que gustaría hacer declaraciones o explicar a un usuario. ¿Cuáles son los términos de los que gustaría hablar? ¿Qué propiedades tienen esos términos? ¿Qué gustaría decir sobre esos términos?

En el caso de una ontología destinada a definir la lista de materiales, los principales términos a considerar son, PART y PARTBOM.

El término PART destaca que tiene una propiedad "ítem_number" que identifica unívocamente una parte en el sistema. Por otro lado, el término PARTBOM tiene una propiedad "quantity" que especifica la cantidad de un componente que se necesita para tener una unidad del producto e "ítem_number" para identificar tanto a los componentes como al producto.

4.- Definir las clases y jerarquía de clases. Las clases se definen en base a los términos de la ontología.

En este caso, se necesita una clase "**Part**" para las partes y una clase "**Part_BOM**" para las relaciones entre partes.

5.- Definir los atributos o propiedades de las clases. Las clases en si mismo no proporciona información suficiente para responder a las preguntas que pueden hacerse a la ontología.

Y, ¿qué atributos se necesitan para definir cada clase?

Para las partes, se van a definir dos atributos: "ítem_number" y "name" y para las relaciones entre partes,

tres: “ParteDestino”, “ParteOrigen” y “quantity”.

De este modo quedaría una clase “Part” definida por los “data propeties”: “ítem_number” y “name” y una clase “Part_BOM” definida por el “data propety”: “quantity” que relaciona partes con los “object properties”: “ParteDestino” y “ParteOrigen”.

6.- Definir los aspectos de los atributos. Los atributos pueden tener diferentes facetas que describen el tipo de valor, los valores permitidos, el número de valores (cardinalidad) y otras características de los valores que puede tomar la propiedad.

7.- Crear instancias. El último paso es crear instancias individuales de clases en la jerarquía. La definición de una instancia individual de una clase requiere (1) elegir una clase, (2) crear una instancia individual de esa clase y (3) completar los valores de los atributos

3.4.2 Definición de la ontología para lista de materiales en protege

Al trabajar en entornos colaborativo, Protege define una ontología como un enlace web (Ontology IRI). Esto permite su posterior versionado de manera que la ontología vaya creciendo de forma colaborativa (Ontology Version IRI).

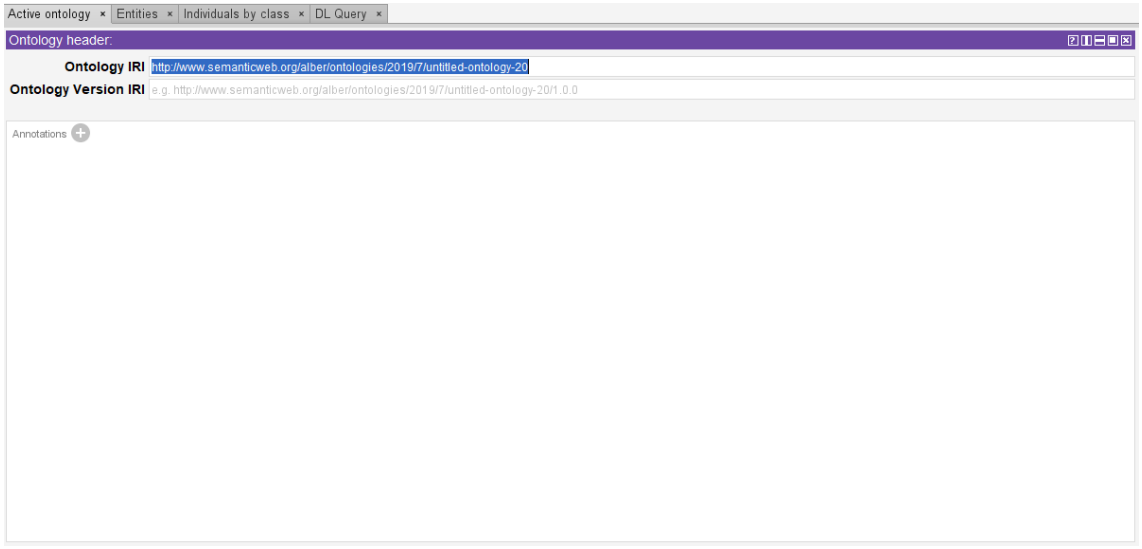

Figura 3.1 Ontología activa en Protege

Definición de clases.

En el menú principal además de la ontología activa se definien las entidades o clases que componen la ontología. Para crear una clase, se debe hacer clic en las pestañas “Entities” y “Classes”, seleccionar “owl: Thing” y darle a “Add subclass”. Todas las ontologías parten de una clase genérica denominada Thing.

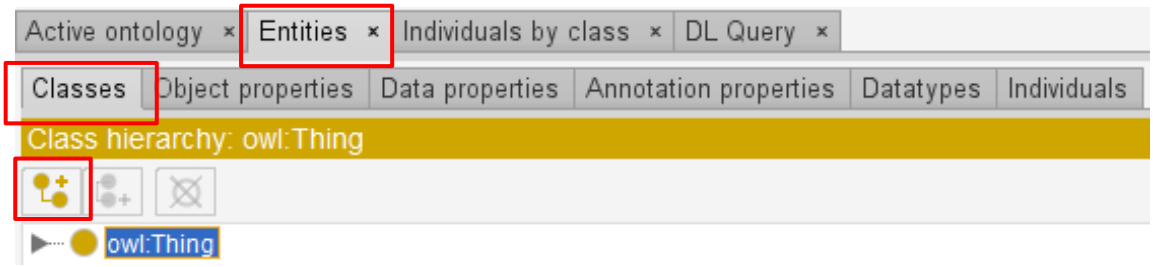

Figura 3.2 Entidades en Protege: Clases.

A continuación, aparecerá una pantalla donde habrá que escribir el nombre de la clase, en este caso, “Part” y clicar en aceptar. Como puede observar, al ser un sistema basado en la web semántica, la clase queda referenciada como el nombre de la ontología seguido por el nombre de la clase, separada por el carácter #.

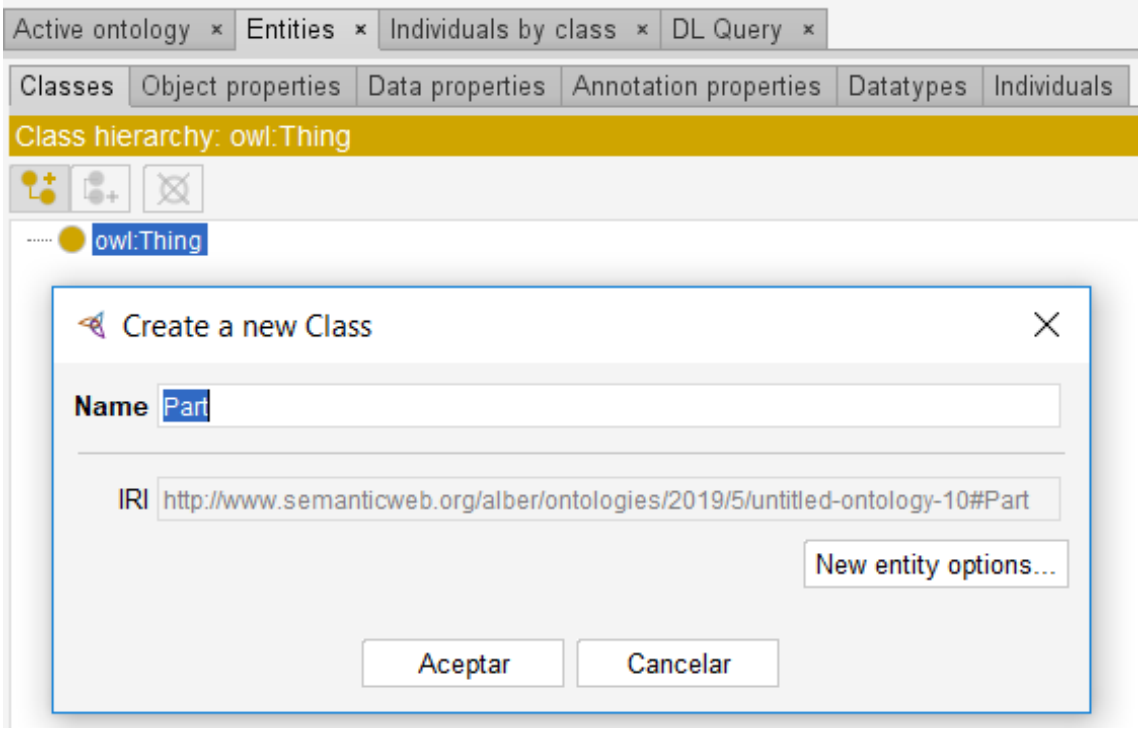

Figura 3.3 Creación de clases en Protege

Así quedaría la clase “Part”.

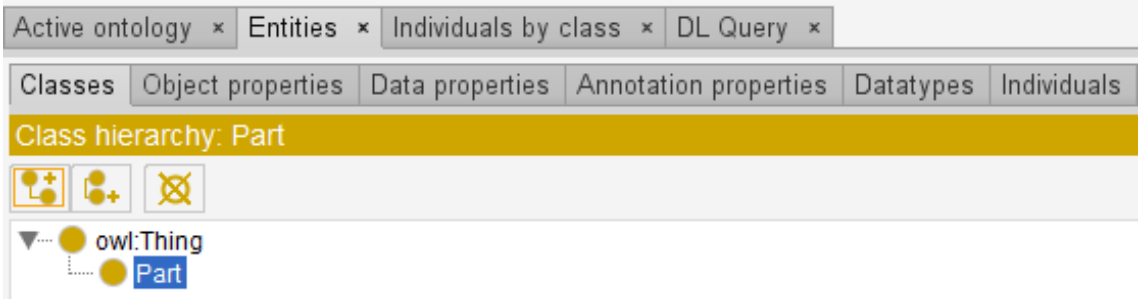

Figura 3.4 Resultado de la creación de la clase "Part"

Si se analiza la estructura de una lista de materiales, se observa qué componente o parte está compuesto por otras partes. La clase PartBOM recoge la relación entre dos partes o componentes. El proceso de definición mediante la herramienta es idéntico a la generación de la clase con la única diferencia que la clase se denomina “Part BOM”.

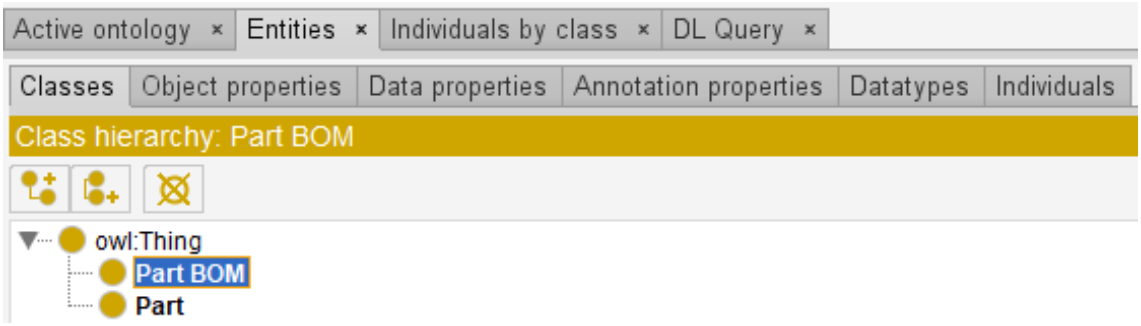

Figura 3.5 Resultado de la creación de la clase "Part BOM"

Figura 3.6 Representación de las clases creadas

En la parte de la derecha, aparecen las ventanas de anotaciones y descripción de la clase.

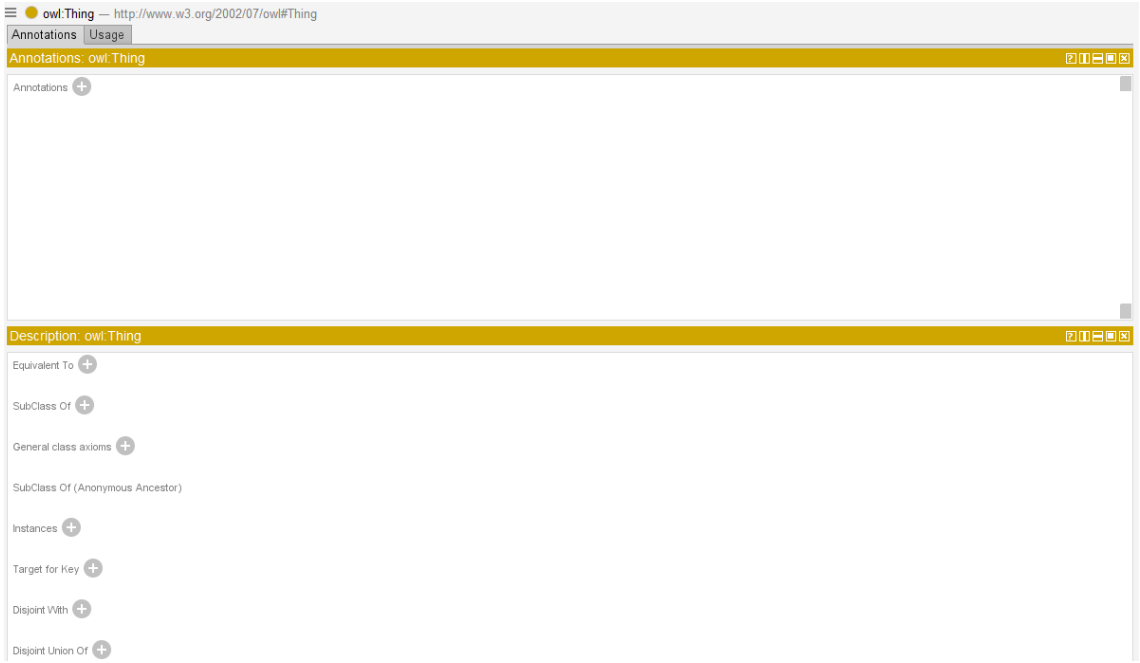

Figura 3.7 Propiedades de una clase

En la primera, se pueden añadir etiquetas, que aparecerán en los últimos nodos de la clase cuando se exporte a un formato XML. Nótese la especial importancia de estas etiquetas, porque Protege no reconoce espacios en los nombres y, automáticamente, los reemplaza por "_". Por ello, si se quiere crear la clase "Part BOM", es necesario crear una etiqueta con ese nombre para añadirla a la clase que Protege va a llamar "Part_BOM".

En la segunda, se pueden añadir propiedades a las clases. Estas propiedades o axiomas son teoremas que se declaran sobre relaciones que deben cumplir los elementos de una ontología. Los axiomas que ya aparecen definidos y los atributos que se van a crear más adelante, es lo que diferencia el OWL de los lenguajes anteriores:

- "Equivalent to" permite hacer dos clases equivalentes. Un axioma que enlaza la definición de una clase con otra clase. Tenga en cuenta que el aximoa "Equivalent to" no implica que ambas clases sean iguales, sino que están relacionadas.
- "SubClass Of" permite colgar una clase B de una clase A, de modo que todas las de tipo B sean de tipo A, pero no todas las de tipo A tengan que ser de tipo B. Esta jerarquía también se puede crear desde la ventana "Class hierarchy" añadiendo las clases a las que ya han sido creadas previamente.
- "Disjoint With" permite independizar por completo una clase de otra. Es decir, se tendrían tres opciones:
    1. Una clase B puede estar totalmente contenida en otra A, de modo que B sería una subclase de A.
    2. Una clase B puede estar parcialmente contenida en A, de modo que las clases A y B estén solapadas y pueda haber individuos que pertenezcan a ambas clases.
    3. Una clase B puede ser totalmente independiente de A, de modo que cada individuo pertenecerá sólo a una de las clases.

- "General class axioms" permite usar cualquiera de las tres propiedades anteriores, pero escribiendo el axioma. Por ejemplo, si se quiere independizar A de B, habría que clicar en "Add" y en la ventana que sale, escribir: "A DisjointWith B".
- "SubClass Of (Anonymous Ancestor)" se escribe automáticamente en una clase C cuando se tiene una clase B que pertenece a una clase A y posteriormente se dice que la clase C es equivalente a la B, de modo que ésta se convertiría automáticamente en subclase de A.
- "Instances" permite visualizar los individuos instanciados de esa calse.
- "Target for Key" permite seleccionar los "object properties" o "data properties" que son clave y, por tanto, hacen única a una clase.
- "Disjoint Union Of" permite decidir quién es la clase principal en un dominio en el que hay clases solapadas.

En este caso, sólo se va a añadir una propiedad a la **clase "Part".** Esta clase tendrá como "Target for Key" uno de sus "data properties", el atributo "**ítem_number**" puesto que es la única manera de que cada part number sea **único** y quede perfectamente identificado. De este modo, si se busca una "Part" cuyo "ítem_number" es XXXXXXXX-XXXX, sólo se puede encontrar una.

Sin embargo, no pasaría lo mismo con el atributo "**name**" puesto que sí podría haber varios part numbers con el mismo nombre. Por ejemplo, sin ir más lejos se podría tener una pieza con "ítem_number" XXXXXXXX-XXXX y "name" soporte que a partir de un avión de corte definido se requiera, en vez de según plano, con una condición de entrega de dar un taladro a diámetro definitivo. Para diferenciar esta pieza de la original, se cambiaría el "ítem-number" añadiendo una terminación al final, XXXXXXXX-XXXX-01, pero el "name" seguiría siendo el mismo.

Definición de atributos o propiedades

Los atributos o propiedades en una ontología pueden ser de dos tipos, DataProperty y ObjectProperty. Los DataProperty se emplean para definir atributos que toman valores, su rango es un tipo de dato básico (entero, flotante, string…). Los ObjectProperty se emplean para definir atributos cuyo contenido corresponde a una clase, es decir, su rango de valores es otra clase.

Para crear una "object property", se debe clicar en la pestaña "Object properties", seleccionar "owl:topObjectProperty" y hacer clic en "Add sub property".

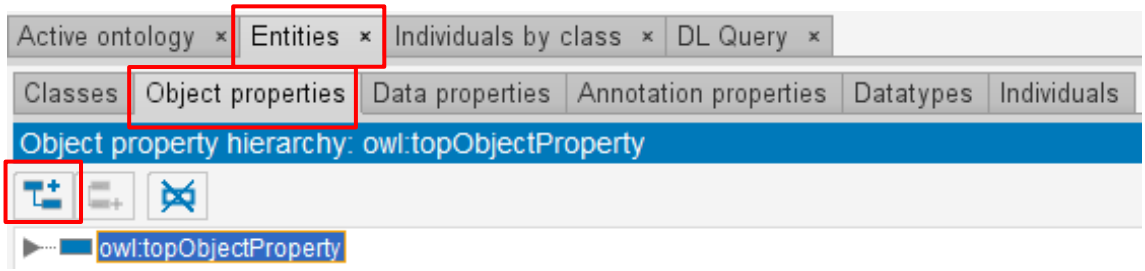

Figura 3.8 Entidades en Protege: Propiedades objeto

A continuación, aparecerá una ventana como esta y habrá que escribir el nombre de la propiedad, en este caso, "ParteDestino".

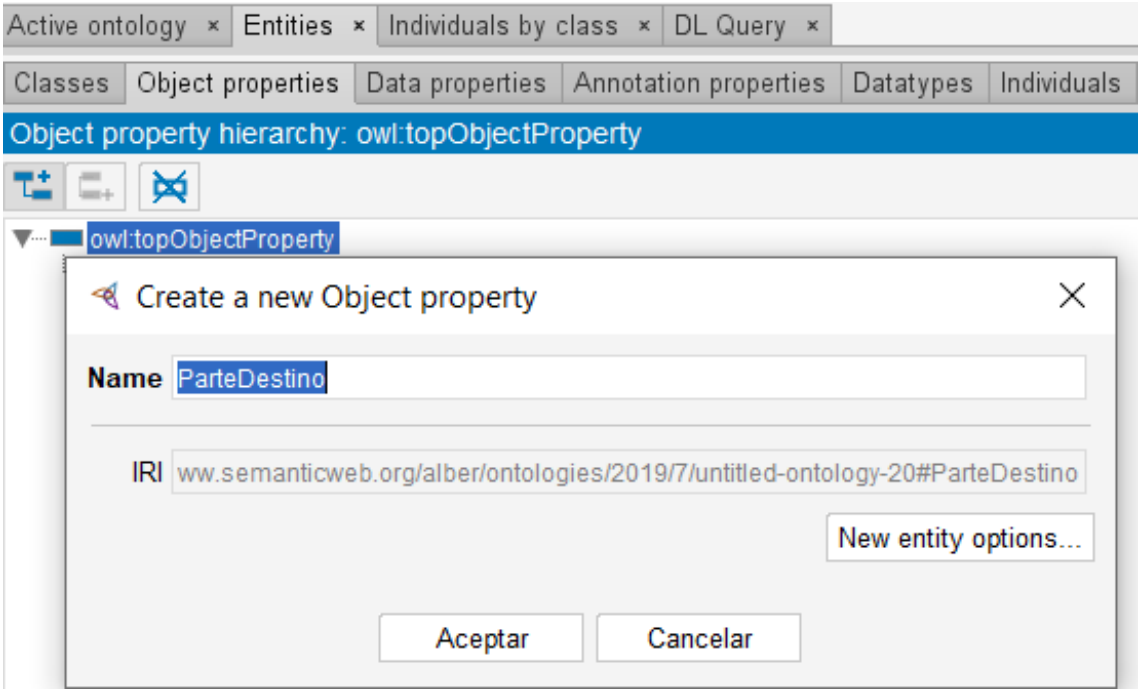

Figura 3.9 Creación de propiedades objeto en Protege

En la parte de la derecha, aparecen las ventanas de anotaciones, características y descripción de la propiedad.

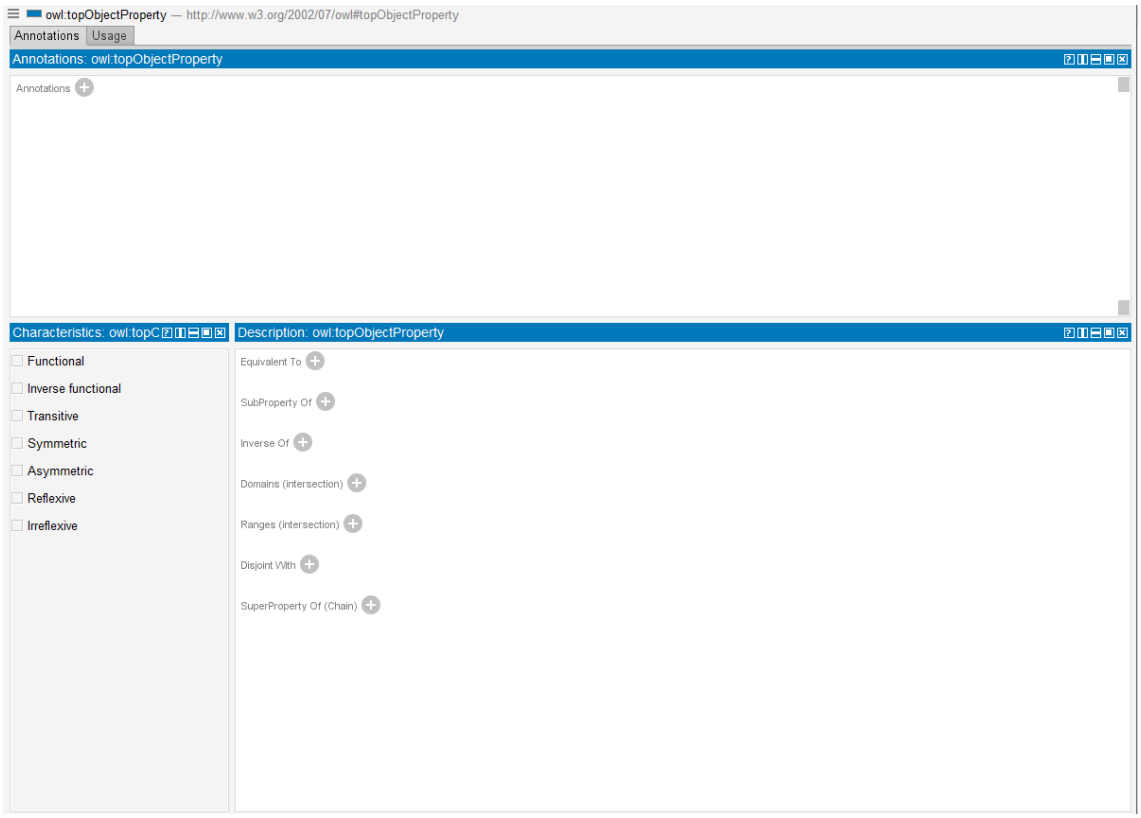

Figura 3.10 Características de una propiedad objeto

Entre las características se encuentran:

- "Functional": Con esta opción se indica que un individuo A sólo puede estar relacionado, a través de esta propiedad, con un único individuo B. Es decir, si se quiere relacionar a A con más individuos, tendrá que ser a través de otras propiedades.

- "Inverse functional": Con esta característica se indica que la propiedad inversa es funcional. Para ello, primero hay que indicar cuál es la propiedad inversa de ésta en el apartado de descripción. Por ejemplo: Si hay un conjunto que se va a relacionar con las elementales que lo componen a través de una propiedad "tiene hijos" y también se van a relacionar las piezas a nivel elemental con su conjunto superior a través de una propiedad "es hijo", se podría decir que las propiedades "tiene hijos" y "es hijo" son inversas la una de la otra.
- "Transitive": Si una propiedad relaciona por una parte a un individuo A con otro B y por otra parte a ese individuo B con otro C y, además, es transitiva, entonces está relacionando a A con C.
- "Symmetric": Si una propiedad relaciona a un individuo A con otro B y, además es simétrica, está relacionando a B con A a través de esa misma propiedad.
- "Assymetric": Si una propiedad relaciona a un individuo A con otro B y, además es asimétrica, entonces B no puede estar relacionado con A a través de esa misma propiedad.
- "Reflexive": Con esta característica, una propiedad relaciona a un individuo consigo mismo.
- "Irreflexive": Con esta característica, una propiedad no puede relacionar a un individuo consigo mismo.

En la descripción, además de las explicadas anteriormente, se tienen:

- "Domains": Clase o serie de clases que indican que cualquier individuo con esta propiedad es una instancia de alguna de ellas. Puesto que "ParteDestino" es atributo de la relación "Part_BOM", su dominio será la clase "Part_BOM".
- "Ranges": Clase o serie de clases que indica que el valor de una propiedad es una instancia de alguna de ellas. Puesto que la relación es entre partes y "ParteDestino", únicamente, puede tomar valores de parte, el rango será la clase "Part".

Para ello, se clica en "Add Domains", se selecciona la clase correspondiente y se pulsa aceptar.

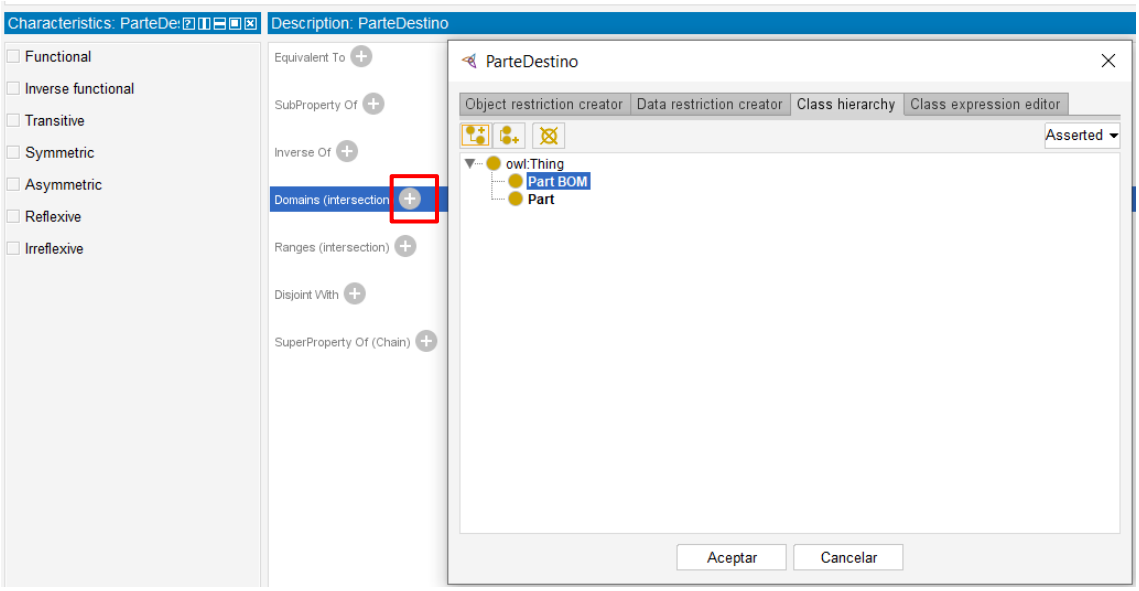

Figura 3.11 Dominio de una propiedad objeto

A continuación, se hace lo mismo con "Add Ranges".

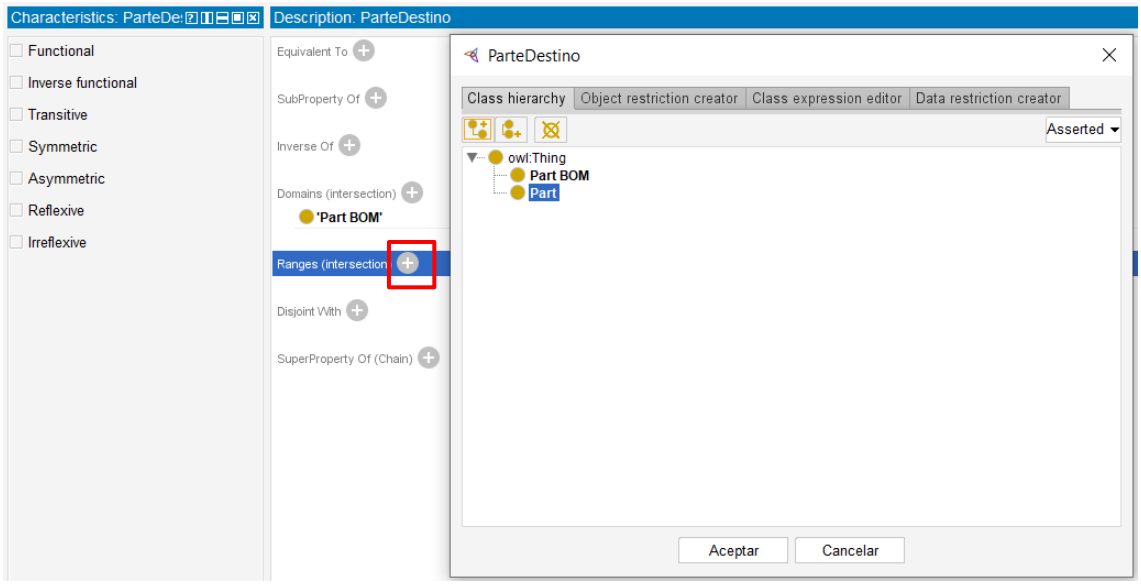

Figura 3.12 Rango de una propiedad objeto

Y así quedaría definida la propiedad objeto “ParteDestino”.

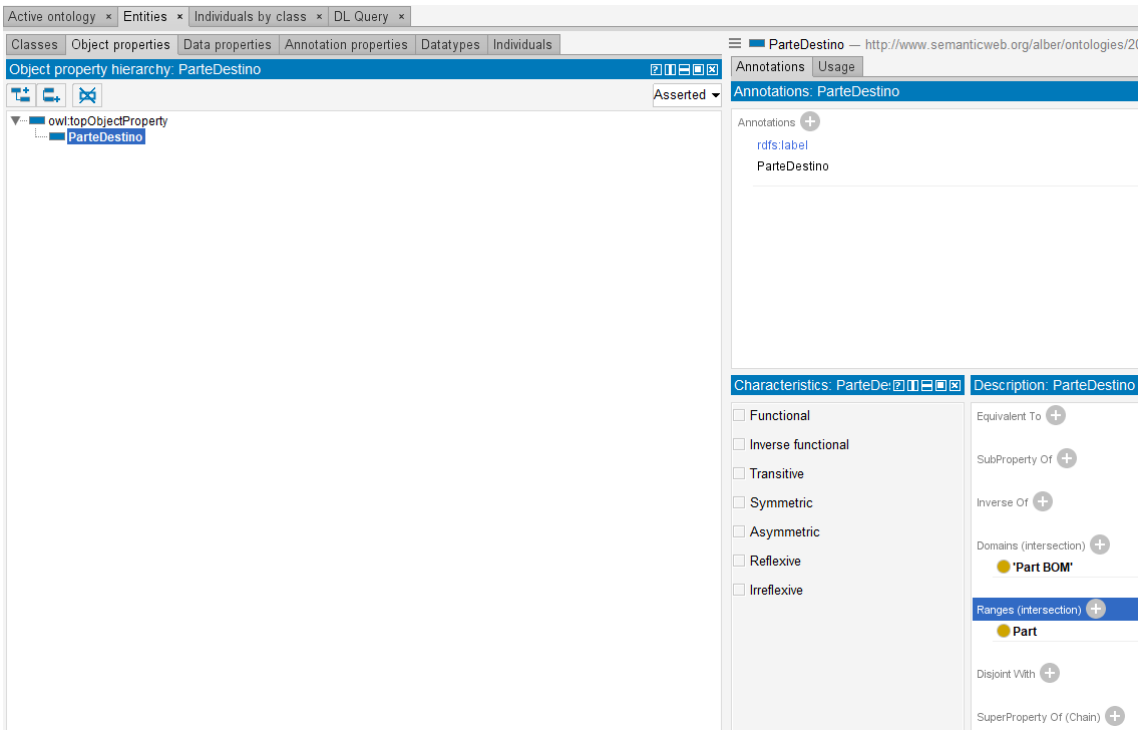

Figura 3.13 Resultado de la creación de la propiedad objeto "ParteDestino"

Ahora habría que repetir estos pasos tantas veces como “object properties” se quieran definir, en este caso, es suficiente con hacerlo una ver más para definir la propiedad objeto “**ParteOrigen**”.

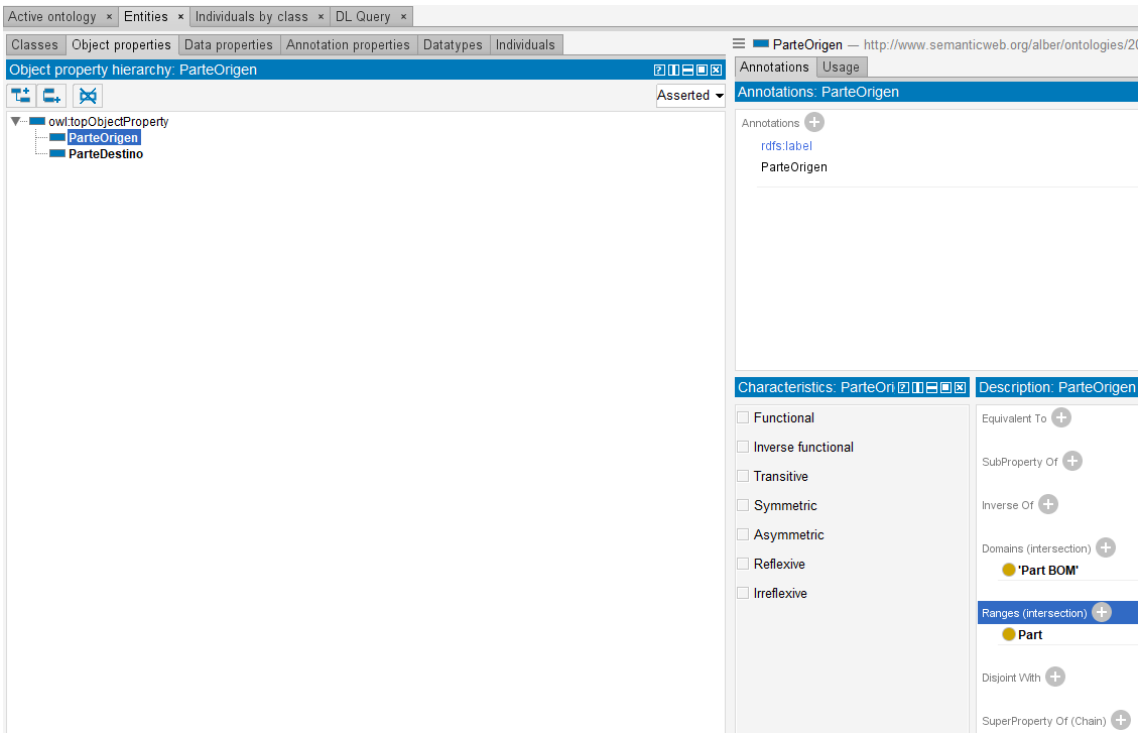

Figura 3.14 Resultado de la creación de la propiedad objeto "ParteOrigen"

De la misma forma que se ha hecho con los “object properties”, se crean los “data properties”. Se clica en la pestaña “Data properties, se selecciona “owl:topDataProperty” y se pulsa “Add sub property”.

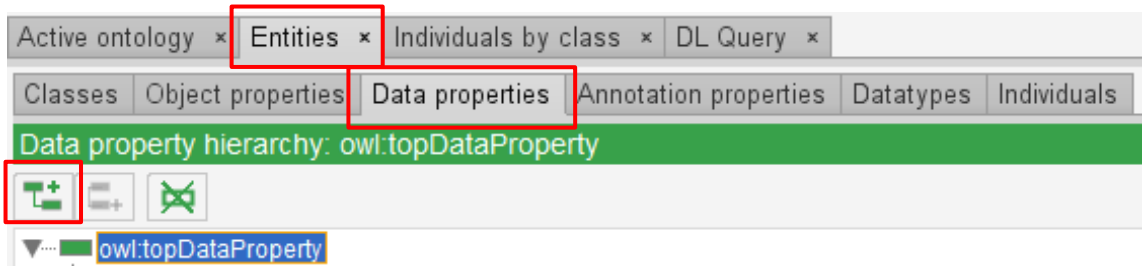

Figura 3.15 Entidades en Protege: Propiedades dato

A continuación, aparecerá una ventana en la que habrá que poner el nombre de la propiedad, “ítem_number” y pulsar aceptar.

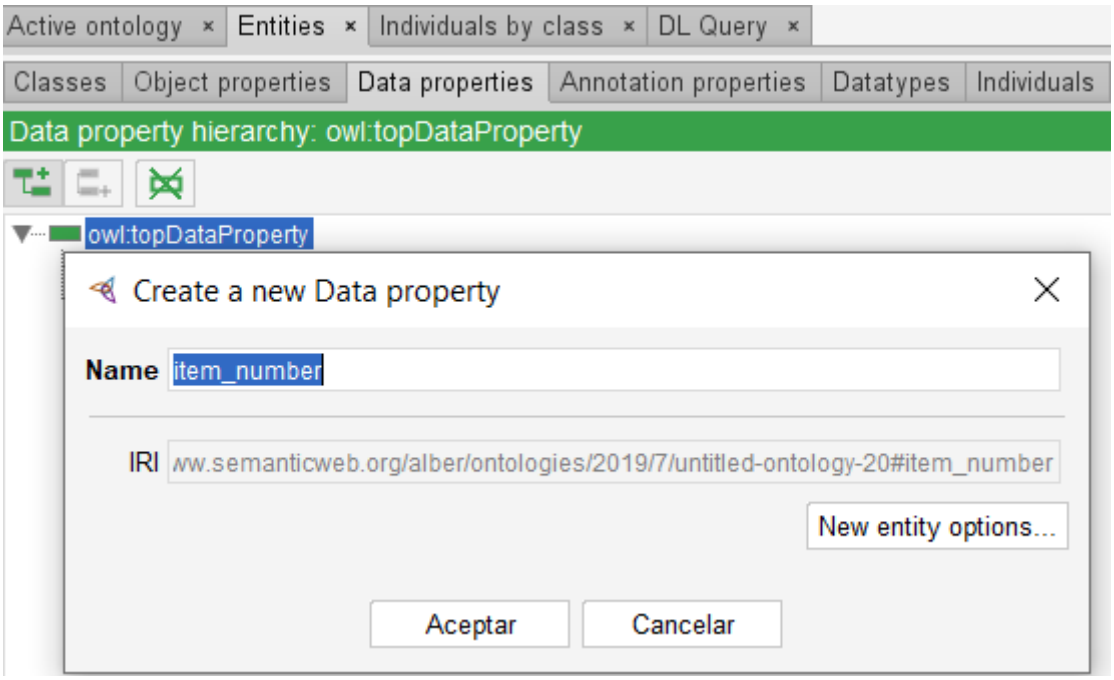

Figura 3.16 Creación de propiedades dato en Protege

En los “data properties” también aparecen los apartados de anotaciones, descripción y características que se han definido anteriormente.

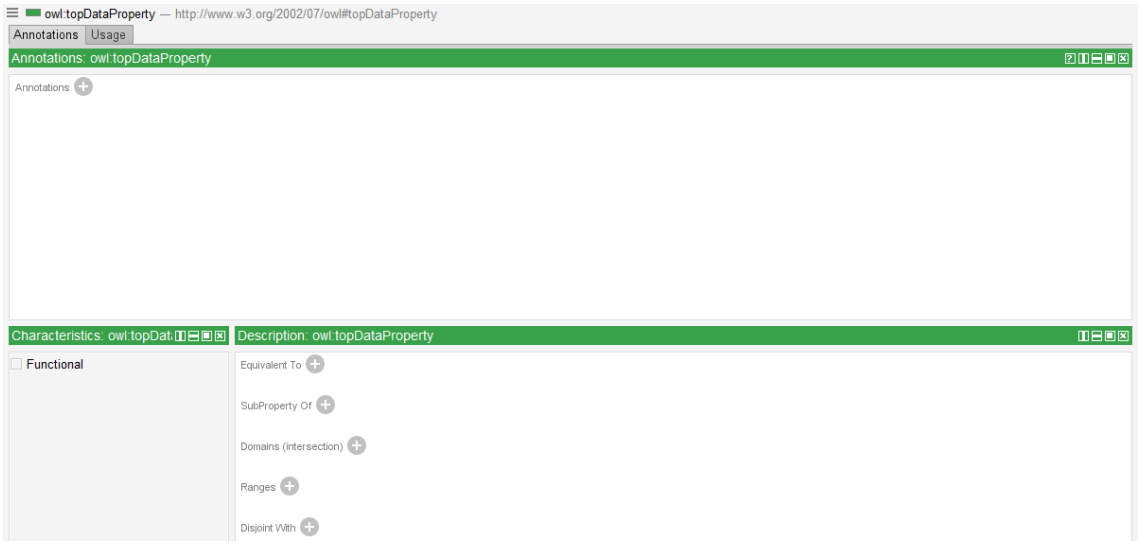

Figura 3.17 Características de una propiedad dato

Aquí la característica “Functional” vendría a decir que un individuo sólo puede tener ese atributo una vez.

En este caso, ambos atributos serán funcionales porque cada individuo instanciado sólo tendrá un único “ítem_number” y un único “name”.

Ahora habría que definir el dominio y el rango. Como esta propiedad es un atributo de la clase “Part” y el valor que va a tomar dicho atributo es una cadena de caracteres, el dominio será “Part” y el rango, “string”.

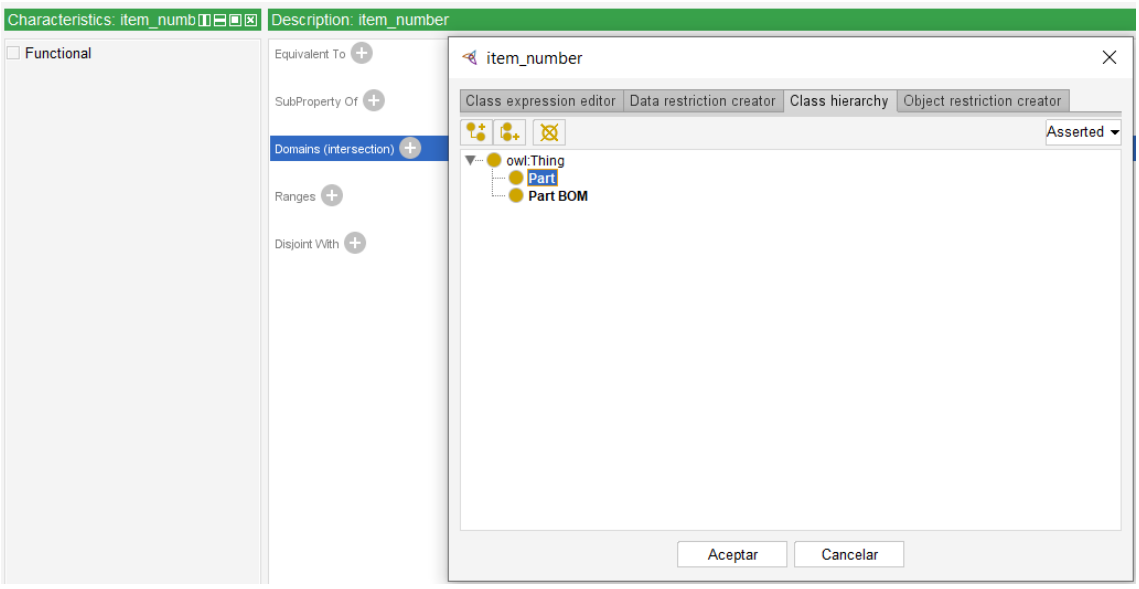

Figura 3.18 Dominio de una propiedad dato

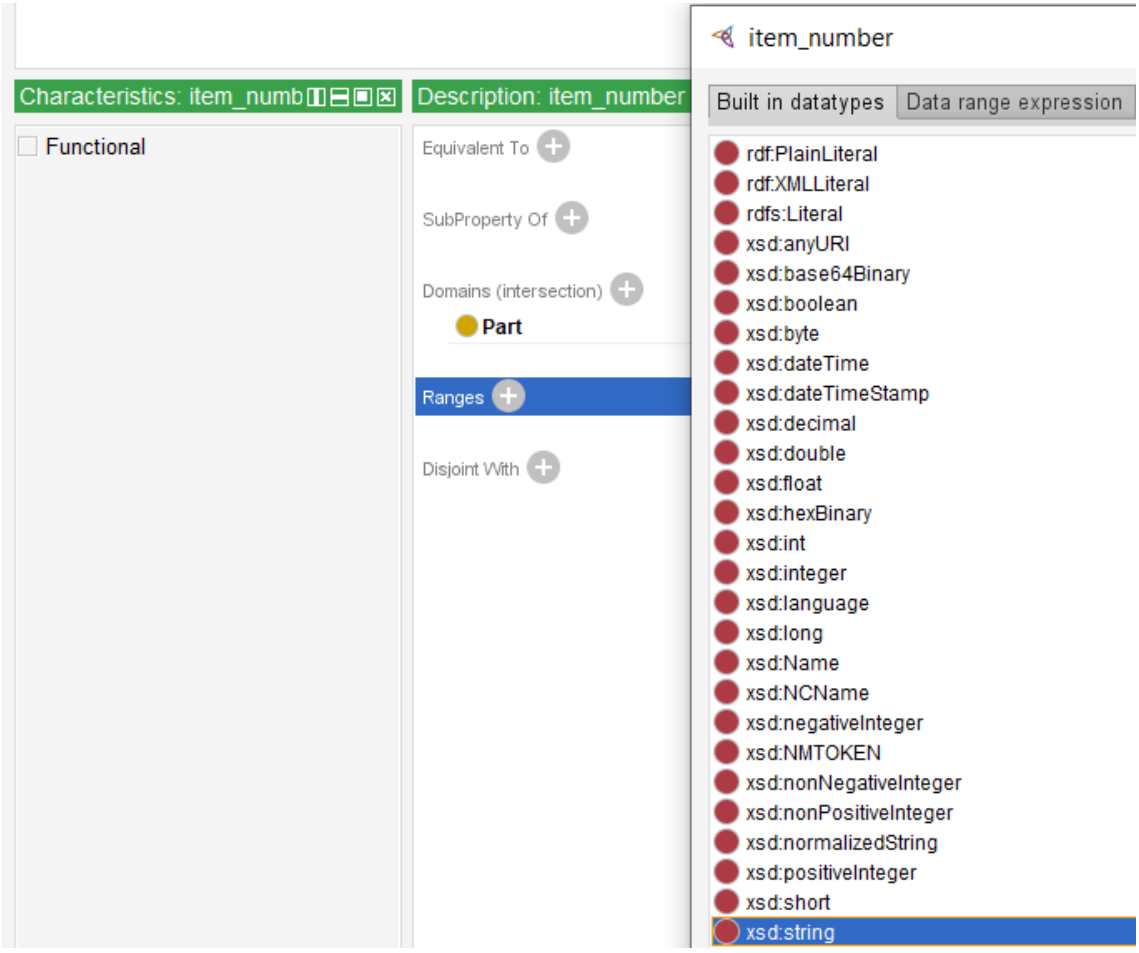

Figura 3.19 Rango de una propiedad dato

Y así quedaría definido el “data property” ítem_number.

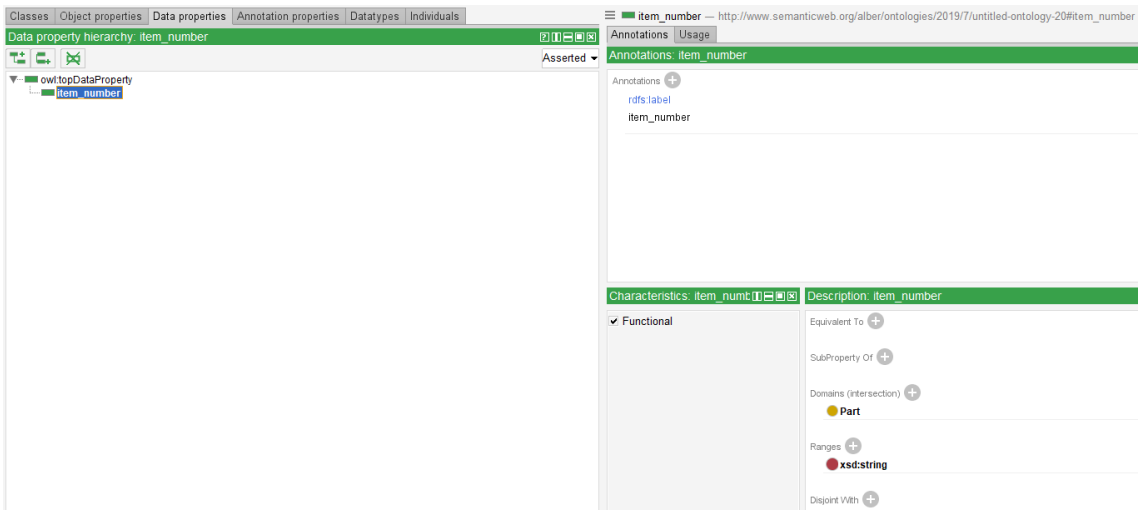

Figura 3.20 Resultado de la creación de la propiedad dato "item_number"

De la misma manera, se crearían el resto de “data properties”.

En el caso de la propiedad “quantity”, puesto que es un atributo de la clase “Part BOM” y el valor de dicho atributo es numérico, el dominio será “Part BOM” y el rango, “int”. Además, este atributo no será funcional porque el mismo individuo puede estar relacionado varias veces con diferente cantidad.

De este modo, los tres “data properties” quedarían así:

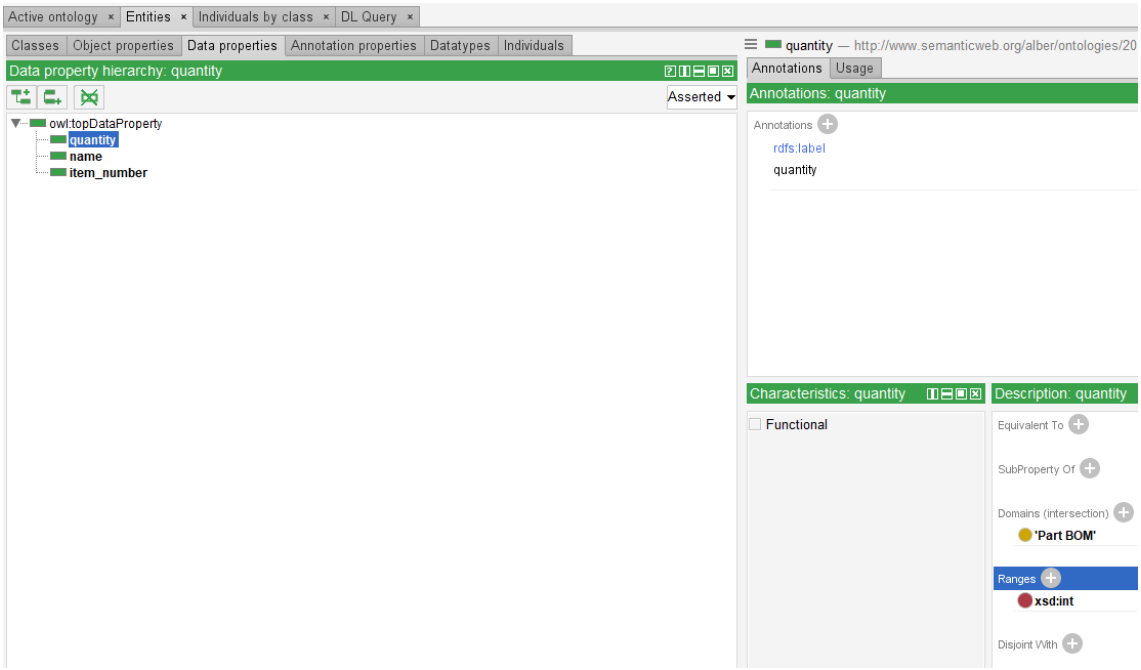

Figura 3.21 Resultado de la creación de la propiedad dato "quantity"

Creación de instancias o individuos

Una vez se han creado las clases y propiedades, se puede instanciar. Es decir, una vez definida la ontología, se pueden generar individuos de los tipos de clases que se han creado y caracterizarlos añadiendo valores a sus atributos.

Para ello, habría que irse a la pestaña “Individuals” y clicar en “Add individual”.

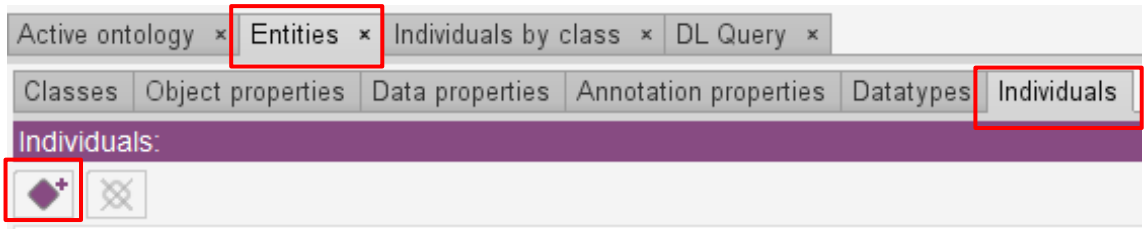

Figura 3.22 Entidades en Protege: Individuos

Ahora aparecerá una ventana en la que habrá que escribir el nombre de la instancia y pulsar aceptar. En este caso, se va a instanciar una "Part" y para hacerla única se le va a llamar por sus atributos clave. Como en este caso, el único atributo clave es "ítem_number", se le va a denominar "CS214001-0001".

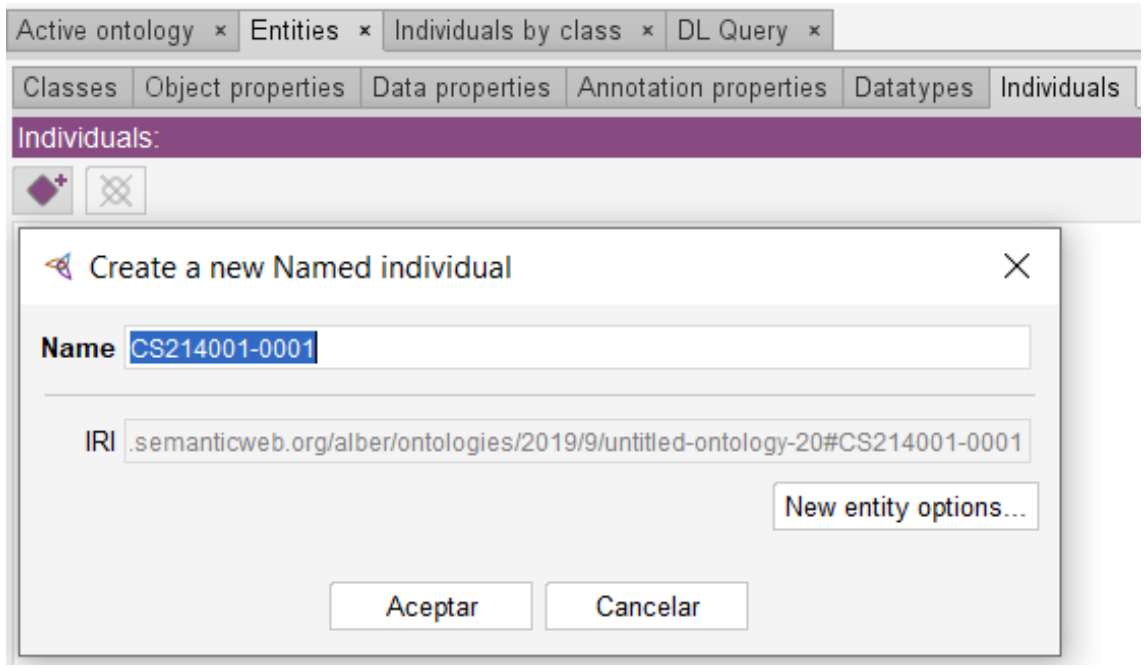

Figura 3.23 Creación de individuos en Protege

A continuación, se indica la clase de la instancia. Pare ello, se va a la ventana "Description", se clica en "Add Types", se selecciona "Part" y se pulsa aceptar.

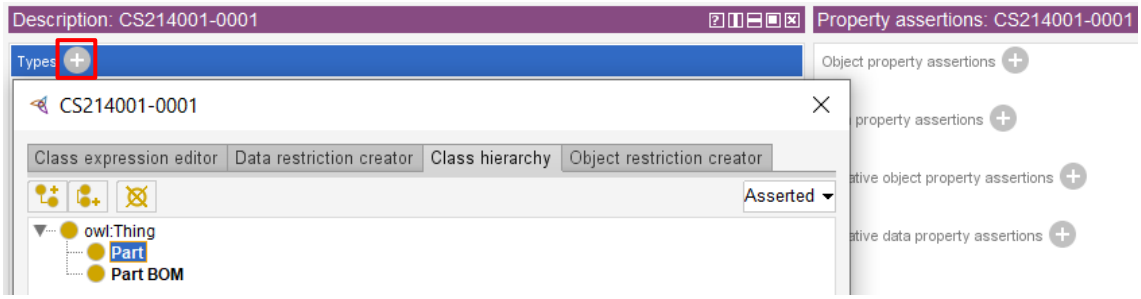

Figura 3.24 Selección de la clase del individuo "CS214001-0001"

La clase "Part" se ha definido con dos atributos, "ítem_number" y "name", por lo que las instancias de este tipo deben tener como mínimo un atributo, el atributo clave, y como máximo dos, los dos que han sido definidos.

En este caso, se va a instanciar dando valores a ambos atributos. Para ello, se va a la ventana "Property assertions", se clica en "Add Data property assertions", se señala "ítem_number", en "type" se selecciona "string", se escribe el valor del atributo en el cuadro y se pulsa aceptar.

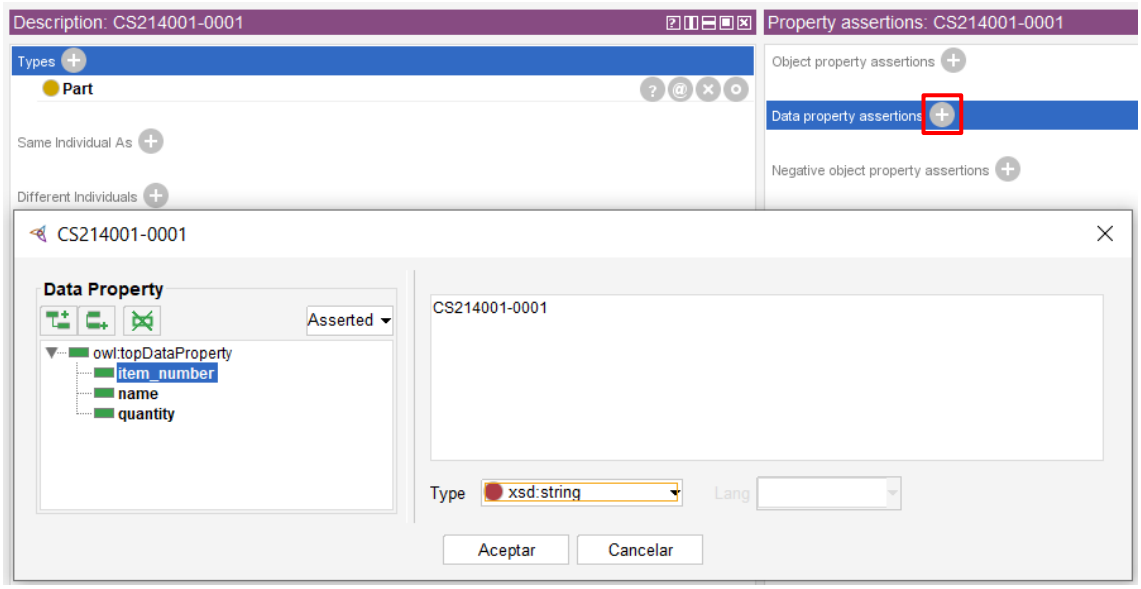

Figura 3.25 Caracterización del individuo “CS214001-0001” a través de propiedades dato

A continuación, se hace lo mismo para el otro atributo, se señala “name”, en “type” se selecciona “string”, se escribe el valor del atributo en el cuadro y se pulsa aceptar.

Así quedaría instanciada la “CS214001-0001”, ahora habría que repetir estos pasos tantas veces como instancias de se quieran hacer.

Figura 3.26 Resultado de la instanciación del individuo "CS214001-0001"

Por último, se va a declarar una relación entre partes. Para eso, se va a instanciar un “Part BOM”. En este caso, se le va a llamar “CS214001-0001-CS214001-0101”, puesto que es como se han definido las dos partes que se van a relacionar.

En “Description”, se clica en “Add Types”, se selecciona “Part BOM” y se pulsa aceptar.

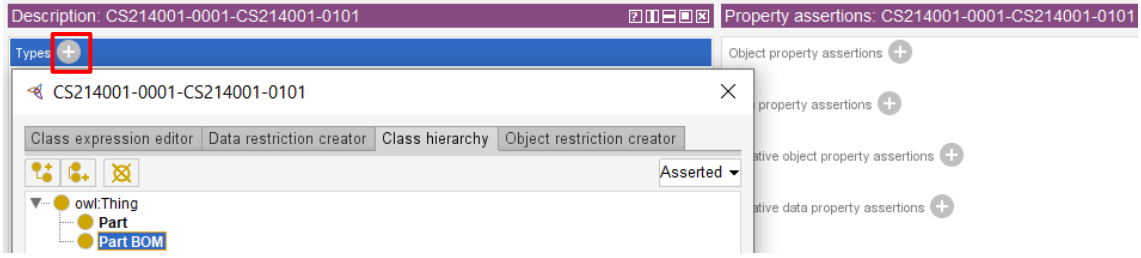

Figura 3.27 Selección de la clase del individuo “CS214001-0001-CS214001-0101”

Puesto que es una relación, habrá que añadir como “object properties” los objetos relacionados, en este caso la “CS214001-0001” y la “CS214001-0101”, además del atributo de la relación.

Para ello, se va a “Property assertions” y se clica en “Add Object property assertions”. En “Object property name” se escribe “ParteDestino” y en “Individual name”, “CS214001-0001”.

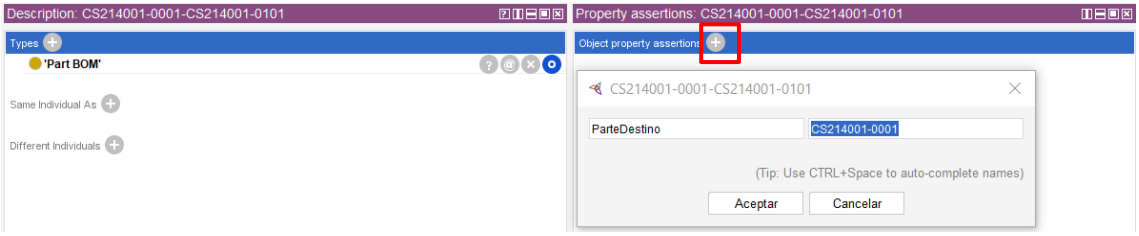

Figura 3.28 Caracterización del individuo "CS214001-0001-CS214001-0101" a través de propiedades objeto

Ahora se añade la otra "object property" escribiendo "ParteOrigen" en "Object property name" y "CS214001-0101" en "Individual name".

Para terminar, se añade el atributo de la relación. En este caso, como en la CS214001-0001 sólo hay una unidad de la CS214001-0101, se clica en "Add Data property assertions", se señala "quantity", en el "type" se selecciona "int" y se escribe "1" en el cuadro.

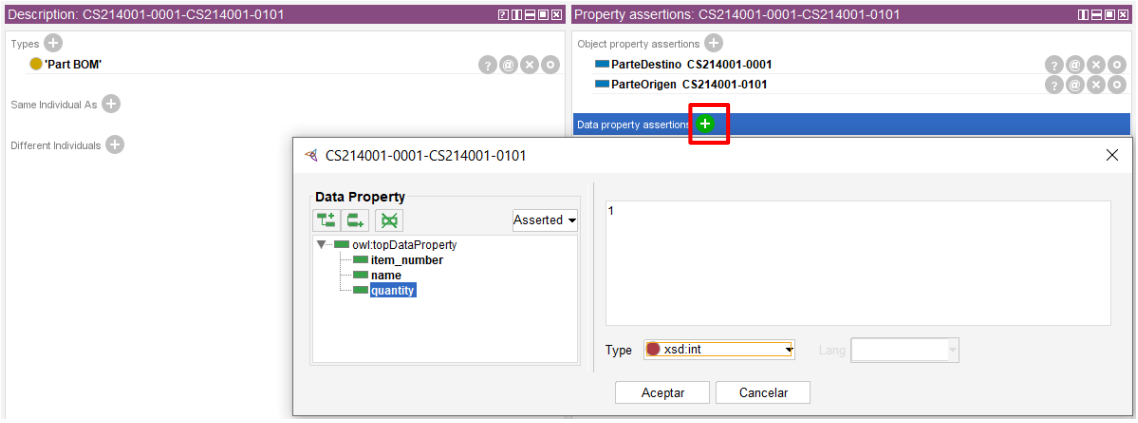

Figura 3.29 Caracterización del individuo "CS214001-0001-CS214001-0101" a través de propiedades dato

Ahora, simplemente hay que repetir estos pasos tantas veces como relaciones se quieran declarar.

En este caso, se han instanciado un total de cinco partes y cuatro relaciones, quedando los nueve individuos que se ven a continuación:

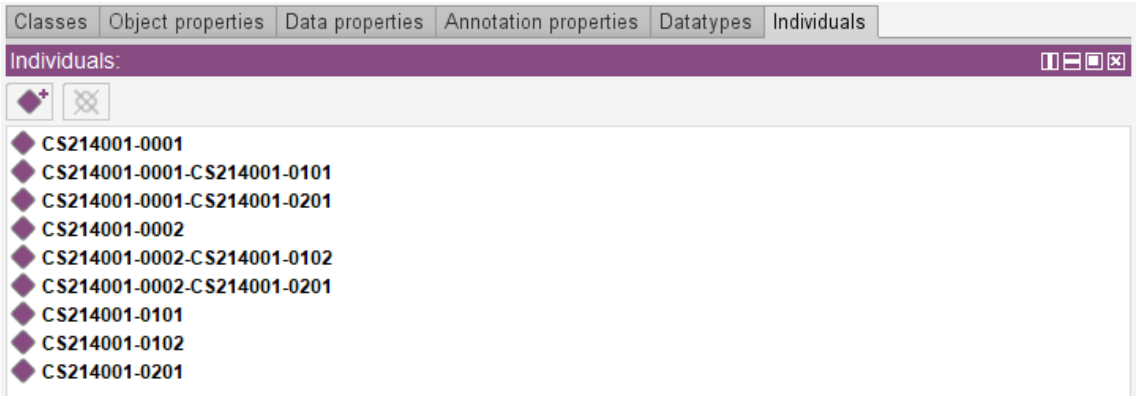

Figura 3.30 Resultado de la instanciación de la ontología

Si se vuelve a la pestaña de las clases, se verá cómo han aparecido las instancias en cada una de ellas.

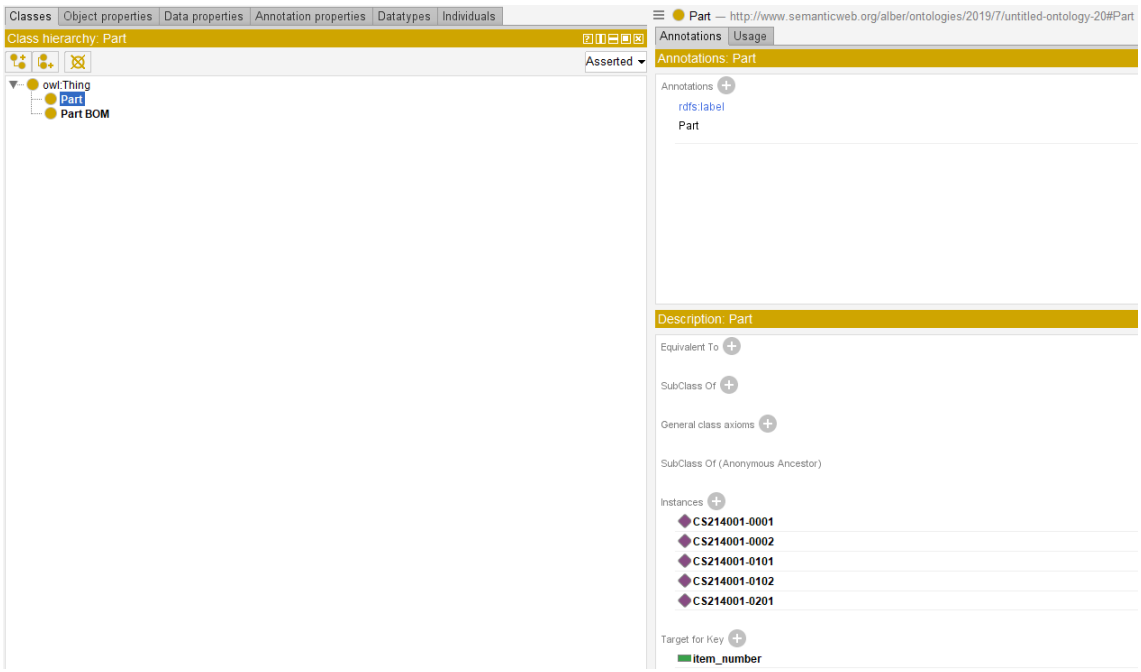

Figura 3.31 Resultado de la clase "Part" con sus miembros instanciados

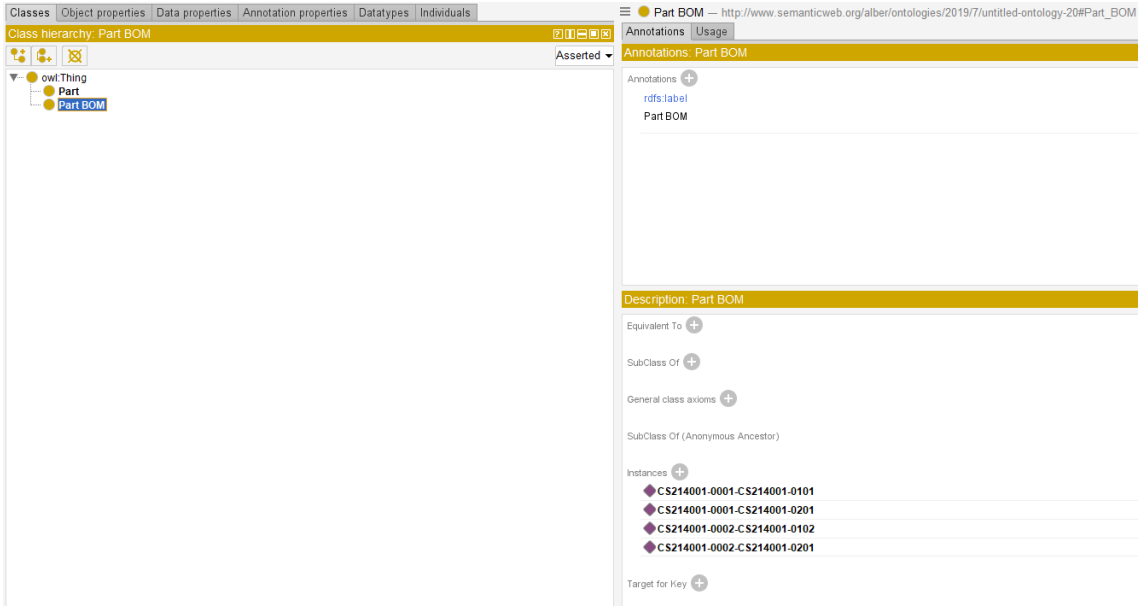

Figura 3.32 Resultado de la clase "Part BOM" con sus miembros instanciados

Una vez hecho esto, se puede exportar haciendo clic en “Save As” y eligiendo el formato “RDF/XML Syntax”.

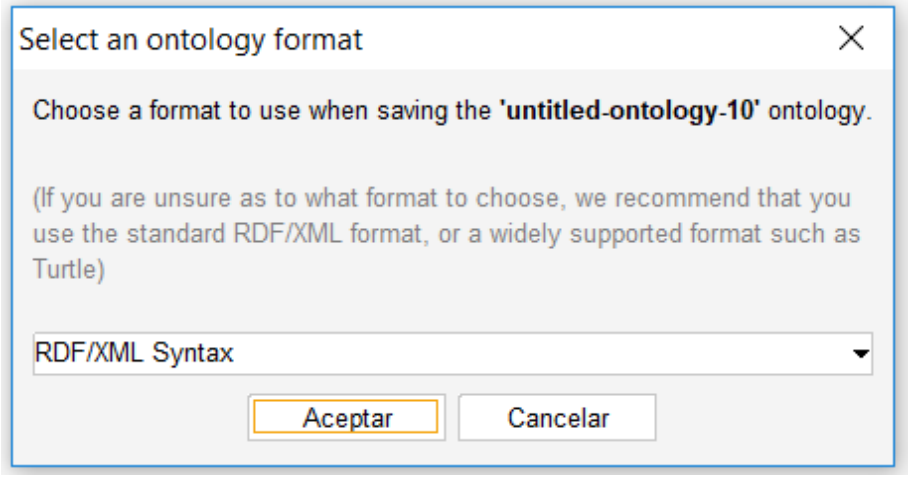

Figura 3.33 Exportación de la ontología desde Protege a un formato XML

3.4.3 ¿Cómo interpreter una ontología desde un XML?

Web Ontology Language OWL es un lenguaje de marcado semántico para publicar y compartir ontologías en la World Wide Web. OWL se desarrolla como una extensión de vocabulario de RDF (Marco de descripción de recursos) y se deriva del lenguaje de ontología web DAML + OIL.

La herramienta Protege permite guardar la ontología en diferentes formatos, incluido la extensión OWL. El formato OWL es un formato neutro (fichero de texto marcado XML) que permite transmitir la información.

Por ese motivo, el último paso del punto anterior ha sido guardar la ontología como XML y lo que se va a mostrar ahora es cómo leerlo. Al ser un fichero texto con marcado basado en XML se puede visualizar en cualquier editor de texto, en este proyecto se selecciona la aplicación Notepad++[xxxv] por su versatibilidad y capacidad de interpretar diferentes lenguajes de programación y marcado.

Además de reconocer la estructura XML permite expandir y contraer nodos facilitando la compresión y análisis de contenido.

Y también hace más fácil su interpretación porque detecta de qué tipo es cada elemento del fichero y lo muestra de un color u otro según la siguiente correlación:

Tipo	Color
Comentario	Verde
Nombre elemento	Azul
Valor elemento	Negro
Nombre atributo	Rojo
Valor atributo	Morado

Tabla 3.2 Correlación entre el tipo y el color de un elemento en Notepad++

El fichero XML siempre se compone de una declaración y el nodo raíz y dentro de este nodo se van a encontrar cuatro partes bien diferenciadas. Es más, Protege genera un nodo comentario al inicio de cada una de las partas a modo de título.

Tal y como se ha dicho, el fichero empieza con la declaración XML. Es fácil reconocerla porque, además de ser la primera línea, se presenta entre símbolos de cierre de interrogación. Su función es mostrar la versión del documento, por eso el nombre del atributo es “versión” y el valor es “1.0”.

A la declaración le sigue el nodo raíz y todo lo que contiene. En este caso, se presenta un elemento raíz “RDF”

y una serie de atributos con los espacios de nombres a los que se hace referencia en el documento.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#"
     xml:base="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

Figura 3.34 Nodo raíz

A continuación, se van a ver el resto de nodos que aparecen como hijos de la raíz diferenciados en cuatro partes.

La primera parte son los “Object Properties”. En el ejemplo se crearon dos: “ParteDestino” y “ParteOrigen”.

Si se quisiera leer el primero, se tendría un “Object Property” llamado “ParteDestino” definido en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 con dos atributos y una etiqueta que lleva su nombre. El primer atributo sería el dominio, puesto que sólo aparece una vez, esta propiedad sólo tendría aplicable como dominio la clase “Part_BOM” definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 y el segundo atributo sería el rango, al igual que el primero, únicamente aparece una vez, por lo que esta propiedad sólo tendrá aplicable como rango la clase “Part” definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20.

Esto quiere decir que si se instancia un individuo que lleve asociada esta propiedad, dicho individuo será miembro de la clase “Part_BOM” y el valor de dicha propiedad será un miembro de la clase “Part”.

```
    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->



    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#ParteDestino">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part_BOM"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part"/>
        <rdfs:label>ParteDestino</rdfs:label>
    </owl:ObjectProperty>



    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#ParteOrigen">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part_BOM"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part"/>
        <rdfs:label>ParteOrigen</rdfs:label>
    </owl:ObjectProperty>
```

Figura 3.35 Nodos de propiedades objeto

La segunda parte son los “Data Properties”. En el ejemplo se crearon tres: “ítem_numer”, “name” y “quantity”.

La lectura de uno de estos nodos sería prácticamente igual que en la parte anterior. Si se quisiera leer el primero se tendría un “Datatype Property” llamado “ítem_number” definido en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 con tres atributos y una etiqueta que lleva su nombre. El primer atributo sería el tipo, en este caso se indica que la propiedad es funcional y esa característica está definida en http://www.w3.org/2002/07/owl, el segundo atributo sería el dominio, puesto que sólo aparece una vez, esta propiedad sólo tendría aplicable como dominio la clase “Part” definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 y el tercer atributo sería el rango, al igual que el segundo, únicamente aparece una vez, por lo que esta propiedad sólo tendrá aplicable como rango la clase “string” definida en http://www.w3.org/2001/XMLSchema.

Esto quiere decir que si se instancia un individuo que lleve asociada esta propiedad, dicho individuo será miembro de la clase “Part”, el valor de dicha propiedad será una instancia de la clase “string”, es decir, una cadena de caracteres y, como es funcional, dicho individuo sólo podrá tener asociado un único valor a través de esta propiedad.

Sería interesante hacer un pequeño paréntesis en este punto para hablar de los prefijos. Nótese que los elementos de los “Data properties” tienen un prefijo “owl” y el atributo de estos elementos, un prefijo “rdf”. Esto quiere decir que un “Datatype Property” existe en el vocabulario de OWL, pero que el “Datatype Property” concreto que se ha creado aquí está definido en un espacio de nombres y para poder hacer referencia a dicho espacio de nombres se necesita hacer uso del vocabulario RDF. Lo mismo pasa con el resto de elementos, pudiéndose

apreciar que la propiedad tipo forma parte del vocabulario RDF mientras que dominio, rango y etiqueta son parte del vocabulario RDF Schema.

Con este paréntesis lo único que se ha querido hacer es volver al inicio del proyecto en el que se hablaba de cómo se había llegado al OWL y mostrar de forma ilustrativa la evolución que se ha seguido con algunas características concretas.

```
<!--
///////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
///////////////////////////////////////////////////////////////////////////////////
 -->



<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#item_number">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>item_number</rdfs:label>
</owl:DatatypeProperty>



 <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#name">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>name</rdfs:label>
</owl:DatatypeProperty>



<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#quantity">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part_BOM"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:label>quantity</rdfs:label>
</owl:DatatypeProperty>
```

Figura 3.36 Nodos de propiedades dato

La tercera parte son las “Classes”. En el ejemplo se crearon “Part” para instanciar individuos y “Part_BOM” para instanciar relaciones entre los individuos de la clase “Part”.

La lectura es sencilla. Si se quisiera leer la primera se tendría una “Class” llamada “Part” definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 con un atributo clave y una etiqueta que lleva su nombre. El atributo clave sería “ítem_number” y estaría definido en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20, y vendría a decir que no puede existir más de un individuo de la clase “Part” con el mismo “ítem_number”.

```
<!--
///////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////
 -->



<owl:Class rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part">
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#item_number"/>
    </owl:hasKey>
    <rdfs:label>Part</rdfs:label>
</owl:Class>



<owl:Class rdf:about="http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20#Part_BOM">
    <rdfs:label>Part BOM</rdfs:label>
</owl:Class>
```

Figura 3.37 Nodos de clases

La cuarta parte son los “Individuals”. En el ejemplo se instanciaron cinco miembros de la clase “Part” y cuatro relaciones “Part_BOM” entre individuos de la clase “Part”.

Si se quisiera leer el primero se tendría un “NamedIndividual” llamado “CS214001-0001” definido en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 con tres atributos y una etiqueta que lleva su nombre. El primer atributo sería el tipo, en este caso se indica que el individuo es miembro de la clase “Part” y que esa clase se encuentra definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20, y el resto de atributos, que son de tipo string, serían “ítem_number” cuyo valor sería “CS214001-0001”, “name” cuyo valor sería “Front Spar Assy” y “label” cuyo valor sería una concatenación de los valores de los atributos clave de esta clase, que en este caso únicamente iba a ser “ítem_number”.

Si se quisiera leer la primera relación, habría que ir a la cuarta instancia, donde se tendría un "NamedIndividual" llamado "CS214001-0001-CS214001-0101" definido en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20 con cuatro atributos y una etiqueta que lleva su nombre. El primer atributo sería el tipo, en este caso se indica que el individuo es miembro de la clase "Part_BOM" y que esa clase se encuentra definida en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20, el segundo y el tercero son "ParteDestino" y "ParteOrigen", respectivamente. En realidad, estos no son atributos de la relación, sino los individuos que están relacionados por eso aparecen en el valor del atributo del nodo y no en el valor del propio nodo.

Esta es una diferencia importante entre los "Object properties" y los "Data properties". Para poder crear una relación entre individuos, primero hay que crear los individuos. Por este motivo, cuando se instancia la relación, el valor del "Object property" es una referencia directa a ese individuo. Por el contrario, para poder crear un individuo lo único que hay que hacer es caracterizarlo con "Data properties" y los valores de éstos pueden ser una cadena de caracteres alfanuméricos cualesquiera, por eso éstos sí aparecen en el XML como valores del nodo, porque son los verdaderos atributos del individuo.

Como se estaba diciendo, en este caso, se tendría como "ParteDestino" el "CS214001-0001" y como "ParteOrigen" el "CS214001-0101" y ambos estarían definidos en http://www.semanticweb.org/alber/ontologies/2019/7/untitled-ontology-20. Nótese, que se ha tomado una perspectiva ascendente, es decir, se está relacionando el hijo con el padre con el objetivo de saber en qué conjunto superior y en qué cantidad monta una elemental.

Para terminar, el cuarto atributo, que como ya se ha dicho, sería el único atributo real de la relación es "quantity", sería de tipo "int" y tendría un valor de una unidad.

Figura 3.38 Nodos de individuos instanciados

4 ONTOLOGÍAS EN ENTORNOS DE FABRICACIÓN

Como se ha comentado anteriormente, los sistemas PLM pueden integrar toda la información relevante a lo largo del ciclo de vida del producto, de manera que sirva de feedback para nuevos productos, sobretodo en los aspectos relacionados con diseño y fabricación. Por este motivo, hay que ser consciente del valor de dicha información y de la importancia que tiene su correcta transmisión.

La idea es generar, integrar y gestionar datos para transformarlos en conocimiento que pueda ser reutilizado. Es más, se podrían usar ontologías, para establecer relaciones entre características de diseño y fabricación para verificar que todo está correcto, como pueden ser discrepancias entre notas y cotas del modelo (hay una nota que dice que una zona de masa tiene diámetro x pero en el modelo mide y), tolerancias difíciles de conseguir (hay una tolerancia muy pequeña de posicionamiento de taladro referenciado a un datum que está a una distancia excesiva), dimensiones que no cumplen con la normativa aplicables (alojamiento de una rótula cuyas dimensiones no están según la norma)… En definitva, analizar datos y ser capaz de darles significado para poder tomar decisiones.

Los diseños se realizan a base de iteraciones y, por ello, aunque se revisan, y una vez cerrados, pasan por muchas manos que lo supervisan antes de firmarlos, no es difícil que se produzcan liberaciones con alguna errata.

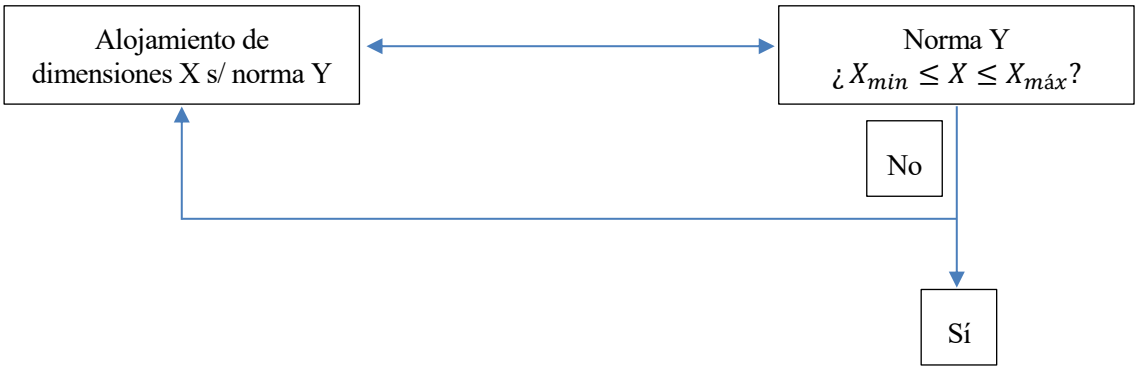

Figura 4.1 Ejemplo de consulta sobre características geométricas

Las ontologías pueden definir formalmente conceptos, controlar su uso, capturar conocimiento y proporcionar una ruta para compartir información a través del diseño y la fabricación. Ofrecen mejor capacidad de razonamiento comparada con las bases de datos de forma y formato fijo puesto que mientras que las ontologías se centran en preservar el significado para facilitar la interoperabilidad, las bases de datos se centran en almacenar. Por tanto, el objetivo es buscar una ontología de fabricación precisa y rigurosamente definida como una base semántica común. Es decir, hay que definir la terminología usada para describir el sistema, pero también hay que enriquecer dicha terminología con significado derivado de relaciones con otros conceptos.

Una característica esencial de las ontologías es la modularidad. Un modelo rígido e inflexible no se puede reciclar y eso conlleva que el conocimiento adquirido en ese dominio no se pueda reutilizar.

En la siguiente figura se presenta una ontología de fabricación en la que se ven claramente los módulos que la componen y la facilidad que supondría añadir, modificar o eliminar dichos módulos. Por eso se dice que las ontologías son de naturaleza extensible.

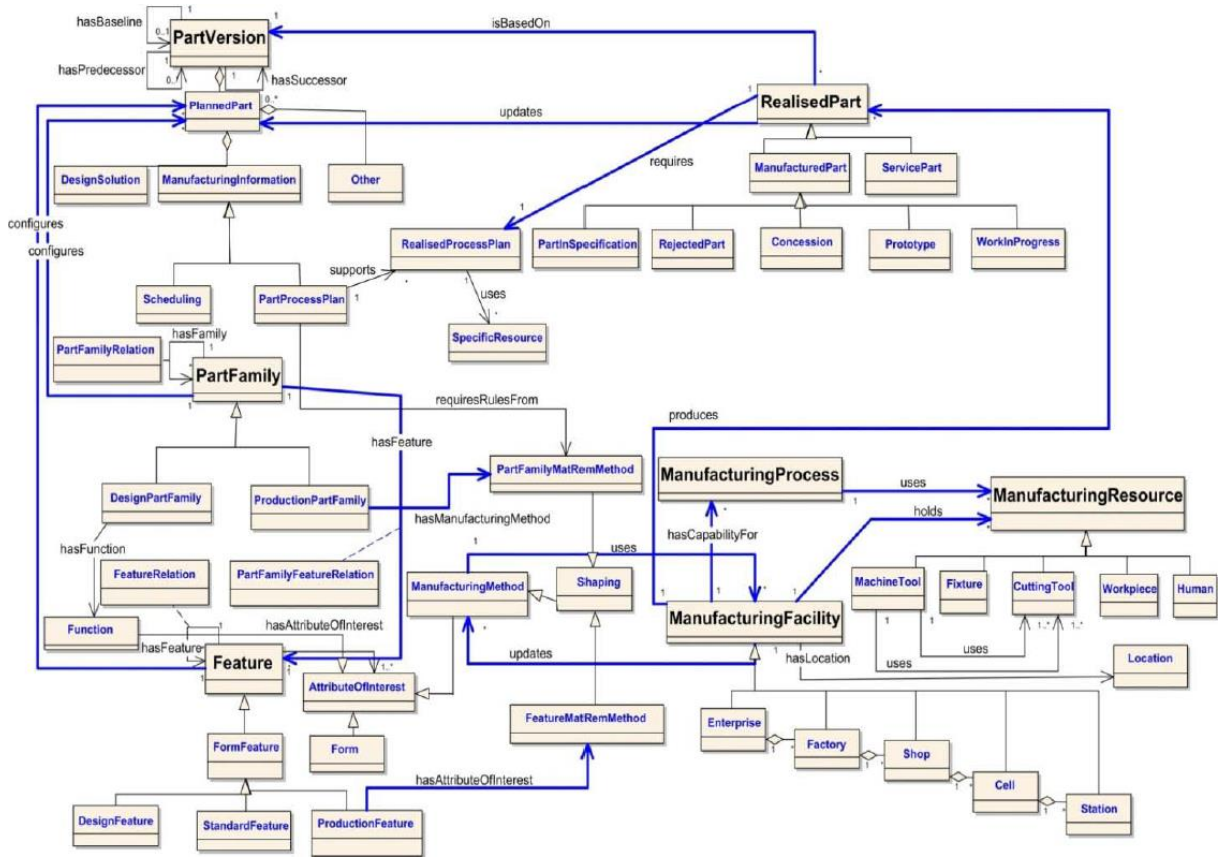

Figura 4.2 Ejemplo de ontología de fabricación[xxxvi]

En este ejemplo, a partir de una definición de componente o parte (Part Version) se ha ido extendiendo la ontología a familia de productos cada una con sus respectivas características y, posteriormente, ha ido creciendo mediante la incorporación del dominio de fabricación y la definición del proceso (Manufacturing Process).

4.1 Dominio PPR

El dominio PPR es el del entorno de la fabricación y está formado a su vez por tres dominios que se describen a continuación:

4.1.1 Dominio producto

El dominio del producto está enfocado a los diferentes estados de maduración por los que va a pasar un producto que está en continuo cambio, desde la fase conceptual a la liberación del mismo.

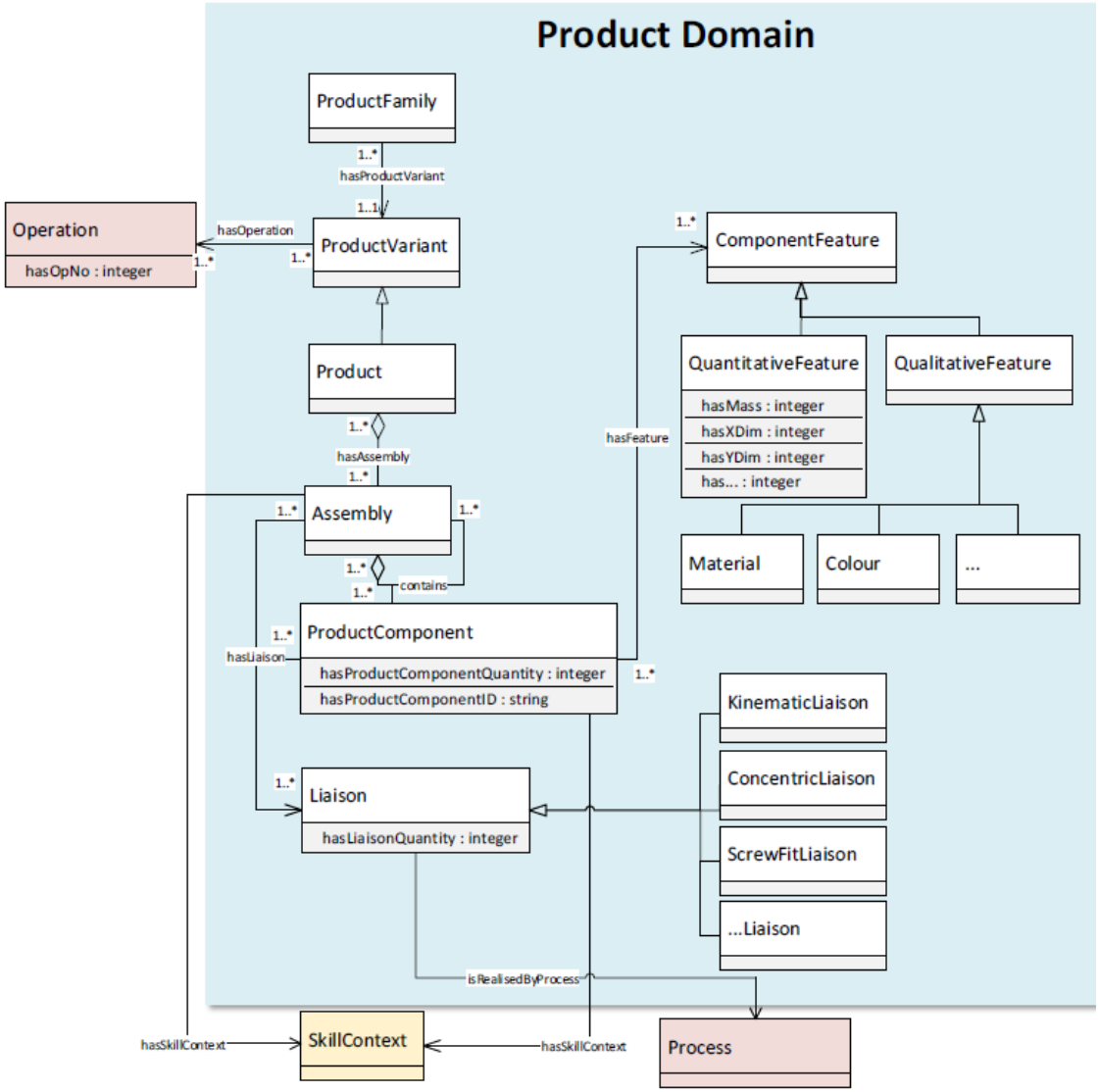

Figura 4.3 Dominio del producto[xxxvii]

En este caso, se ve como las familias de productos tienen variantes y cada una de ellas define un producto concreto, que a su vez puede estar formado por una serie de componentes ensamblados y tanto los componentes como el proceso de ensamblaje pueden tener características o restricciones asociadas.

4.1.2 Dominio proceso

El dominio del proceso está más enfocado al flujo de fabricación que se debe seguir para conseguir ese producto.

Por una parte, el dominio del proceso debe estar conectado con el dominio del producto para que en fases de diseño se tengan en cuenta las restricciones que van a permitir que un producto sea fabricable. Por ello, debe haber una interconexión entre ambos dominios que permita liberar el diseño de un producto tras recibir feedback desde fabricación.

Por otra parte, debe estar conectado con el dominio de los recursos porque para establecer una planificación, no basta con la información del producto y el conocimiento de los procesos, también se necesita saber qué recursos se tienen disponibles.

El hecho de relacionar el dominio del producto con el del proceso establecería que un producto específico es realizado por un proceso específico, esa sería la manera de generar conocimiento. Por ejemplo, si un mecanizado de acero requiere un alivio de tensiones cuando el material tiene una resistencia superior a x MPa y se establece esa relación entre producto y proceso, se está generando conocimiento porque cada vez que se tenga un mecanizado de acero, automáticamente se va a saber si requiere alivio de tensiones o no. A través una característica cuantitativa como la resistencia mecánica del material se puede consultar una característica

cualitativa del producto.

Exactamente lo mismo pasaría si se relaciona el dominio del proceso con el del recurso. Por ejemplo, si en un proceso de ensamblaje se requiere apretar un número de tuercas de manera simultánea y se tiene una máquina capaz de apretar x tuercas, sabiendo el número de tuercas que tiene un conjunto se va a saber si es posible montarlo o no con los recursos actuales.

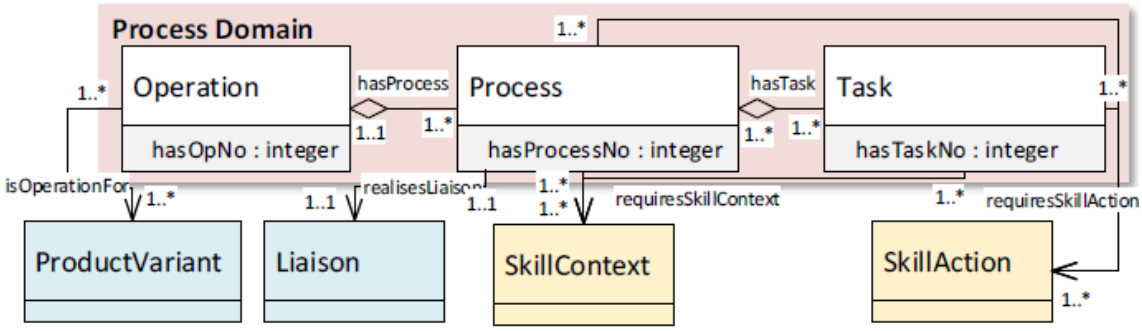

Figura 4.4 Dominio del proceso[xxxvii]

En este caso, se ve como la orden de producción se compone de operaciones y dichas operaciones pueden tener asociados procesos, que a su vez se pueden dividir en tareas.

4.1.3 Dominio recurso

Por último, el dominio del recurso está orientado a analizar los requerimientos del producto y el proceso y comprobar si se tienen recursos capaces de llevarlos a cabo.

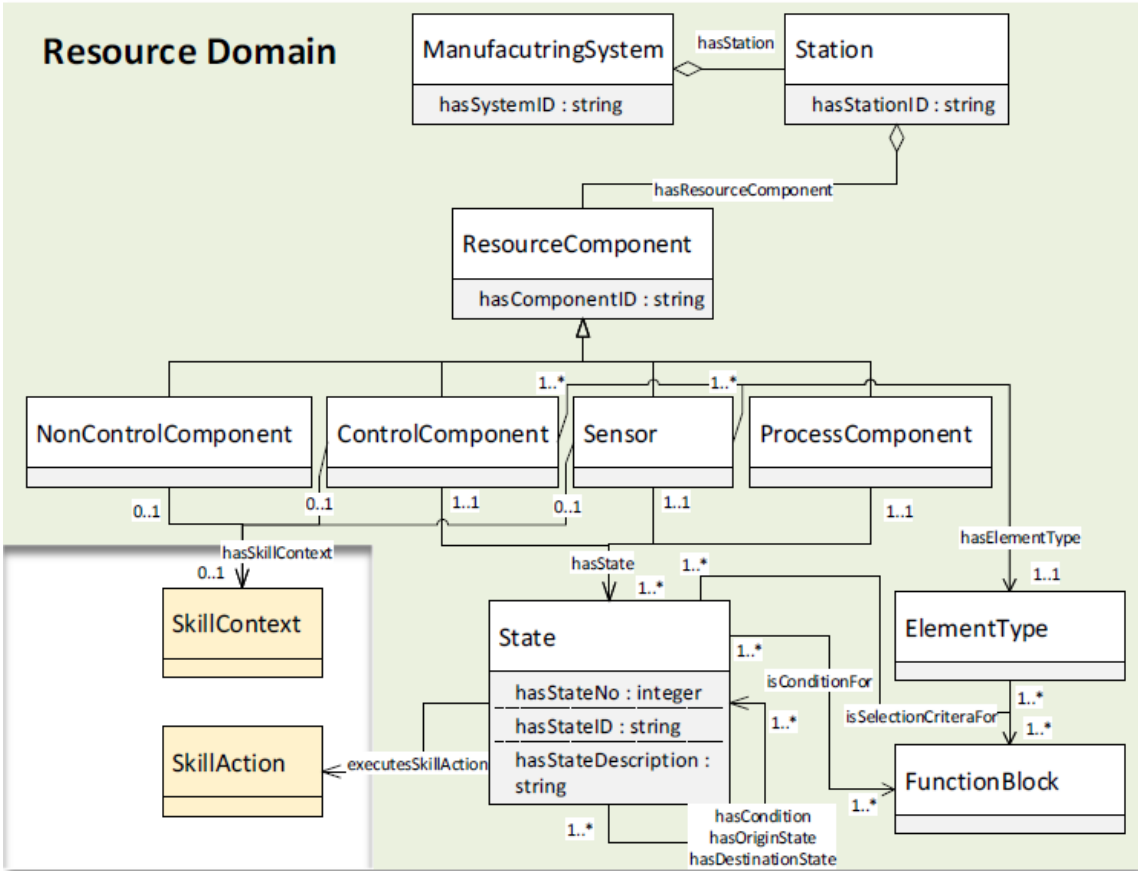

Figura 4.5 Dominio del recurso[xxxvii]

Por último, en este caso se ve como el sistema de fabricación está estructurado en estaciones y en cada una de ellas hay una serie de recursos asignados.

5 ONTOLOGÍA ORIENTADA AL ENSAMBLAJE

Aquí la clave es adaptar el dominio PPR al sistema de ensamblaje.

El producto será el elmento final, pero para llegar a él en el leadtime establecido será necesario planificar la secuencia de ensamblaje, definiendo el orden de los procesos y los recursos que deben ser asginados para llevarlos a cabo, y equilibrar dicha secuencia para tener bien distribuida la carga de trabajo y evitar así la aparición de cuellos de botella.

En la siguiente figura se muestra el modelo de datos empleado en este proyecto y cuya ontología se desea definir y, posteriormente, instanciar.

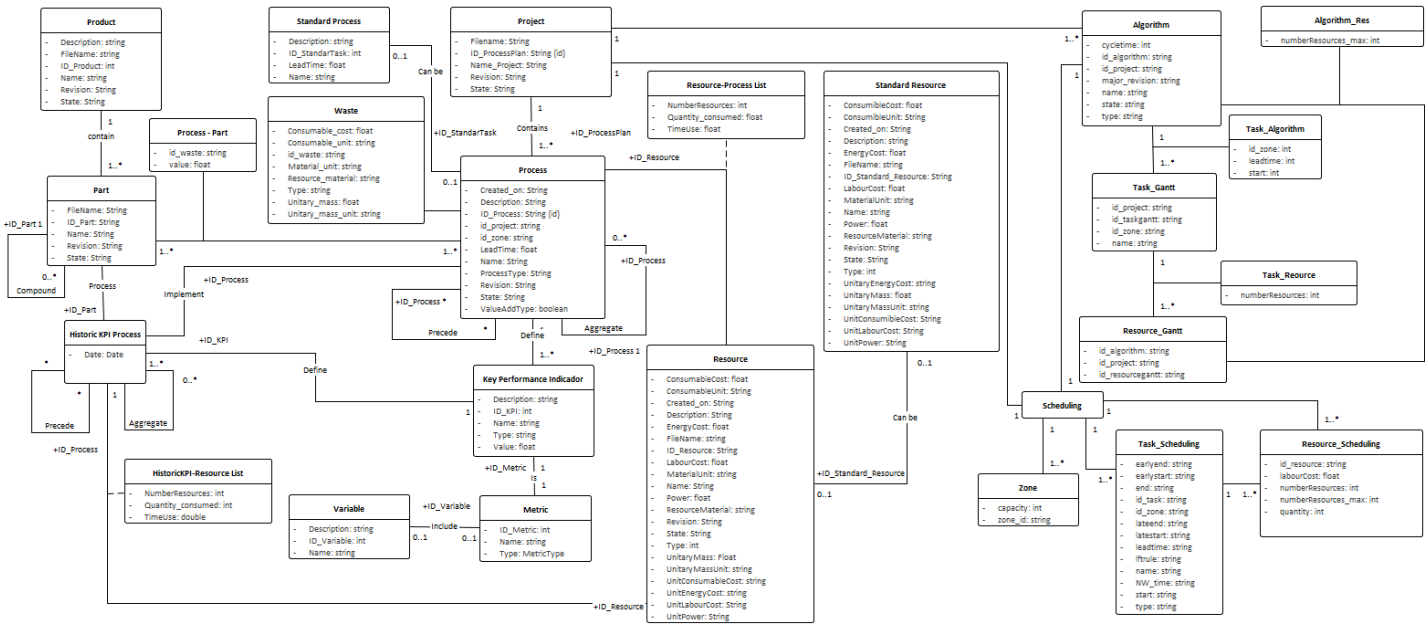

Figura 5.1 Modelo de datos empleado en este proyecto

Para demostrar el funcionamiento de la herramienta desarrollada y el propósito de este TFM se ha simplificado esta ontología, aprovechando la característica de modularidad de la que se hablaba en capítulos anteriores y se ha creado una propia con cuatro clases interrelacionadas entre sí.

Como elementos principales se van a tener proyectos, que serán de clase "Mpp_project"; procesos, que serán de clase "sus_mpp_Process", recursos, que serán de clase "sus_mpp_Resource" y productos, que serán de clase "Part".

De este modo, se podría crear un proyecto por cada solución de fabricación que se elaborara en cada una de las estaciones de la línea incluyendo todos los procesos que sean requeridos.

Es decir, se crearía un proyecto compuesto de una serie de operaciones que convierten el elemento que entra en la estación en el que sale de ella. Además, dichas operaciones se podrían desglosar en tareas de manera que se pudiera llegar a tener tanto nivel de detalle como se quisiera.

Por último, estarían los recursos. Cada una de las operaciones que transforman el producto, independientemente de si añaden valor o no, requieren una serie de recursos para poder realizarse, por lo que también habrá que tenerlos en cuenta en la creación del proyecto.

En el caso de una línea final de ensamblaje, existirá un único producto final, el entregable, de modo que un proyecto en sí no va a tener asignado un componente como tal salvo que se defina un semiproducto que sea el producto final en el estado en el que sale de la estación de cada proyecto.

En el caso de una fábrica de ensamblaje de componentes estructurales si puede tener más sentido el asignar un producto a la estación porque el taller puede recibir elementales que requieran un preequipado antes de meterse en la grada de montaje. Aquí, sí existiría un producto identificado que se pudiera asociar al proyecto. Por ejemplo, se podrían recibir elementales de chapa que hubiera que equipar con tuercas remachables e identificar como conjunto o se podrían recibir mecanizados que pidieran la instalación de casquillos o rótulas y la identificación como conjunto. En este caso, este planteamiento sería muy útil porque podría haber operaciones o tareas dentro de operaciones que no se pudieran realizar en las instalaciones. Por ejemplo, podría darse el caso de que hubiera que instalar rótulas, pero fueran tan pocas que a la empresa no le rentara certificarse para dicho proceso y prefiriera subcontratarlo o podría darse el caso de que hubiera que instalar casquillos y se estuviera certificado para el proceso de encasquillado pero en casos concretos, en los que el componente pueda entrar en contacto con fluido hidráulico, se pidiera proteger el cordón de sellante, que se aplica sobre la junta que queda en la unión entre ambos elementos para asegura la estanqueidad, con una imprimación especial según una especificación diferente para la que no se está certificado. Es decir, se podría estar certificado para el proceso, pero si dicho proceso es una operación que requiere tareas previas y posteriores al encasquillado como pueden ser la limpieza previa de los elementos a unir, el sellado y la protección del sellante con imprimación, podría darse el caso de que no se estuviera certificado para todas las tareas que requiere la operación. Por este motivo, parece interesante poder saber de qué procesos se compone cada operación, cuáles son especiales, dónde se realizan, si es en interno o en un proveedor externo, y si se está certificado en caso de ser necesario.

Como ya se sabe, una ontología define un dominio compuesto de clases que pueden estar interrelacionadas entre sí. Ya se sabe cuáles son las clases que se van a definir en este supuesto, pero ¿cómo se van a relacionar entre ellas?

Se ha dicho que en el caso de la línea de ensamblaje cada proceso estará asociado a una estación por lo que más que relacionar componentes con procesos, habrá que definir procesos en base a proyectos. Nótese que se puede tener una misma operación en dos estaciones, pero con un leadtime diferente por ser para elementos distintos, de ahí la importancia de definir las operaciones en base a los proyectos. En otros casos, como ya se ha explicado, si cobra importancia tener una relación entre componentes y procesos que podrá denominarse "sus_mpp_Process_part".

Puesto que se ha propuesto descomponer operaciones en tareas, se podría jerarquizar y crear una subclase de procesos, pero finalmente se ha optado por crear una clase que relaciona todas las tareas que componen una operación y se le ha denominado "sus_mpp_Process_aggregate".

También se ha tenido en cuenta el orden y se ha establecido una relación entre procesos, denominada "sus_mpp_Process_precedence", que indica qué tareas no se pueden iniciar hasta que no se terminen otras que deben ser previas a ella.

Por último, quedarían por relacionar los procesos con los recursos, es decir, tener en cuenta qué va a ser necesario para llevar a cabo un proceso y asignárselo. Para ello, se ha establecido una relación entre procesos y recursos denominada "sus_mpp_Process_resource".

A continuación, se muestra una representación gráfica de la ontología y más adelante se explica detalladamente la definición del dominio.

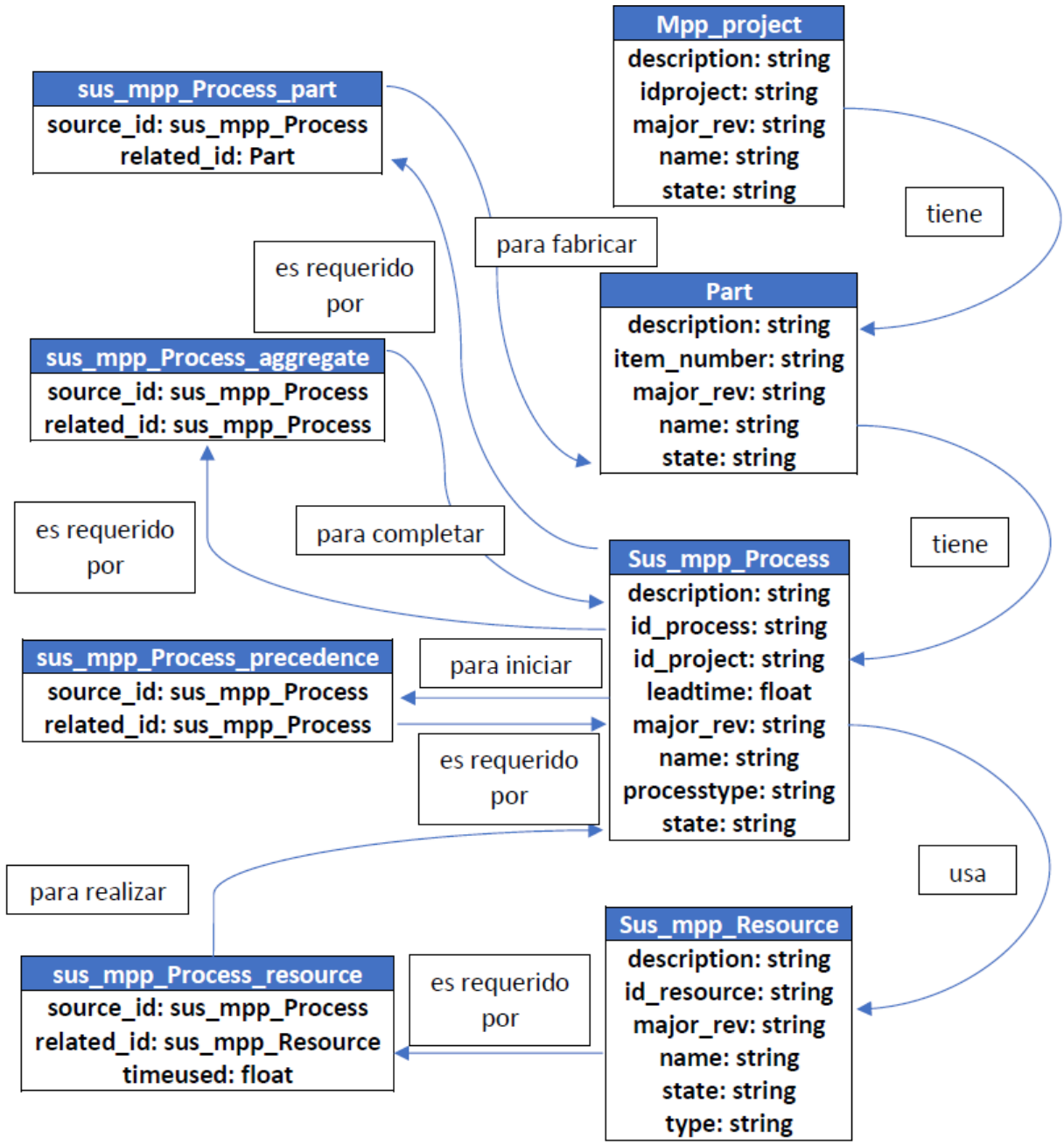

Figura 5.2 Representación gráfica de la ontología empleada en este proyecto

5.1 Clases de individuos

Ahora se detalla la definición de cada una de las clases de individuos.

5.1.1 Mpp_Project

Se va a definir una clase "Mpp_project" con cinco atributos de tipo "string". Dichos atributos serán "description", "idproject", "major_rev", "name" y "state". De los cinco, serán atributos clave "idproject",

"major_rev" y "state", de manera que pueda haber proyectos con las mismas descripciones o los mismos nombres, incluso puede haber proyectos con el mismo "idproject", "major_rev" o "state", pero al ser estos tres últimos los atributos clave, lo que no puede haber es proyectos con el mismo trío de valores. Es decir, si hay un proyecto con "idproject" = 01, "major_rev" = A y "state" = In work, para que exista otro proyecto con "idproject" = 01 y "state" = In work, tienen que tener una "major rev" ≠ A.

5.1.2 Part

Como en el caso de "Mpp_project", se va a definir una clase "Part" con cinco atributos de tipo "string". Dichos atributos serán "description", "item_number", "major_rev", "name" y "state". De los cinco, serán atributos clave "ítem_number", "major_rev" y "state".

5.1.3 Sus_mpp_Process

Nótese que esta ontología está centrada en los procesos y por ello esta clase va a ser la más caracterizada. Aquí, se ha definido una clase "sus_mpp_Process" con ocho atributos, siete de tipo "string" como los son: "description", "id_process", "id_project", "major_rev", "name", "processtype" y "state"; y uno de tipo "float" como "leadtime". De los ocho, serían atributos clave "id_process", "id_project", "major_rev" y "state".

Ya se habló en el punto anterior de definir procesos en base a proyectos porque era la única manera de identificarlos por separado y poderles asociar un leadtime independiente. Por eso se ha puesto como atributo el "idproject". De esta manera, si en dos estaciones hay una operación de remachado, pero el número de remaches a dar en una es el triple que en la otra y, por tanto, el leadtime también, se pueden tener las dos operaciones diferenciadas y cada una tener su leadtime.

5.1.4 Sus_mpp_Resource

Por último, se ha creado una clase "sus_mpp_Resource" con seis atributos de tipo "string". Dichos atributos serán "description", "id_resource", "major_rev", "name", "state" y "type". De los seis, serán atributos clave "id_resource", "major_rev" y "state".

5.2 Clases de relaciones

Por último, se termina la definición del dominio con la descripción de las relaciones entre clases de individuos.

5.2.1 Sus_mpp_Process_part

Esta clase va a relacionar procesos con productos, de modo que cada producto quede asociado con todos los procesos que han hecho posible su transformación desde un estado inicial hasta un estado final. Puesto que la única intención de esta clase es que queden asociados, no se va a caracterizar con ningún atributo.

Las propiedades objeto serán "source_id" de tipo "sus_mpp_Process" y "related_id" de tipo "Part".

5.2.2 Sus_mpp_Process_aggregate

Esta clase se ha creado para relacionar una operación con todas las tareas que la componen. Puesto que, tanto las operaciones como las tareas van a ser procesos, esta clase relaciona procesos con procesos y como la única intención es asociarlas tampoco va a tener ningún atributo.

Las propiedades objeto serán "source_id" de tipo "sus_mpp_Process" y "related_id" de tipo "sus_mpp_Process".

5.2.3 Sus_mpp_Process_precedence

La clase "sus_mpp_Process_precedence" se ha creado para imponer una restricción, de manera que quede registrado cuáles son los procesos que exigen un orden, es decir, cuáles requieren la finalización de un proceso anterior para poder iniciarse. Como se ha dicho, simplemente es para imponer una restricción, por lo que lo

único que va a hacer esta clase es asociar procesos con procesos y tampoco se va a caracterizar con ningún atributo. Si bien es cierto, que en una ontología más orientada a la fabricación de elementales en la que el orden de las operaciones es fundamental para fijar el flujo, puede cobrar sentido añadir un atributo que indicara el tiempo máximo entre operaciones, puesto que se trata de un factor que hay que medir sobre todo en procesos finales o incluso en la propia fabricación en el caso de materiales compuestos en los que el tiempo de vida empieza a correr desde que el material preimpregnado sale del congelador y hay que controlarlo hasta que se va a curar en la estufa o el autoclave.

Las propiedades objeto serán "source_id" de tipo "sus_mpp_Process" y "related_id" de tipo "sus_mpp_Process".

5.2.4 Sus_mpp_Process_resource

Por último, se crea la clase "sus_mpp_Process_resource" para relacionar procesos con recursos. Esta parte es de vital importancia para la empresa porque es la que va a permitir realizar una planificación de recursos. Como puede verse, esta clase si tiene un atributo, el "timeused", que es de tipo "float". Este dato es muy importante para la planificación porque el hecho de que un proceso tenga un leadtime X no quiere decir que durante todo ese tiempo se requieran recursos para para realizar dicho proceso. Por ejemplo, en una operación de pintura se tendrá un leadtime X, pero sólo se requerirá un operario el tiempo que se esté preparando la mezcla y pintando, el resto del tiempo que necesita el elemento para secarse no se está requiriendo al operario. Más llamativo incluso puede ser un proceso de marcado en tinta, que en algunos casos se pide barnizar para evitar que la tinta pueda correrse y, consecuentemente, pueda perderse la identificación del elemento, donde el tiempo de marcado con pistola es prácticamente nulo, el de aplicar barniz también es insignificante, pero el tiempo de secado que necesita el barniz hace que el leadtime total de esta operación pueda llegar a un día. Por ello, es importante tener una variable que indique el tiempo que se va a necesitar tener asignado un recurso a un proceso para calcular cuántos recursos son realmente necesarios.

Finalmente, aquí las propiedades objeto serán "source_id" de tipo "sus_mpp_Process" y "related_id" de tipo "sus_mpp_Process".

6 DISEÑO DEL SISTEMA DE INSTANCIACIÓN

El diseño del sistema propuesto en este proyecto se representa en la siguiente figura. Se trata de un diagrama de paquetes de datos, distinguiendo cinco grandes paquetes con las dependencias que existen entre ellos a través de arcos de conexión.

El nucleo del diseño se encuentra en el "Instance system" este subsistema es el encargado de la lectura e interpretación de la ontología, así como de procesar las instancias entre los sistemas de información involucrados. Este trabajo se ha centrado en analizar la interoperabilidad entre los sistemas Protege y el sistema PLM Aras innovator.

El subsistema "Instance system" interpreta tanto la ontología como la instancia, que se encuentran en formato XML y lenguaje OWL. Para poder procesar ficheros XML se ha desarrollado una librería que se describe en el paquete "XML System Interface". Las interfaces de comunicación con herramientas externas se incluyen en dos paquetes. El paquete de "Aras System Interface" define las funciones de conexión e intercambio de información con el sistema PLM ARAS, mientras que "Protege Ontology and Instance System" define las interfaces con la aplicación Protege. Finalmente, se ha definido unas librerías para la gestión de datos, en concreto listas parametrizadas o plantilla de listas que son empleadas por cada paquete definido en el proyecto.

En este trabajo se ha optado por interoperar entre Protege, un editor de ontologías, y Aras, un sistema PLM, pero esto mismo se podría hacer con otras aplicaciones cualesquiera.

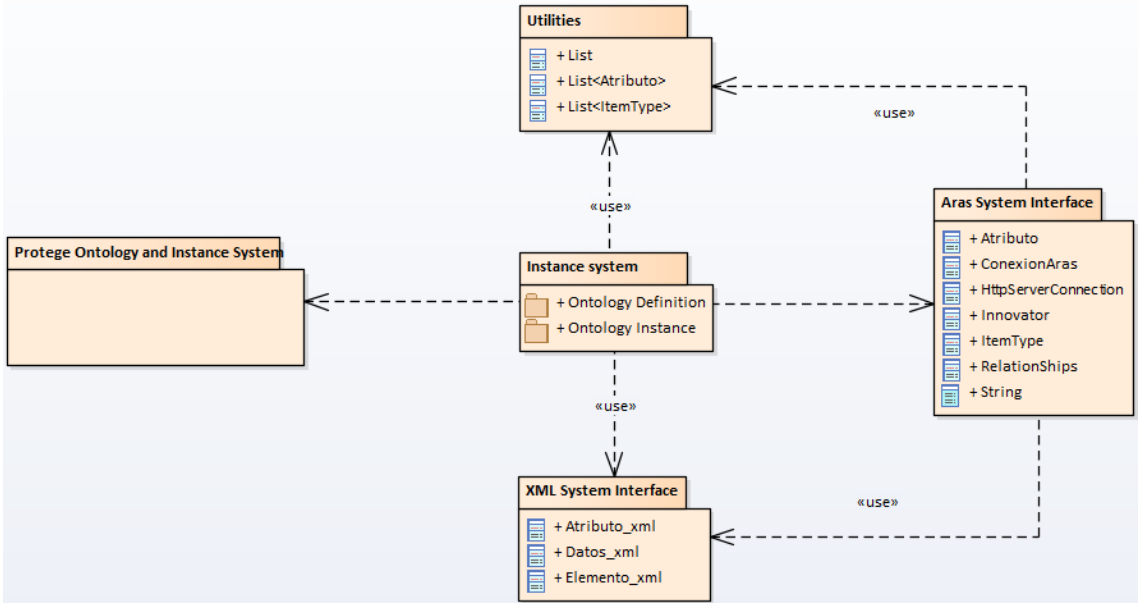

Figura 6.1 Representación de la solución adoptada en un diagrama de paquetes

Echando la vista atrás, en el primer capítulo se mostraba un esquema con la idea del proyeto que, obviamente, tiene una clara similitud con este. A continuación, se va a representar dicho esquema de nuevo, pero esta vez particularizando los sistemas como Protege y Aras.

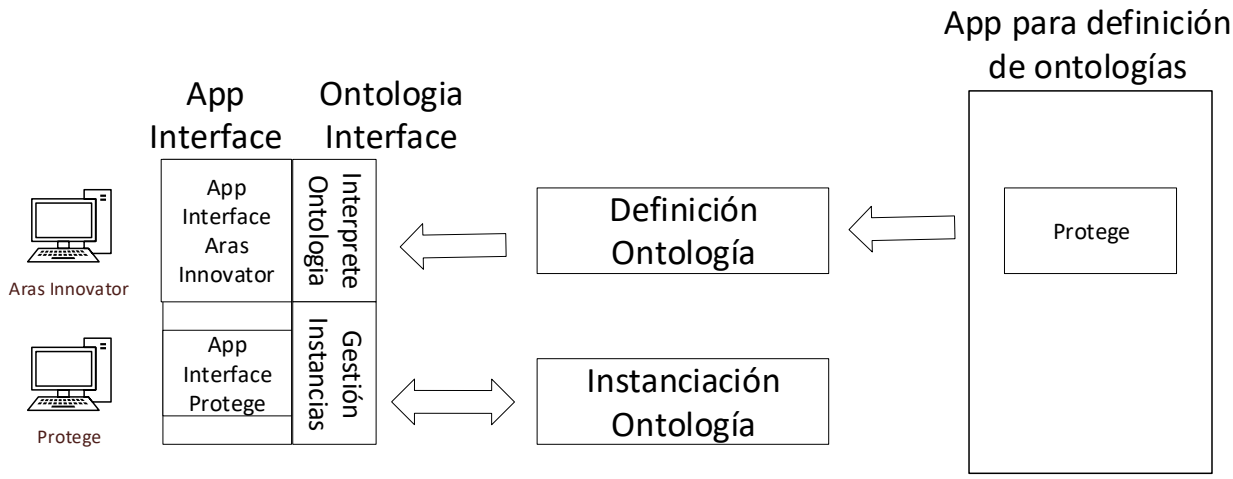

Figura 6.2 Arquitectura básica de instanciación de ontologías para Protege y Aras

Nótese que la idea es crear un sistema de instanciación de ontologías que sirva para que dos aplicaciones diferentes se comuniquen, pero para ello habrá que hacer uso de los recursos del lenguaje de programación que se vaya a utilizar, en este caso C#, y de un grupo de funciones que permitan interactuar con Aras y otro que permita comprender el formato XML en el que se va a transmitir la información.

Al fin y al cabo, la definición del dominio sólo es necsario hacerla la primera vez, aunque se puede modificar tantas como sea necesario. Con esto lo único que se quiere decir es que lo que se va a realizar en realidad es demostrar que es posible importar y exportar información de un sitio a otro.

El primer paso será la definición del dominio en Protege porque es un editor de ontologías, pero dado que desde ese editor se puede instanciar se va a tener la posibilidad no sólo de exportar la información que se quiera desde Aras sino también de importar a Aras la información que se quiera desde Protege.

6.1 Definición de la ontología en Protege

Una vez se tiene clara la definición del dominio, se puede generar la ontología a través de Protege como se ha

mostrado en capítulos anteriores.

6.1.1 Clases

Aquí se muestran las ocho clases que se han creado, cuatro para instanciar individuos y otras cuatro para relacionar los individuos instanciados.

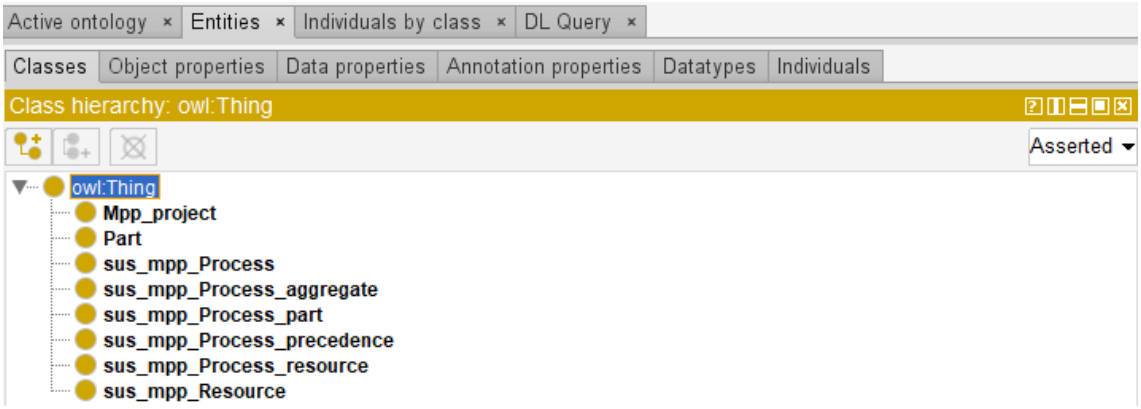

Figura 6.3 Definición de las clases para el caso práctico

La primera clase que aparece es “Mpp_project” y como se puede ver se le ha puesto una etiqueta con su nombre y se le han asignado los atributos clave, que previamente han sido definidos como “Data properties”. De la misma forma quedarían el resto de clases.

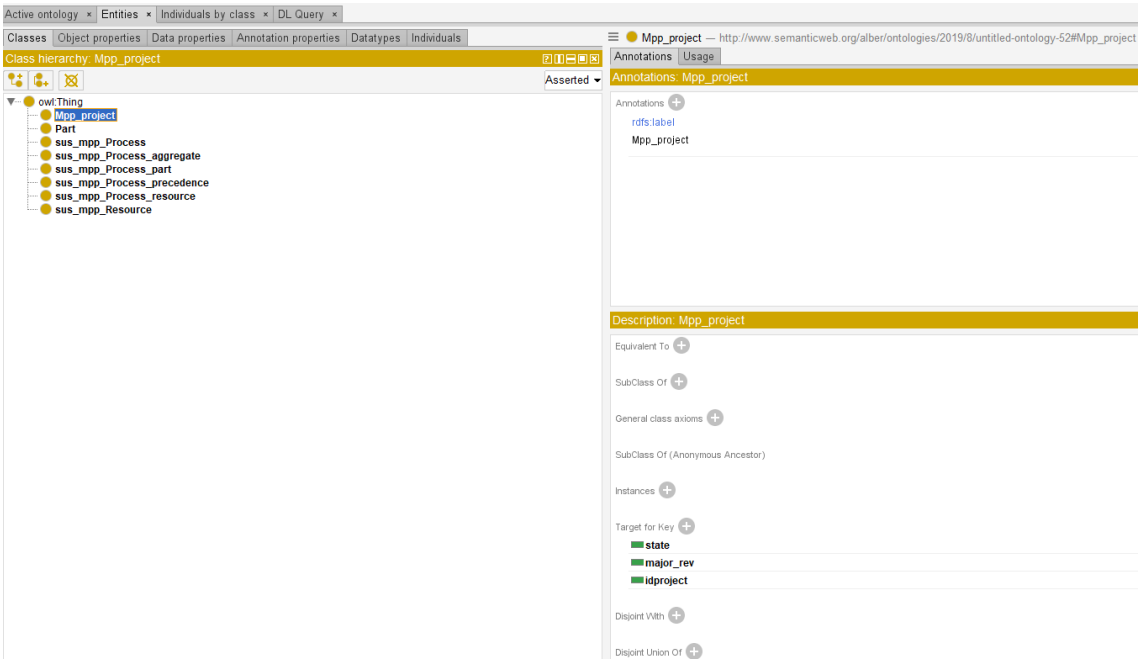

Figura 6.4 Muestra de la clase "Mpp_project" definida

6.1.2 Object properties

Ahora se muestran las “Object properties”. Aunque se han creado cuatro clases para relacionar individuos, todas ellas tienen la misma propiedad para el objeto origen y el objeto relacionado, por eso hay sólo hay dos y no ocho propiedades objeto. Para que esto quede bien definido, las propiedades no pueden tener como característica “Functional”, porque es la única forma de asegurar que un individuo pueda estar relacionado con varios a través de la misma propiedad.

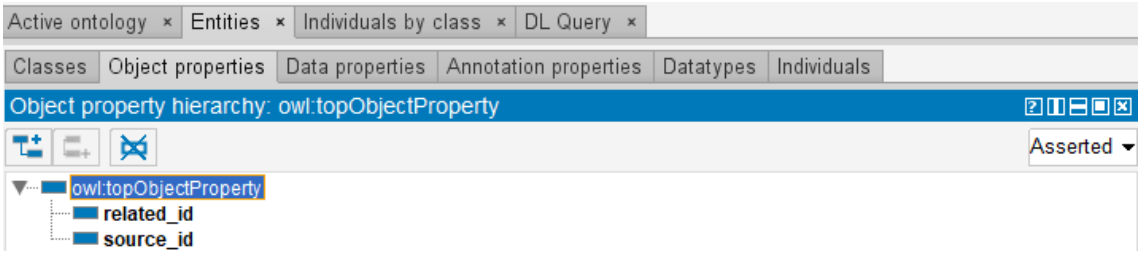

Figura 6.5 Definición de las propiedades objeto para el caso práctico

La primera sería “related_id” y como se puede ver se le ha puesto una etiqueta con su nombre y se ha definido su dominio y rango.

Puesto que esta propiedad va a aparecer en los cuatro tipos de clases de relaciones, su dominio estará compuesto por esos cuatro tipos y como según el tipo de relación, el objeto relacionado puede ser miembro de las clases “Part” (si es “sus_mpp_Porcess_part”), “sus_mpp_Process” (si es “sus_mpp_Process_aggregate” o “sus_mpp_Process_precedence”) o “sus_mpp_Resource” (si es “sus_mpp_Process_resource”), su rango estará compuesto por estas tres clases. De la misma manera quedaría definida la otra propiedad.

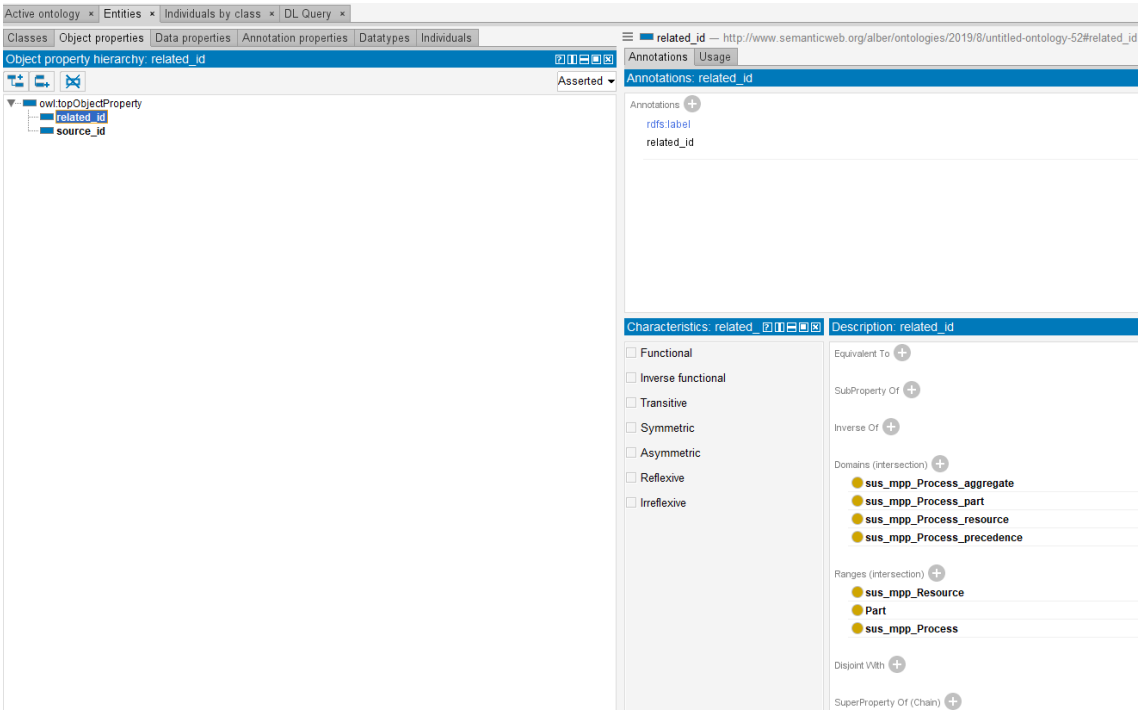

Figura 6.6 Muestra de la propiedad objeto "related_id" definida

6.1.3 Data properties

A continuación, se muestran los “Data properties”. Al igual que en el caso de los “Object properties”, hay algunos atributos que sirven para definir varias clases. Puede verse cómo se han creado un total de trece atributos siendo doce para clases de individuos y uno para clases que relacionan individuos.

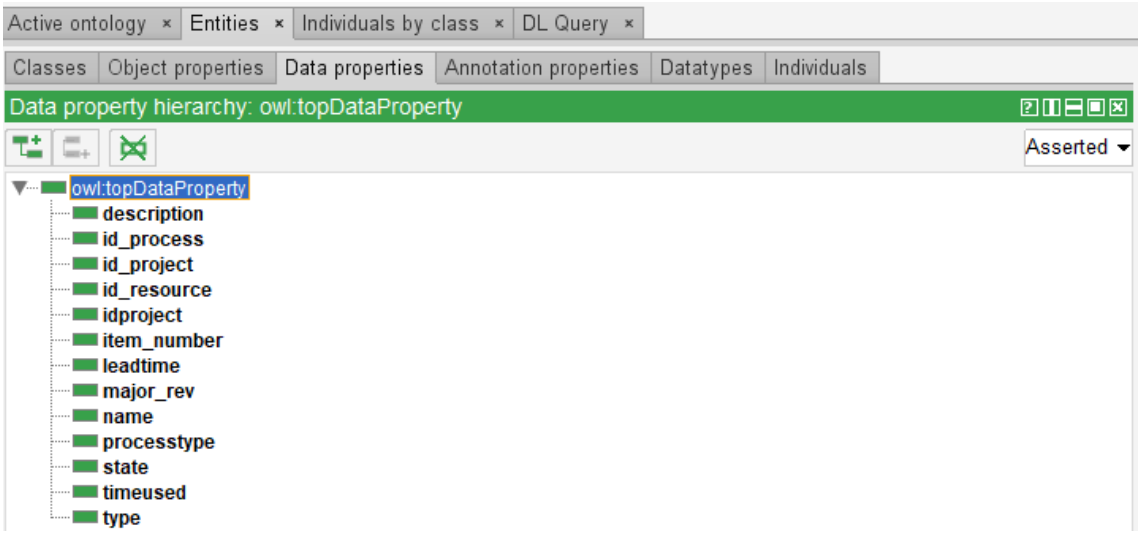

Figura 6.7 Definición de las propiedades dato para el caso práctico

La primera sería "description", que como a las otras propiedades, se le ha puesto una etiqueta con su nombre y se ha definido su dominio y rango.

Puesto que esta propiedad puede aparecer en las instanciaciones de miembros de cualquiera de las cuatro clases de individuos creadas, su dominio estará compuesto por esas cuatro clases y como independientemente del tipo de clase, el valor de este atributo siempre va a ser una cadena de caracteres, su rango será la clase de tipo "string".

Nótese que se ha marcado como característica de este atributo la propiedad "Functional". Esto lo único que viene a decir es que un individuo no puede tener más de un valor para el mismo atributo. Como es lógico, todos los atributos clave deben ser funcionales, pero, además, puede carecer de sentido tener atributos repetidos en individuos únicos que están perfectamente identificados, por lo que en este caso se van a marcar todos los atributos como funcionales.

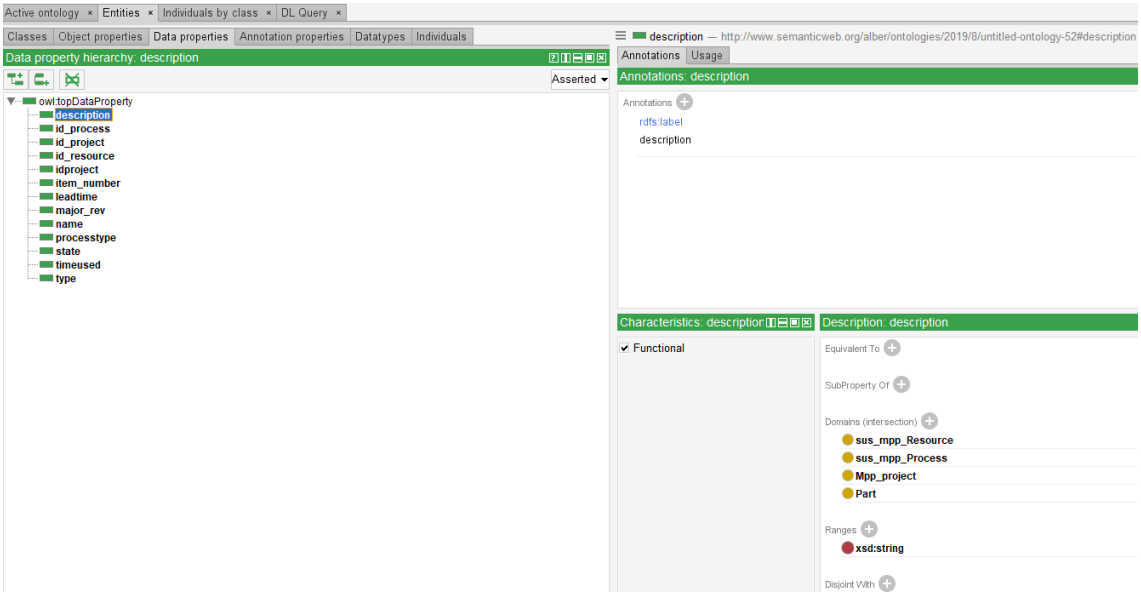

Figura 6.8 Muestra de la propiedad dato "description" definida

6.1.4 Individuals

Puesto que de momento sólo se pretendía definir la ontología, no se van a instanciar individuos, de manera que ese bloque quedará vacío y se generará un XML, únicamente, con la información que se ha introducido hasta aquí.

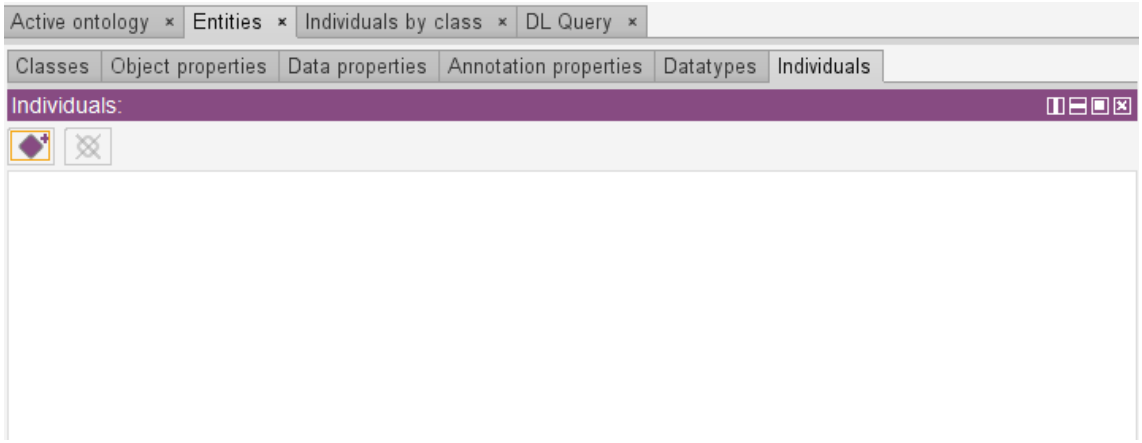

Figura 6.9 Individuos para el caso práctico

6.2 Generación de XML

Como se ha recalcado, de momento no se ha instanciado nada por lo que en el XML aparecerán tres bloques claramente diferenciados en vez de cuatro, a falta del de los individuos.

A continuación, se muestra una captura de cada bloque mostrando únicamente los dos primeros nodos para no extender este punto en exceso.

Como puede verse, el nodo de cada propiedad está compuesto por un elemento cuyo primer atributo tiene como valor el nombre de la propiedad con la dirección en la que está definida y una serie de nodos hijos que definen el dominio y el rango de la misma manera, es decir, con el nombre de la clase y el espacio de nombres en el que está definida dicha clase en el valor del primer atributo de cada elemento. Además, en el último nodo aparece una etiqueta con el nombre de la propiedad.

Esto vale tanto para las "Object properties" como para las "Data properties", pero en estas últimas puesto que, en este caso, se han definido todas como funcionales aparecerá también un nodo hijo que define el tipo indicando en el valor del primer atributo que se trata de una propiedad funcional y el espacio de nombres en el que está definida esta característica.

```
<!--
///////////////////////////////////////////////////////////////////////////////////
//
// Object Properties
//
///////////////////////////////////////////////////////////////////////////////////
 -->




<!-- http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#related_id -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#related_id">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_aggregate"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_part"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_precedence"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_resource"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Part"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Resource"/>
    <rdfs:label>related_id</rdfs:label>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#source_id -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#source_id">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_aggregate"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_part"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_precedence"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_resource"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process"/>
    <rdfs:label>source_id</rdfs:label>
</owl:ObjectProperty>
```

Figura 6.10 Muestra de las propiedades objeto de la ontología

```
<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
///////////////////////////////////////////////////////////////////////////////////////
 -->



<!-- http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#description -->

<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#description">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Mpp_project"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Part"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Resource"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>description</rdfs:label>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#id_process -->

<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#id_process">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>id_process</rdfs:label>
</owl:DatatypeProperty>
```

Figura 6.11 Muestra de las propiedades dato de la ontología

Por último, quedarían las clases, que van a quedar declaradas como las propiedades con la salvedad de que éstas van a tener como nodos hijos tantos como atributos clave se les asignen además de una etiqueta con el nombre de la clase.

```
<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////////
 -->



<!-- http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Mpp_project -->

<owl:Class rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Mpp_project">
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#idproject"/>
    </owl:hasKey>
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#major_rev"/>
    </owl:hasKey>
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#state"/>
    </owl:hasKey>
    <rdfs:label>Mpp_project</rdfs:label>
</owl:Class>



<!-- http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Part -->

<owl:Class rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Part">
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#item_number"/>
    </owl:hasKey>
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#major_rev"/>
    </owl:hasKey>
    <owl:hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#state"/>
    </owl:hasKey>
    <rdfs:label>Part</rdfs:label>
</owl:Class>
```

Figura 6.12 Muestra de las clases de la ontología

6.3 Sistema de instanciación

Como se ha explicado anteriormente, el sistema que se ha desarrollado se compone de una parte de definición y otra de instanciación, pero la función que tiene es, en realidad, instanciar en un sentido u otro y dependiendo de dicho sentido se estará imporatando/exportando información a/desde una aplicación.

6.3.1 Introducción a Aras Innovator

ARAS INNOVATOR[xxxviii] es un software Open Source de la compañía ARAS CORPORATION. Una empresa de origen estadounidense que ofrece servicios de integración de soluciones PLM y PDM en una única plataforma.

ARAS INNOVATOR permite al usuario poder gestionar todo el ciclo de vida de un producto. Desde su creación, pasando por su diseño, fabricación, montaje y servicio hasta llegar a su fin de vida.

Las soluciones ARAS están divididas en tres grandes bloques:

- PLM: Para gestionar de manera colaborativa los procesos a través del ciclo de vida.
- PDM: Para gestionar los datos de CAD en referencia a: Producto, parte y documentación.
- PLATAFORMA: Para adaptar, extender y construir aplicaciones a las necesidades de la organización.

Al ser una aplicación Open Source, el código de programación es abierto, por lo que el software puede ser distribuido, desarrollado y modificado libremente sin restricciones de licencias. Esto va a permitir tener una gran flexibilidad a la hora de poder adaptar el software a las necesidades de las empresas con independencia del tamaño que estas posean.

No obstante, ARAS ofrece la posibilidad de poder comprar paquetes de licencias con módulos desarrollados por la propia compañía con independencia de lo ya implementado o desarrollado por terceros.

Para lograr esa flexibilidad, ARAS se ha centrado en la creación de una aplicación web run-time que ejecuta un conjunto de servicios acoplados de manera escalada, es decir, formando una arquitectura orientada a servicios (SOA). Un modelo de arquitectura que se caracteriza por un procedimiento para crear y usar los diversos procesos, reunidos en forma de servicios, que configuran un determinado Proceso de Negocio (Un proceso de negocio se puede ver como un conjunto estructurado de tareas, que contribuyen colectivamente a lograr los objetivos de una organización). Además, ofrece la posibilidad de conectar abiertamente con otras soluciones de negocio, ya sean sistemas ERP (SAP, Oracle) u otros sistemas PLM (Dassault Systemes, SIEMENS).

El sistema está compuesto por los clientes web, un servidor de aplicación, una base de datos SQL server y un servidor de archivos, Figura 6.13. Todo basado en protocolos estándar de Internet, incluyendo HTTP / HTTPS, XML y SOAP (Simple Object Access Protocol).

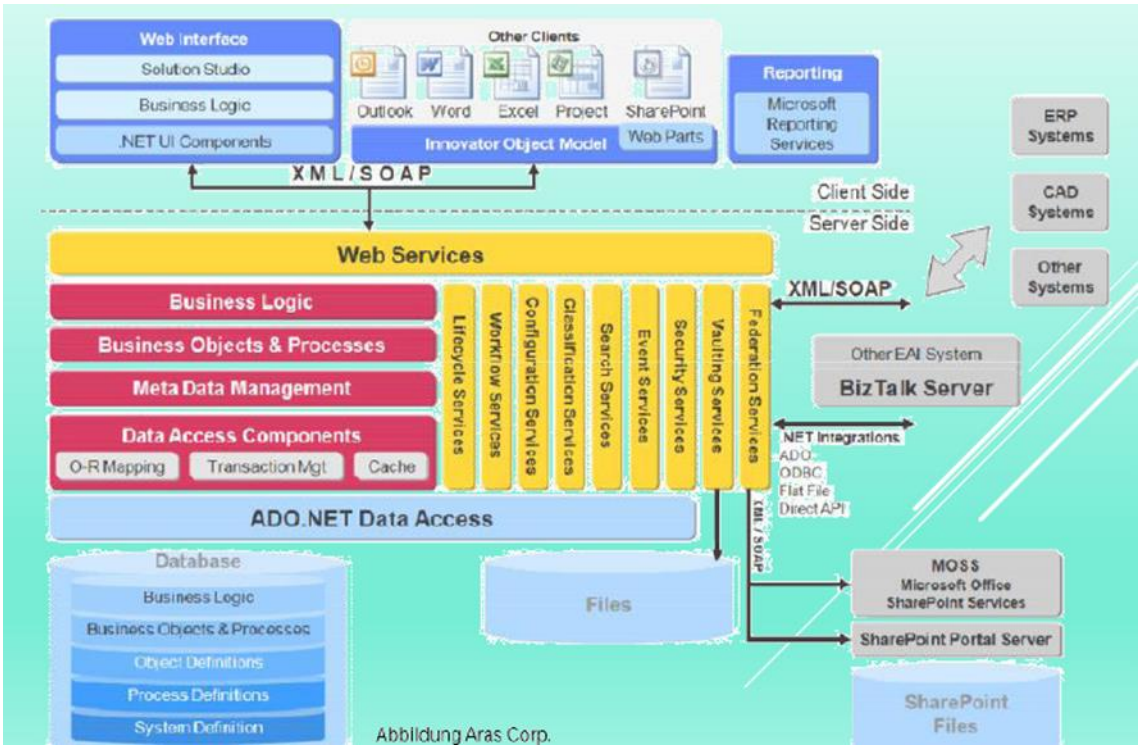

Figura 6.13 Arquitectura Software Aras Innovator

6.3.2 La customización del entorno PLM Aras Innovator

La gran ventaja competitiva que ofrece el sistema ARAS Innovator frente a otros competidores más poderosos como las herramientas de Dassault Systemes dentro del mercado de los sistemas PLM es la capacidad que ofrece para la customizacion y personalización de sus funcionalidades, adaptándose fácilmente a los requerimientos del cliente y, consecuentemente, haciéndolo idóneo para pequeñas y medianas empresas.

La customizacion o personalización del sistema ARAS Innovator está basada en el concepto de Ítem. Un Ítem es un objeto gestionado desde ARAS y que el sistema permite crear, modificar y establecer relaciones entre ellos. El concepto de ItemType de ARAS equivale a la definición de una clase vista en el apartado 3 de este TFM.

Los Ítems son el centro del sistema ARAS, en ellos se definen sus propiedades, formularios o vistas, ciclos de vida del objeto, flujo de trabajo, permisos y relaciones.

Los Ítems están relacionados entre sí mediante el concepto de relaciones, las cuales habilitan la conexión entre los diferentes tipos de Ítems. Las relaciones se componen del Ítem origen y del Ítem destino, además ARAS permite expresar la cardinalidad existente entre elementos.

Los Ítems, por tanto, permiten la configuración del entorno de ARAS para incorporar nuevos conceptos como en el caso de este TFM añadiendo las clases del diagrama además de definir las relaciones descritas previamente entre los diferentes objetos.

A continuación, se describe brevemente y a modo de ejemplo como se crean nuevos Ítems, equivalentes a las clases en el modelo de datos propuesto. La implementación paso a paso del modelo de datos se describirá en el siguiente apartado.

El primer paso para la configuración de la herramienta, previo a la implenetación de nuevos objetos, es la creación del nuevo tipo de dato en forma de ItemType.

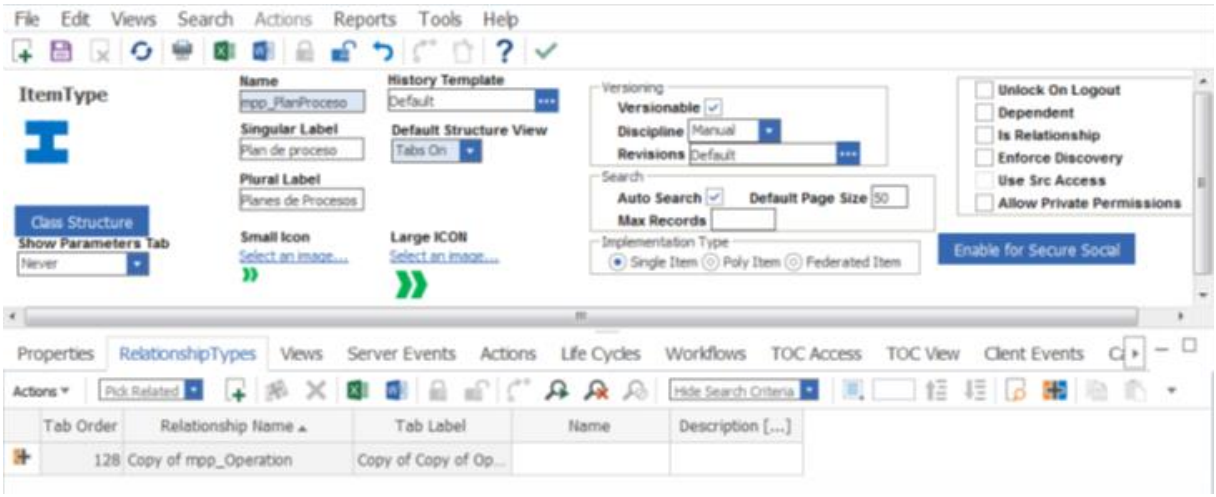

Figura 6.14 Creación de un nuevo ÍtemType

Ahora, se incluyen los atributos definidos previamente por el usuario en la pestaña propiedades del ÍtemType. Aquí se define la etiqueta con la que aparecerá en el objeto, qué tipo de dato es el atributo, etc.

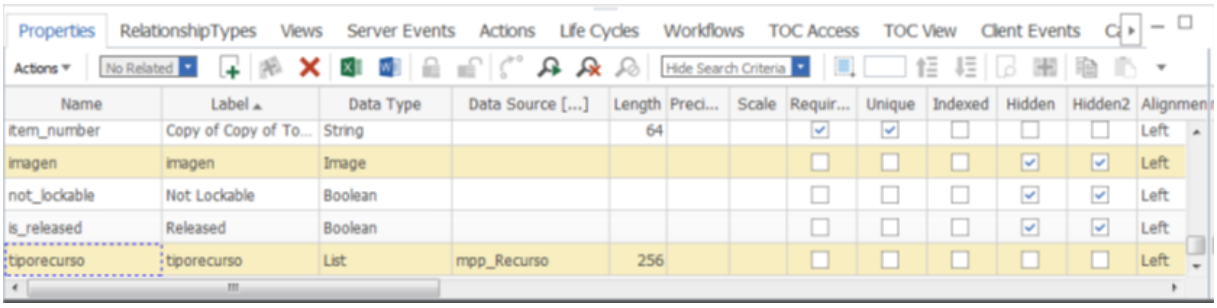

Figura 6.15 Adición de nuevos atributos al ItemType

Se rediseña el formulario del ítem, siendo el formulario, la ventana que vera el usuario cuando seleccione el objeto. Aquí se puede definir el nombre de los campos definidos en la pestaña de propiedades, qué propiedades desea el usuario que sean visibles, su colocación en el espacio, etc.

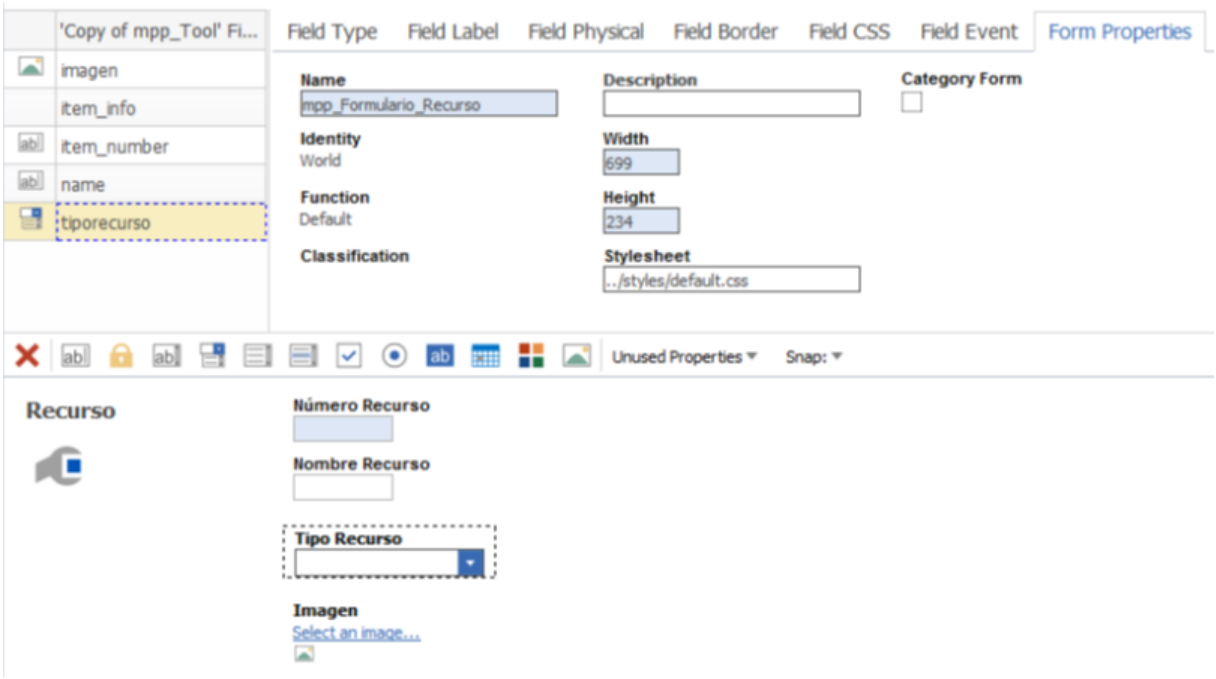

Figura 6.16 Rediseño del formulario del ItemType y vista que presentará

La herramienta ARAS permite también la personalización añadiendo los permisos de acceso y de visibilidad de los nuevos objetos para aquellos usuarios que se definan.

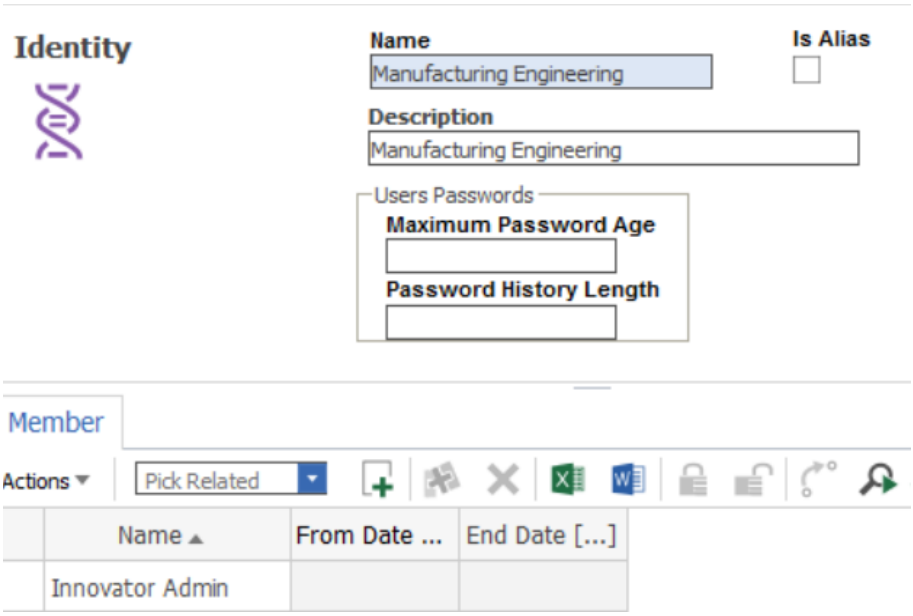

Figura 6.17 Asignación de permisos de acceso al ItemType

Si una clase se puede clasificar en distintos tipos se necesita crear una lista que contenga las distintas opciones que pueden existir del tipo de dato. Por ejemplo, para el modelo propuesto en la clase del tipo “Operación”, estas operaciones pueden puede ser del tipo Provide, Ensamblado, etc.

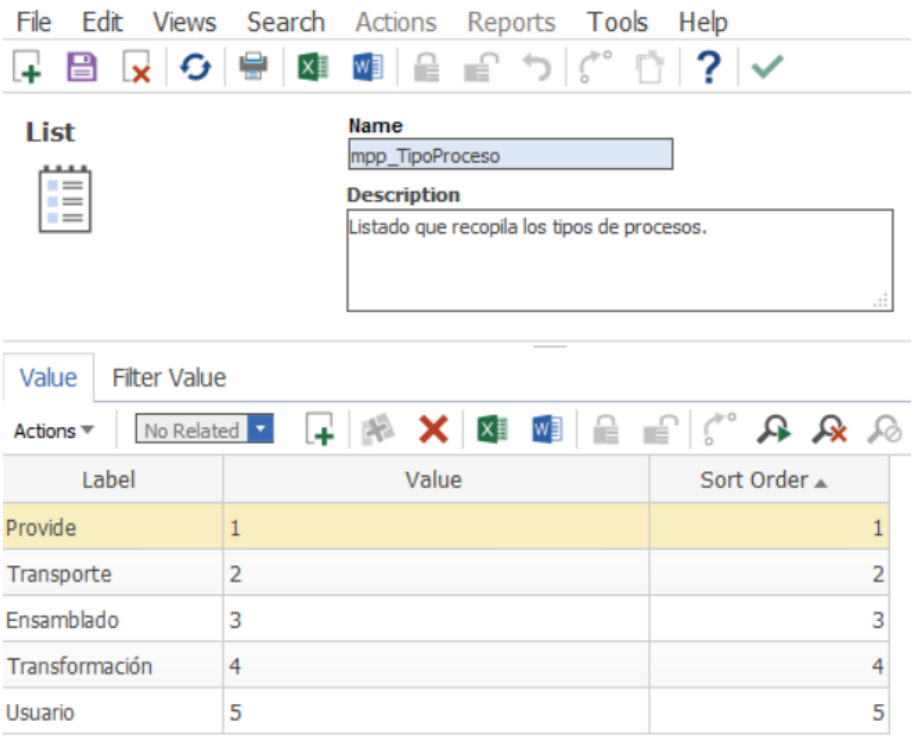

Figura 6.18 Creación de listas de tipos de procesos

En la pestaña relaciones del ItemType, se especifican las relaciones entre objetos, indicando el objeto origen y el objeto relacionado, así como su cardinalidad.

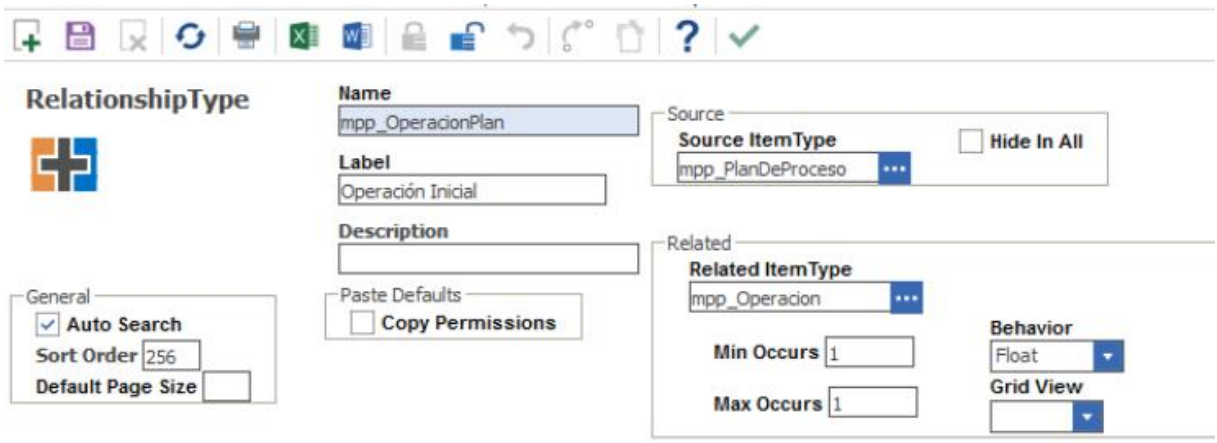

Figura 6.19 Incorporación de relaciones al ItemType

6.3.3 Diseño de la interface de gestión de la ontología con el sistema PLM

La interface para la gestión de instanciación con el sistema PLM permite la creación e incorporación de datos al sistema PLM. Las funciones que permiten interactuar con Aras podrían dividirse en cuatro módulos, de modo que se tendrían las de la conexión a la aplicación que va a permitir introducir y extraer información, las de los ItemTypes y las de los atributos y relaciones asociados a dichos ItemTypes.

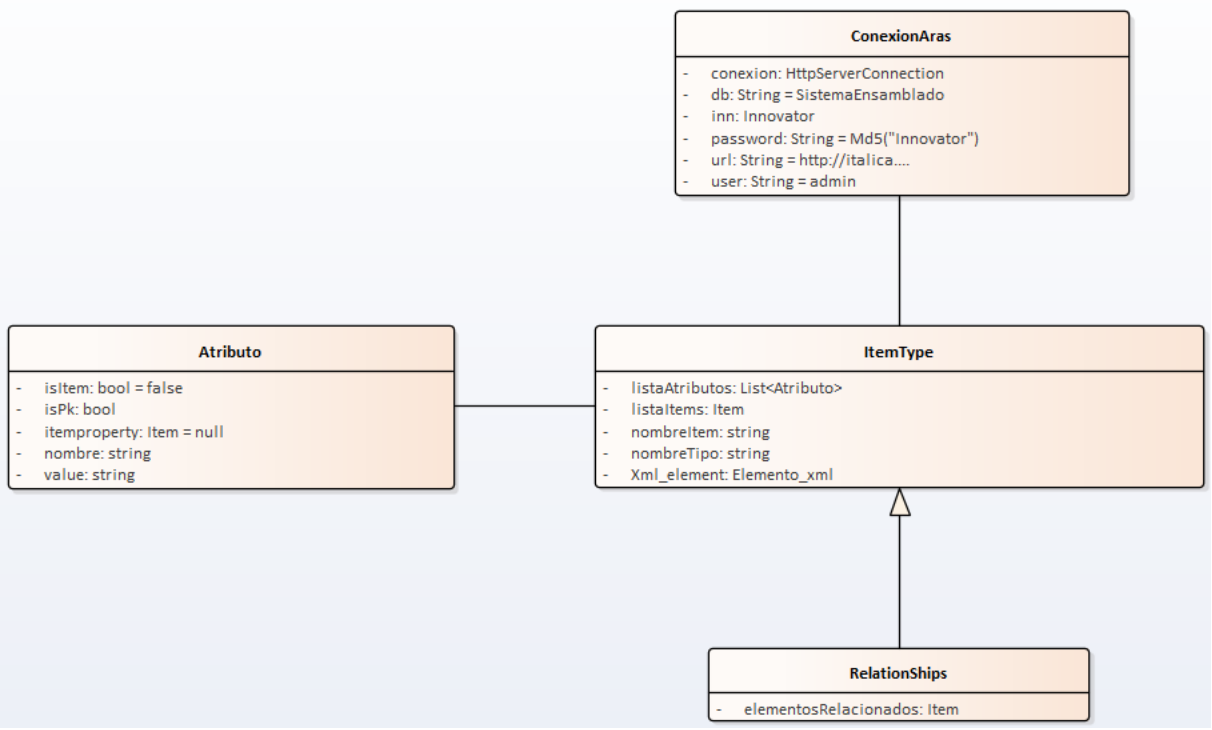

Figura 6.20 Interface de gestión de la ontología con el sistema PLM

La clase ItemType permite accede a cualquier elemento definido en el sistema PLM, los itemtypes se identifican unívocamente por nombre (nombreItem o nombreTipo). Los itemtypes tiene un conjunto de propiedades (atributos) y se relacionan entre si (RelationShips). La interface de conexión con aras se realiza mediante mensajería soap, donde se especifica un usuario, clave, base de dato y la url donde está instalado el sistema. La clase Itemtype dispone de métodos para escribir y leer el nombre y el tipo de un individuo, para obtener una lista de sus atributos, para obtener una lista de los miembros de una clase…

Entre las de los atributos se tienen funciones o métodos para escribir y leer el nombre y el valor del atributo, para establecer que sea atributo clave, para obtener si es atributo clave…

Entre las de las relaciones se tienen funciones o métodos para escribir y leer relaciones, para obtener los individuos relacionados con un individuo concreto…

6.3.4 Diseño de subsistema de análisis de ontologías

Volviendo a las dos partes que componen el sistema desarrollado, se tiene una parte de definición y otra de instanciación.

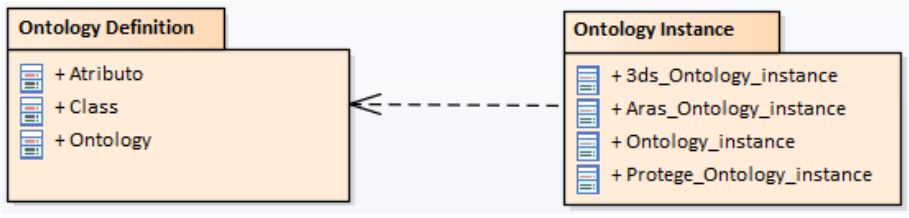

Figura 6.21 Núcleo del sistema de instanciación

La definición de ontología permite la carga y el procesado de la información que contiene la ontología, principalmente clases, “Data properties” y “Object properties” con sus respectivos axiomas. El procesado de la ontología se realiza usando las utilidades definidas en el sistema, en concreto las listas parametrizadas. En la parte de definición lo único que se hace es una lectura de la ontología de manera que se crean listas de clases, aprovechando los recursos del lenguaje de programación.

La instanciación permite ampliar la ontología con datos, por lo que es el propio modelo de interoperabilidad entre sistemas.

A continuación, se muestran algunas de las funciones que tendrían las listas.

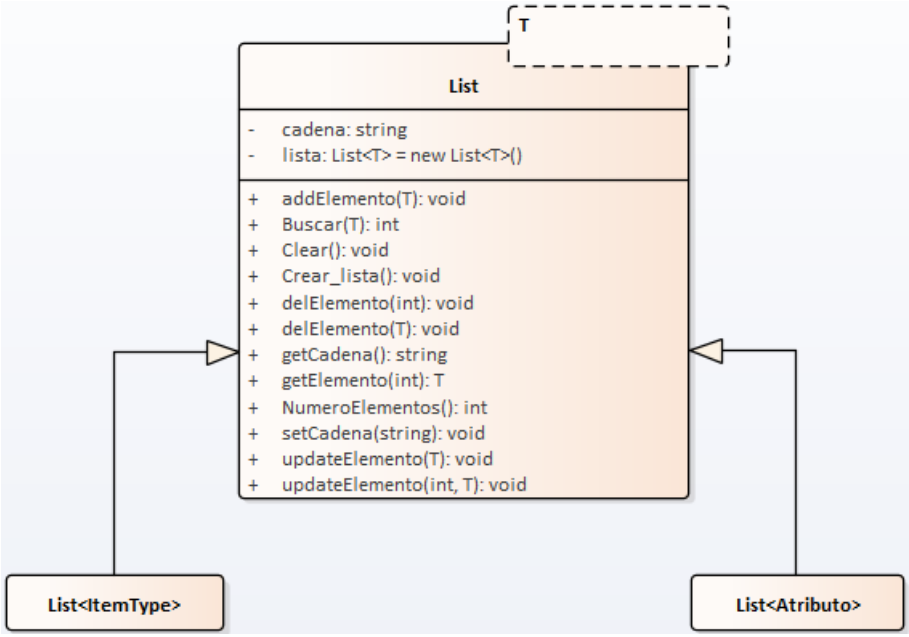

Figura 6.22 Funciones de las listas de una clase

La idea para definir el dominio es recorrer el documento nodo a nodo buscando clases y cada vez que se encuentre una, buscar los atributos asociados a dicha clase recorriendo las propiedades y localizando cuáles son las que tienen como dominio la clase en cuestión. Por ello, cada vez que se encuentra una clase, se crea una lista de atributos con todos los asciados a dicha clase incluyendo características como si es clave o no y, posteriormente, se añade a la lista de ItemTypes dicha clase con sus respectivos atributos.

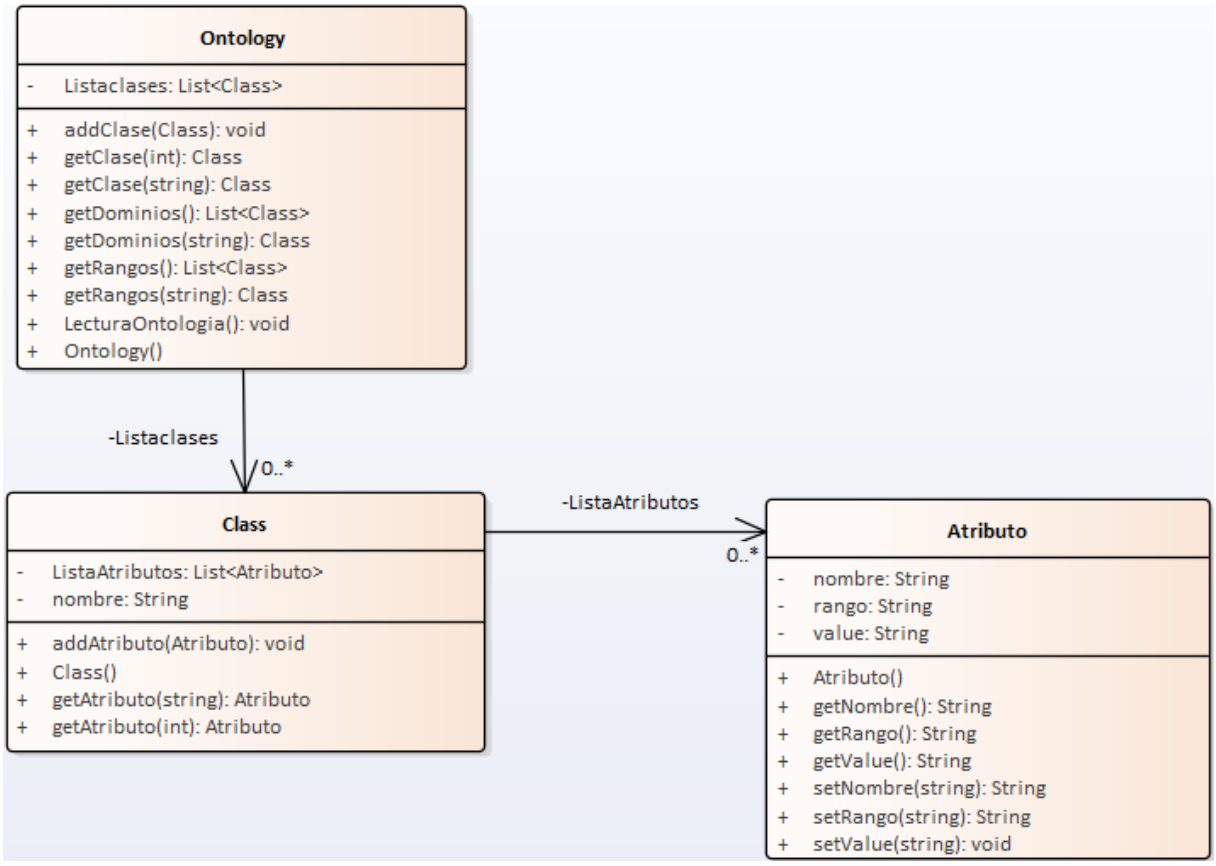

Figura 6.23 Subsistema de definición de ontologías

En la parte de instanciación, se importa o exporta información en base a la definición de la ontología que ha sido

hecha previamente.

Estas funciones que, en este proyecto, han sido desarrolladas para Protege y Aras lo que va a hacer es coger el XML, recorrer las instanciaciones, generar esos individuos a partir de las clases definidas con los valores de los atributos del XML y almacenarlos en la aplicación correspondiente.

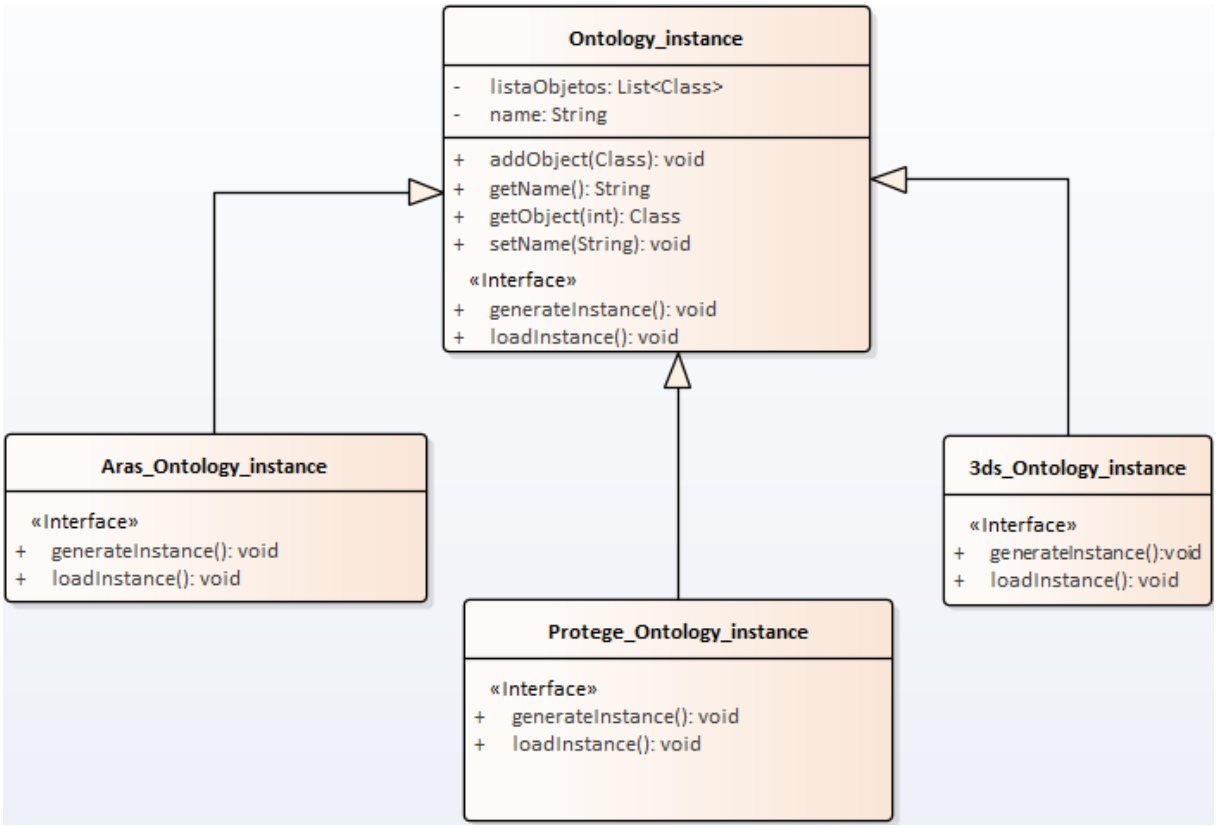

Figura 6.24 Subsistema de instanciación de ontologías

Por último, quedarían las funciones necesarias para comprender una ontología. Como se ha dicho en repetidas ocasiones las ontologías se representan en un formato XML que debe ser la sintaxis común a la hora de intercambiar información. Por ello, será necesario tener funciones que permitan leer y generar documentos XML.

Entre los tres bloques se pueden encontrar funciones para cargar y guardar documentos, para leer y escribir la raíz de dicho documento, para recorrer los nodos de la raíz, para añadir elementos, para recorrer los atributos de un elemento, para añadir atributos…

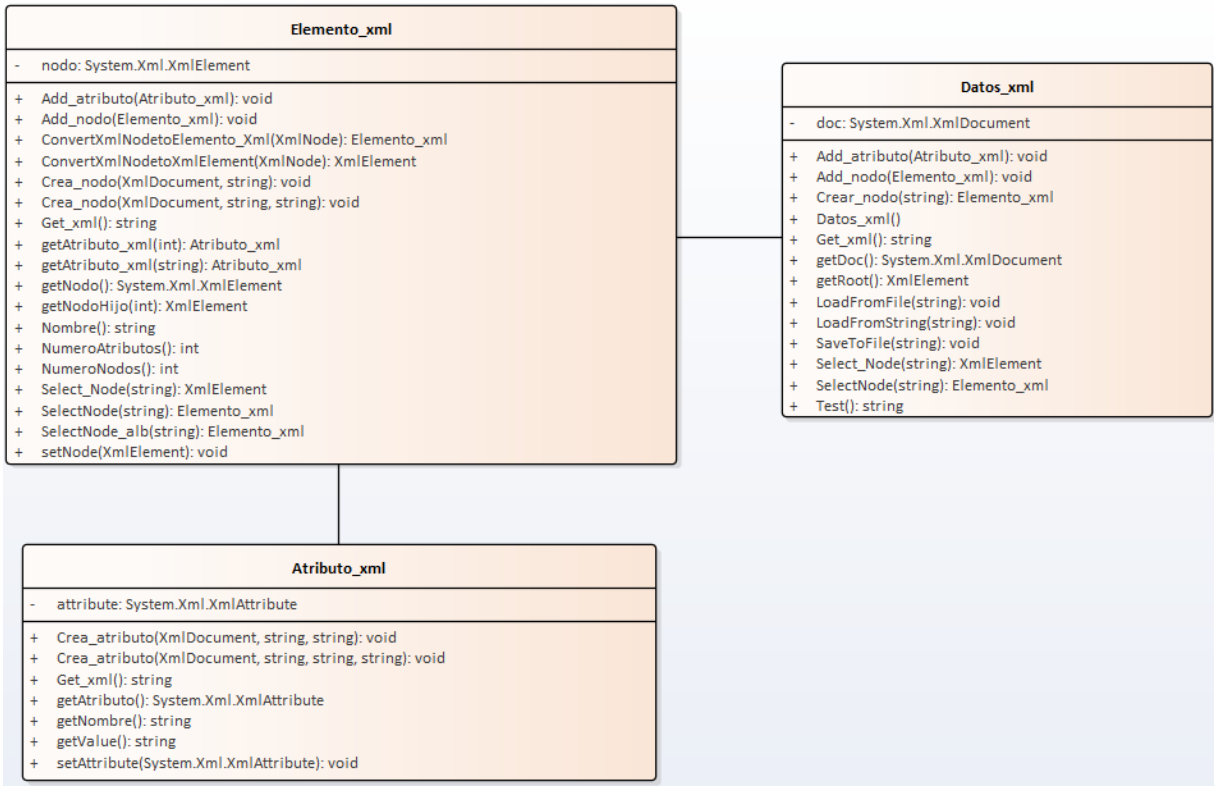

Figura 6.25 Librería desarrollada para poder procesar ficheros XML

7 DEMOSTRACIÓN

Una vez desarrollada la aplicación y definida la ontología, se pude proceder a la demostración.

7.1 Exportación desde Aras

Se han creado en Aras una serie de proyectos con datos, para que se pueda probar que definiendo una ontología con la información de interés se puede extraer esa información desde Aras haciendo uso de la aplicación.

Para hacer la prueba los únicos inputs que necesita el programa son la dirección en la que está la definición de la ontología y la dirección en la que se quiere guardar la instanciación de la ontología que se va a generar.

Una vez hecho esto, solo queda arrancar y clicar en "ArasToXml".

Automáticamente, la aplicación devuelve un fichero XML que se podrá encontrar en la ubicación que se ha introducido como input.

Ahora la idea es leer ese XML con Protege y ver si se ha extraído la información correctamente.

Para abrir una ontología con Protege hay que desplegar el menú "File" y, posteriormente, clicar en "Open". A continuación, aparecerá una ventana para buscar el archivo en la ubicación en la que se encuentre, y una vez que se selecciona sólo queda pulsar "Abrir".

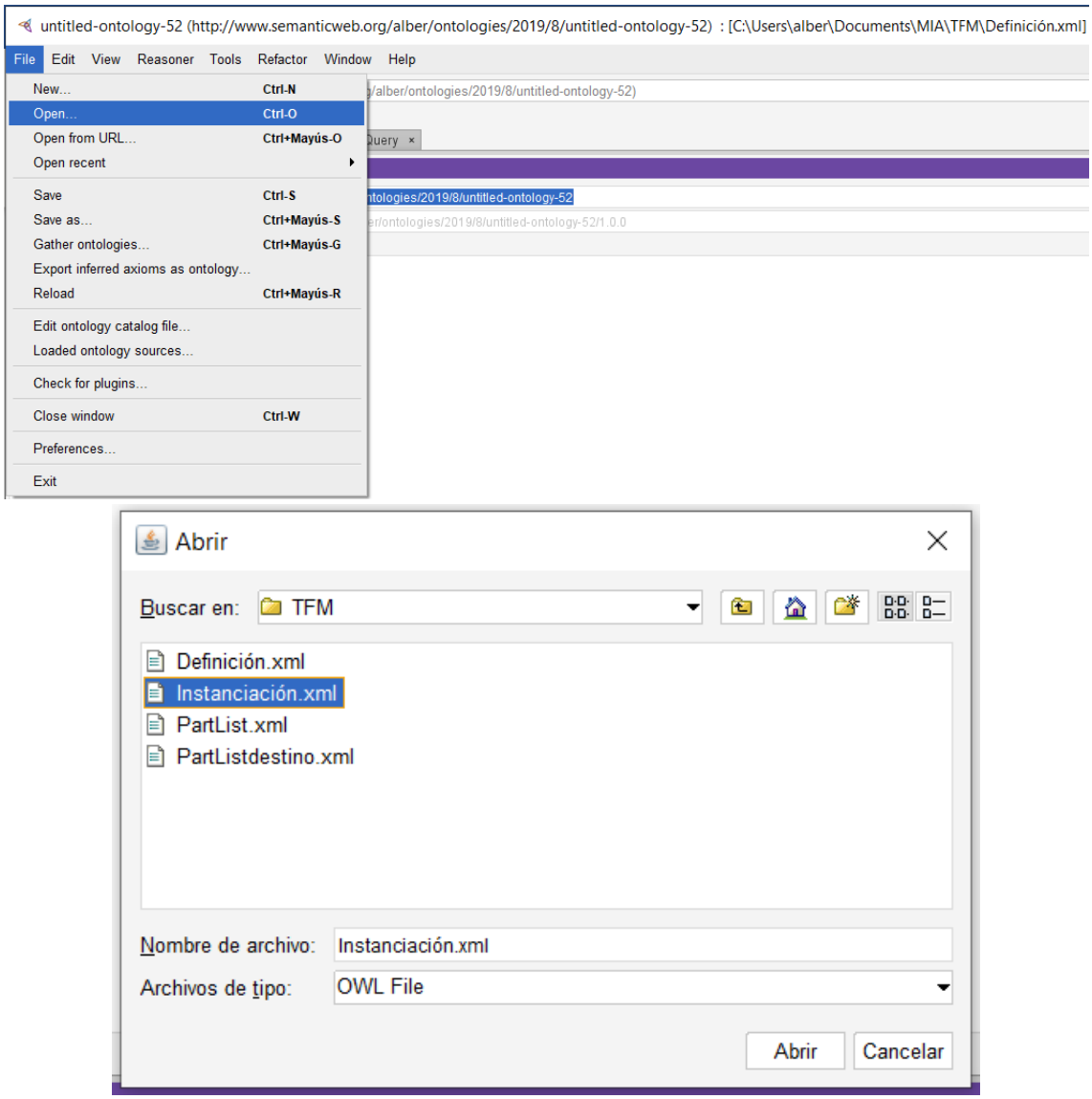

Figura 7.1 Pasos para abrir una ontología desde Protege

La definición de la ontología no cambia, la única diferencia entre el archivo de origen y de destino son los individuos que se han extraído de Aras y que aparecerán en la pestaña “Individuals”.

A continuación, se presentan muestras de los individuos instanciados.

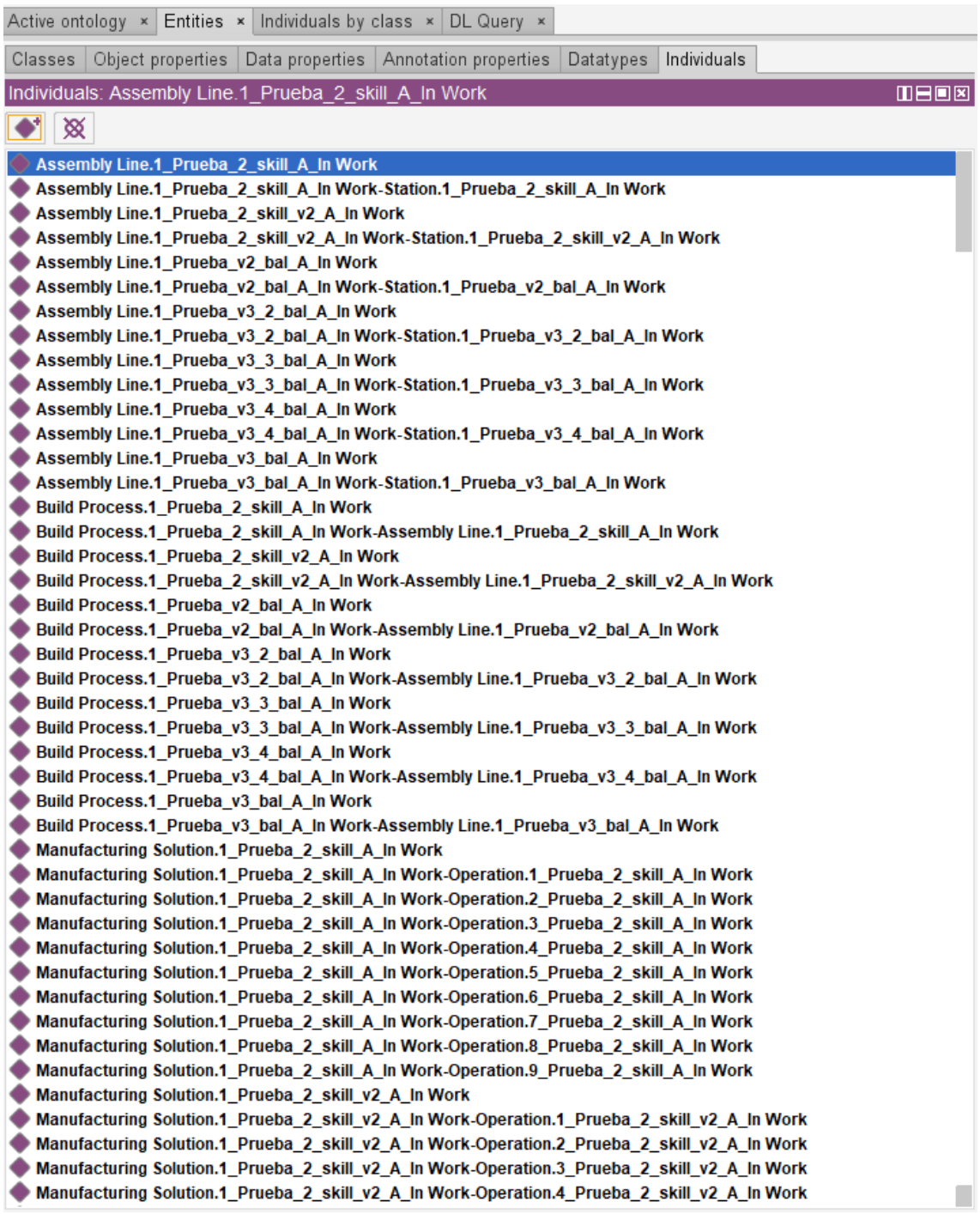

Figura 7.2 Muestra de individuos exportados desde Aras (1/4)

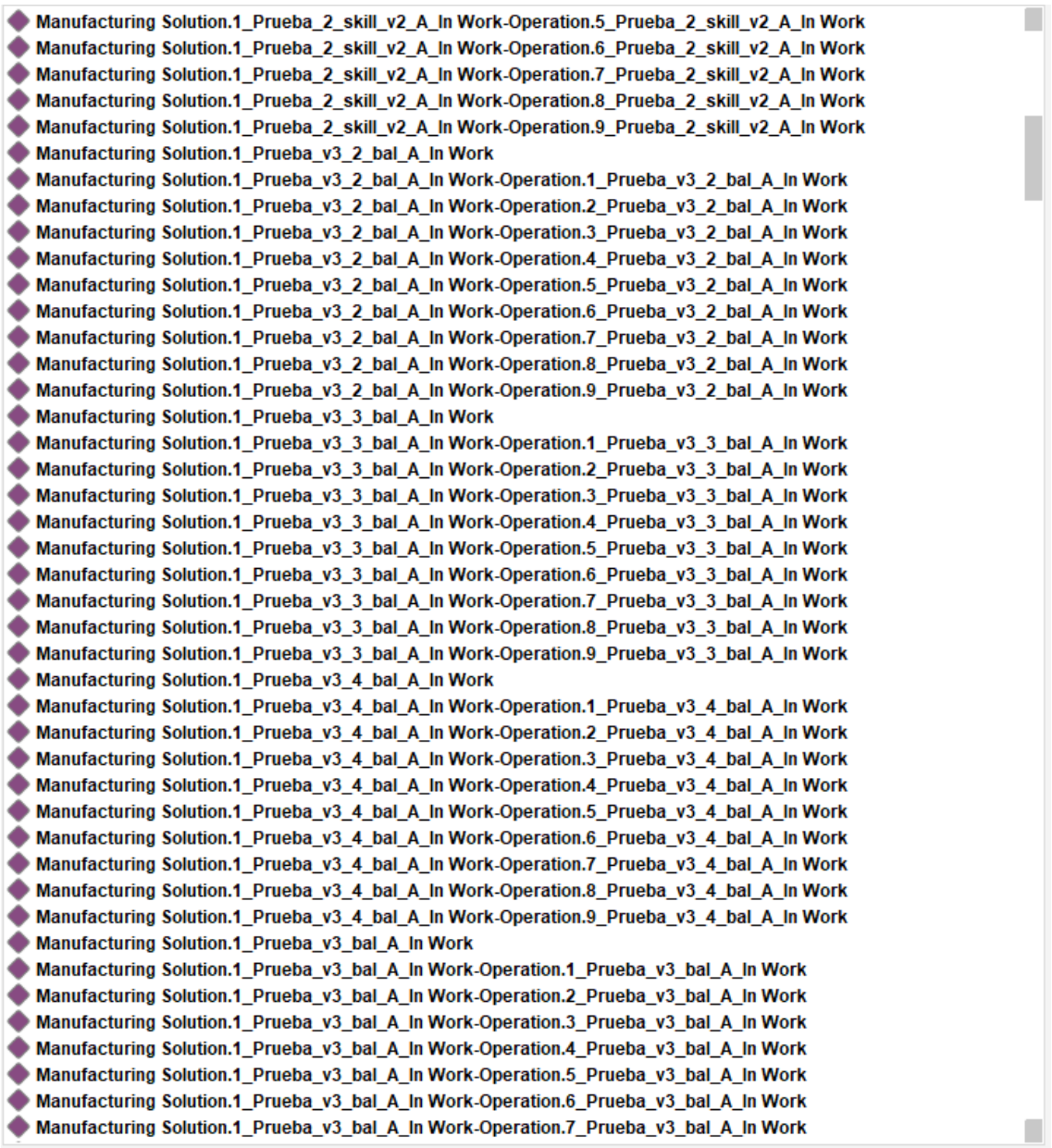

Figura 7.3 Muestra de individuos exportados desde Aras (2/4)

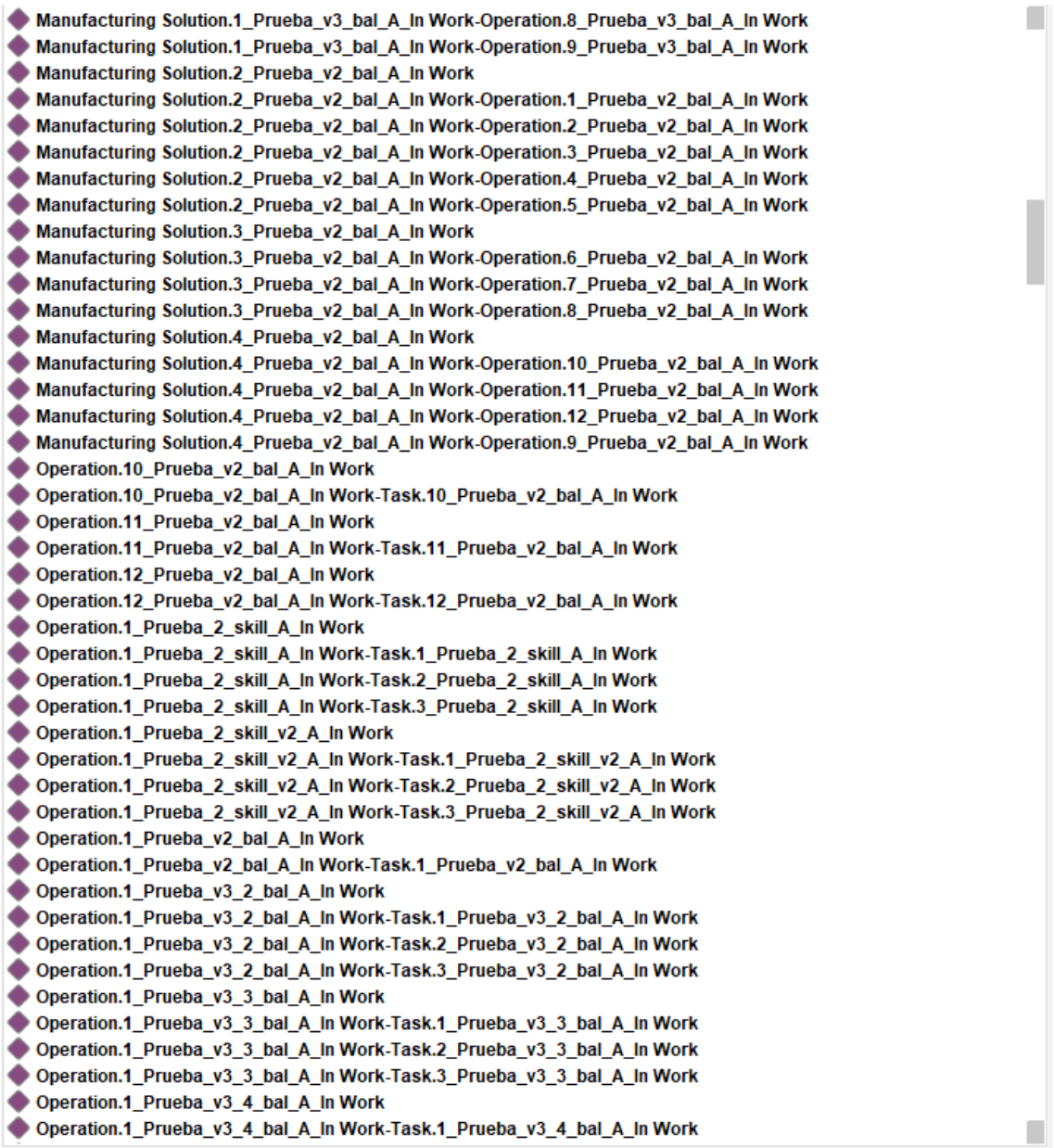

Figura 7.4 Muestra de individuos exportados desde Aras (3/4)

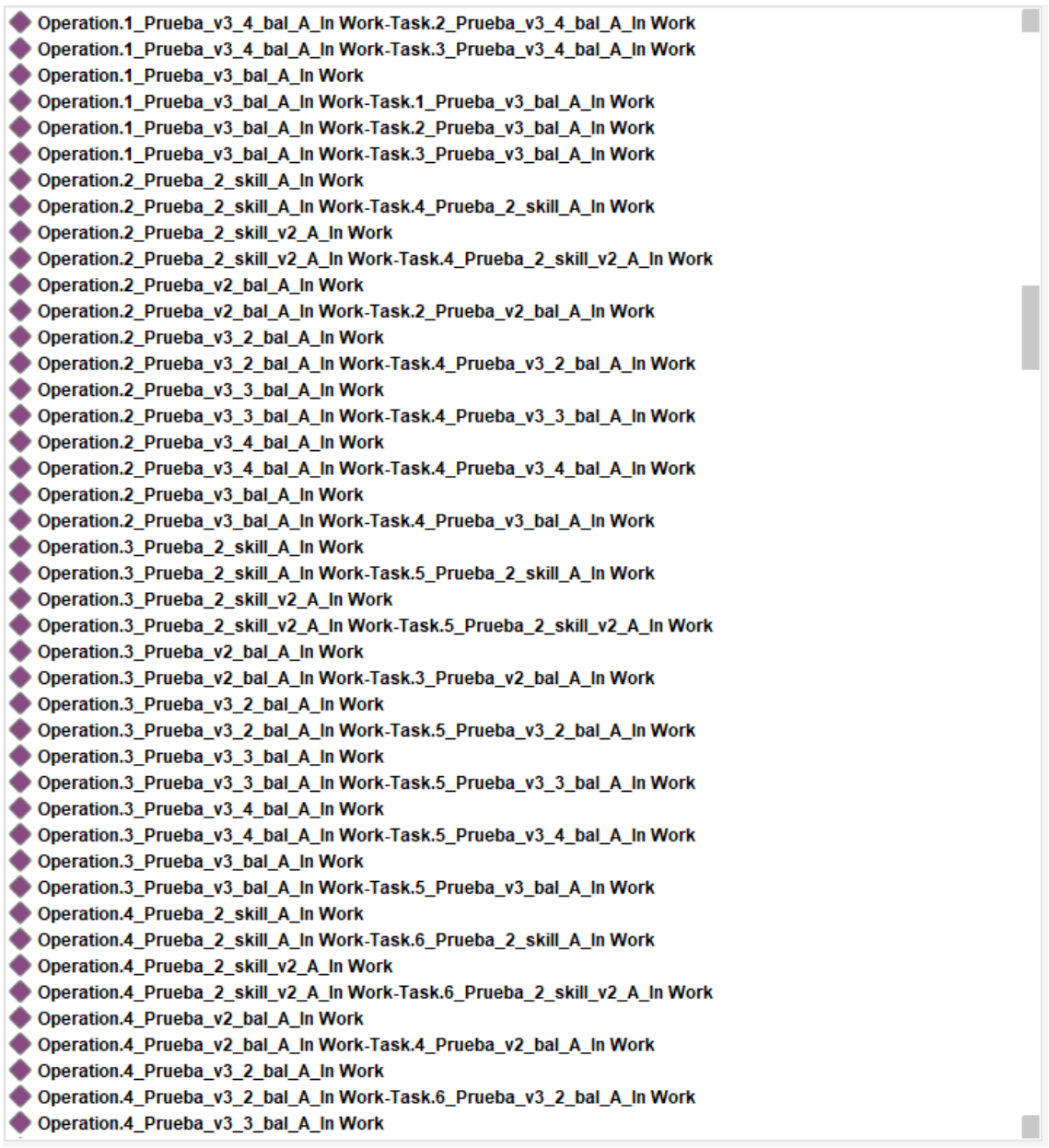

Figura 7.5 Muestra de individuos exportados desde Aras (4/4)

Y así hasta 545 individuos exportados. En el Anexo, se puede ver un extracto parcial del XML que incluye un par de miembros de cada clase.

Para ver cómo se han transferido los datos, se van a seleccionar un par de individuos. Por ejemplo, "Task.1_Prueba_2_skill_A_In Work" y "Task.2_Prueba_2_skill_A_In Work-Task.1_Prueba_2_skill_A_In Work".

Ejemplo 1:

- En la venta "Annotations" se puede apreciar que el individuo tiene una etiqueta con su nombre.
- En la ventana "Description" se puede ver que el individuo es miembro de la clase "sus_mpp_Process".
- En la ventana "Property assertions" se tienen las propiedades que se han definido para esa clase en la ontología con los valores que dichas propiedades tenían en Aras para este individuo.
  Nótese que se ha configurado la aplicación para que los atributos que no tienen un valor asignado y, por tanto, no aportan información, no sean extraídos. Esta es la razón por la que aquí falta el atributo "description", porque como se verá más adelante, este atributo para este individuo está vacío en Aras.

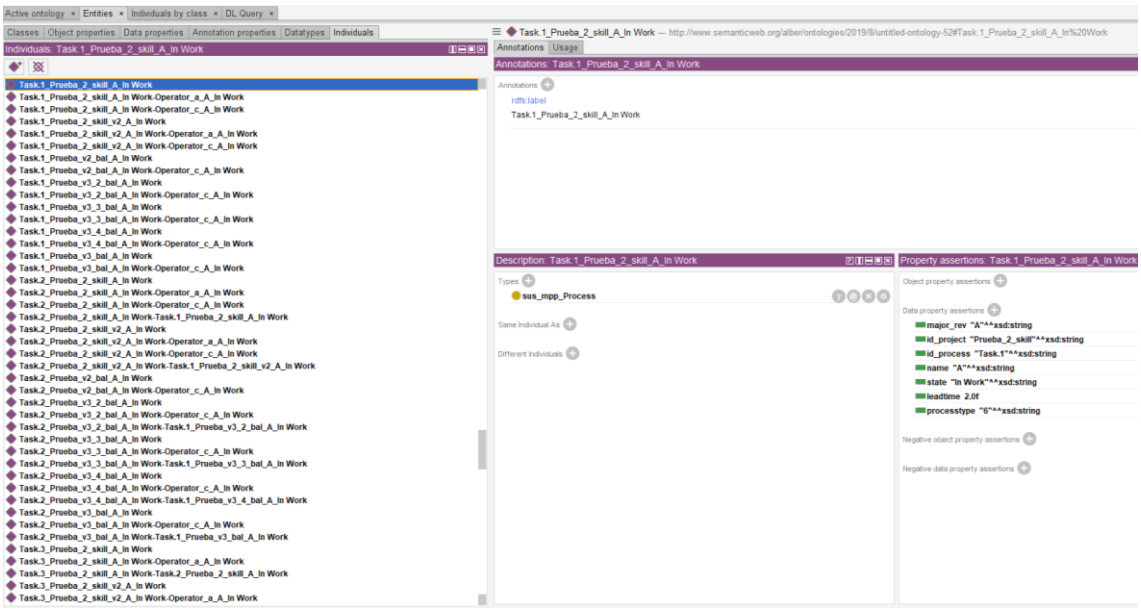

Figura 7.6 Ejemplo 1: "Task.1_Prueba_2_skill_A_In Work"

Ejemplo 2:

- En la venta “Annotations” se puede apreciar que el individuo tiene una etiqueta con su nombre.
- En la ventana “Description” se puede ver que el individuo es miembro de la clase “sus_mpp_Process_precedence”.
- En la ventana “Property assertions” se tienen las propiedades que se han definido para esa clase en la ontología. En este caso, al tratarse de una relación tiene que haber propiedades objeto, pero no tiene por qué haber atributos, ya que si la relación, únicamente, tiene la intención de asociar dos individuos, puede quedar totalmente definida sin tener que añadir más características.
  Esta relación vendría a decir que la “Task.1_Prueba_2_skill_A_In Work”, tiene que hacerse antes que la “Task.2_Prueba_2_skill_A_In Work”.

  Una cosa interesante de Protege es que como las relaciones entre individuos son de individuos que están instanciados en la misma ontología, se crean hipervínculos entre ellos y se puede navegar de manera que, si se clica en un elemento relacionado, automáticamente se abre el individuo en cuestión.

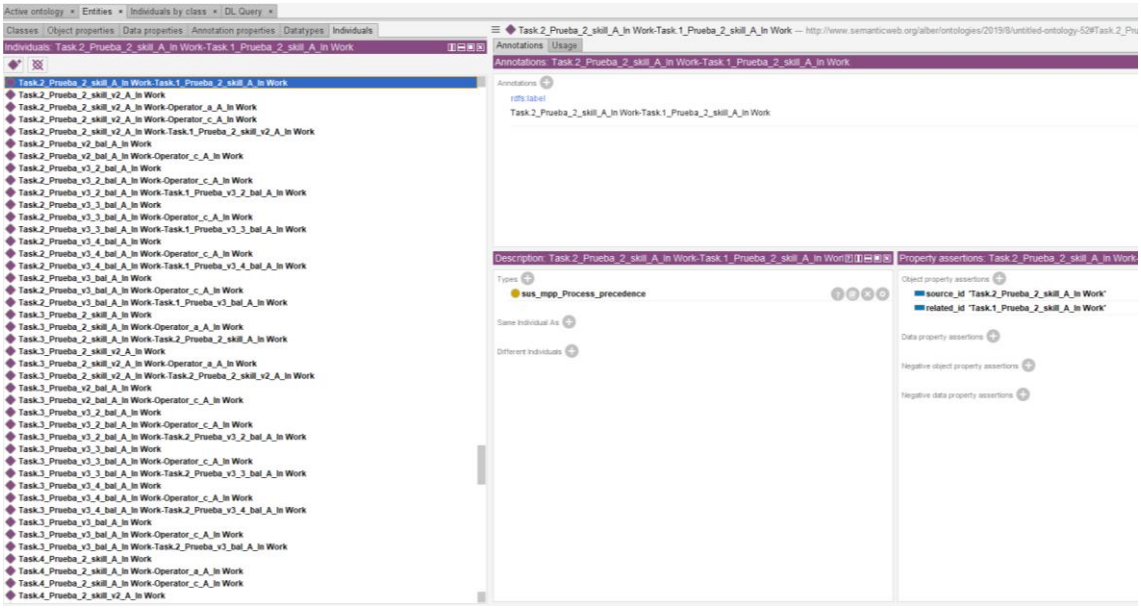

Figura 7.7 Ejemplo 2: “Task.2_Prueba_2_skill_A_In Work-Task.1_Prueba_2_skill_A_In Work”

Otra cosa interesante de Aras es que en el apartado de clases te indica si tiene instanciaciones, por lo que se

puede buscar un individuo directamente por el tipo.

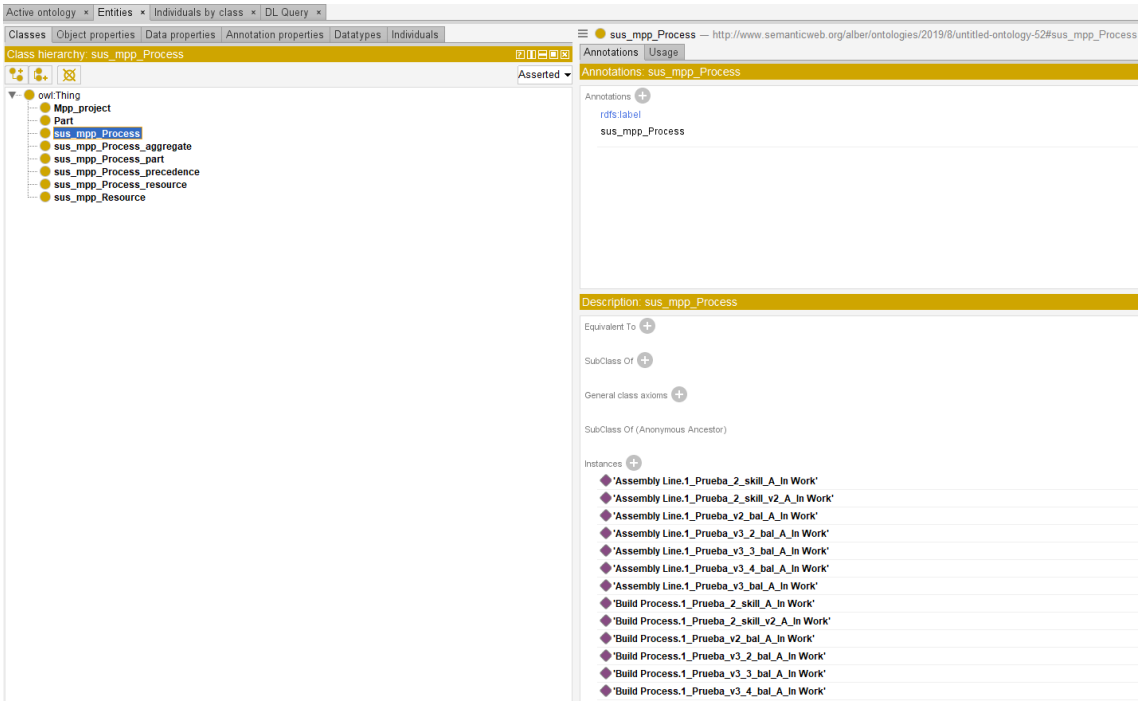

Figura 7.8 Muestra de miembros de la clase "sus_mpp_Process"

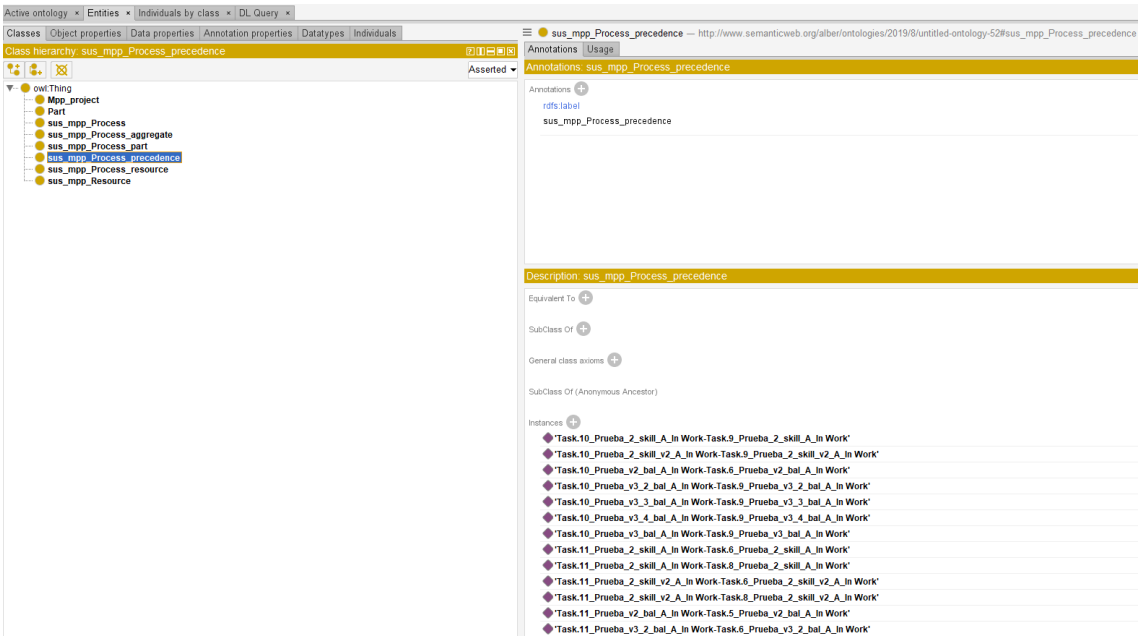

Figura 7.9 Muestra de miembros de la clase "sus_mpp_Process_precedence"

Para terminar esta parte, se van a buscar en Aras los ejemplos mostrados para comparar la información que se tiene con la que se ha extraído.

Una vez se entra en Aras, habría que irse a la parte de procesos y buscar el "Task.1_Prueba_2_skill_A_In Work". Si se filtra por "Id_Process" los que son "Task.1", aparecen siete procesos diferentes, uno por cada proyecto. Si además se filtrara por "IdProject" los que son "Prueba_2_skill", por "major_rev" los que son "A" y por "state" los que son "In Work", sólo podría aparecer uno porque se estaría filtrando por sus cuatro atributos clave.

Como puede verse en estas dos capturas, cada columna se corresponde con las características que se pueden almacenar en Aras de cada proceso. Así, es fácil darse cuenta de que el proceso buscado es el que aparece en la tercera línea sin necesidad de tener que filtrar por todos los atributos clave.

Nótese que al seleccionarlo se puede ver la fecha de creación y de la última modificación.

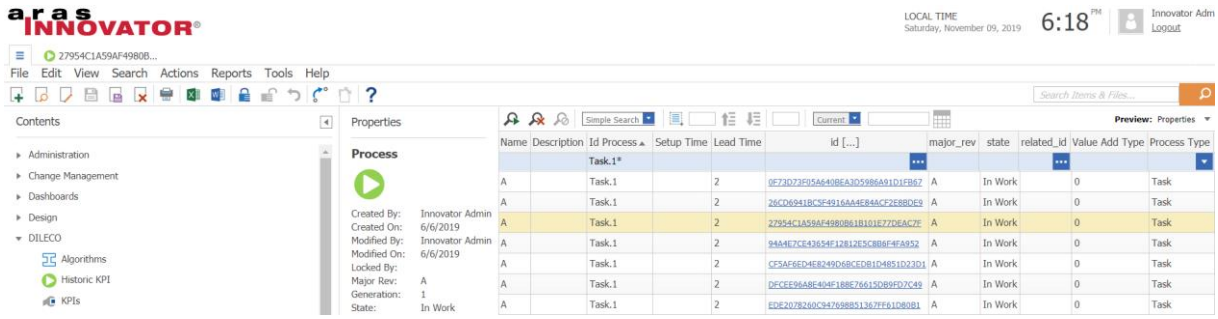

Figura 7.10 Búsqueda en Aras de "Task.1_Prueba_2_skill_A_In Work" (1/2)

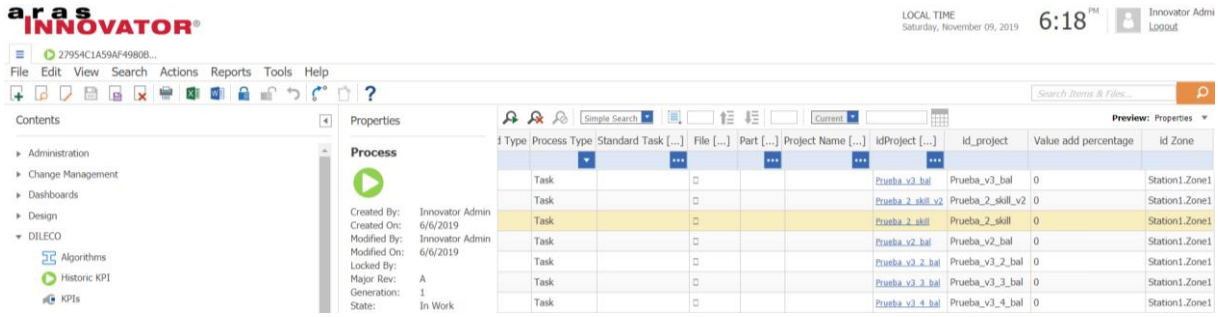

Figura 7.11 Búsqueda en Aras de "Task.1_Prueba_2_skill_A_In Work" (2/2)

Si se clica dos veces sobre el proceso en cuestión, se puede ver lo que se muestra a continuación:

En la parte superior, se muestran los atributos del individuo. Nótese que los valores que se habían extraído son correctos y que el campo de la descripción está vacío y por eso no se ha extraído nada.

En la parte inferior, se muestran las relaciones que tiene este individuo como origen. Puesto que este proceso sólo tiene relaciones con recursos, habría que irse a esa pestaña para encontrar los individuos con los que está relacionado.

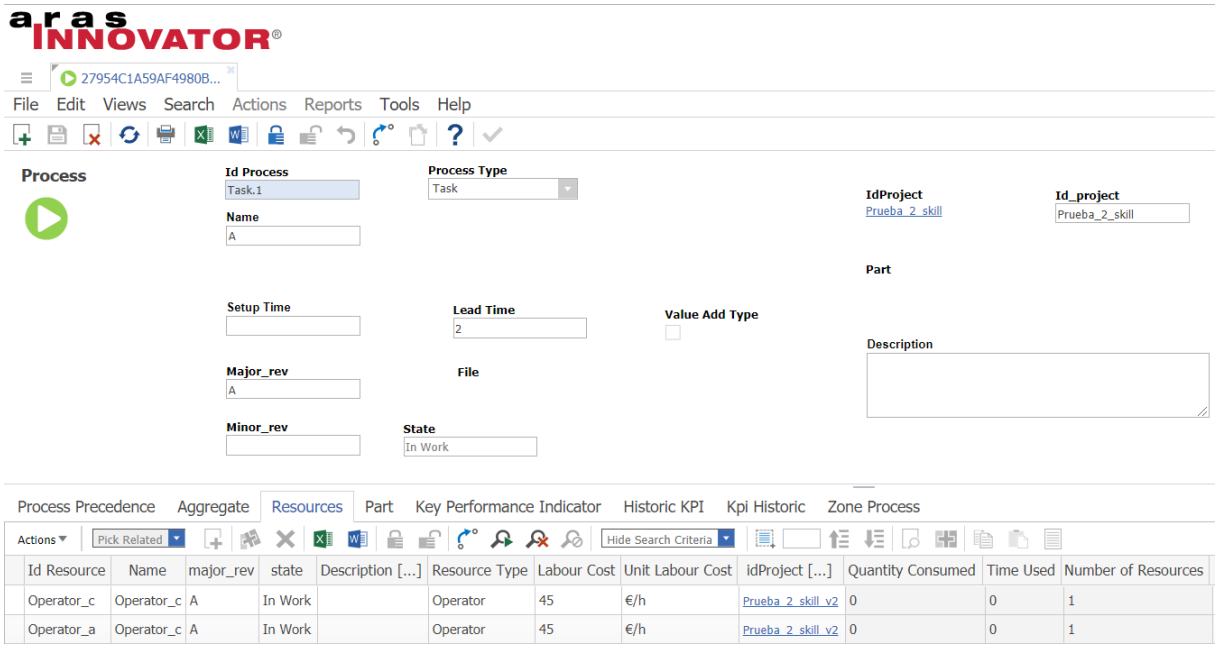

Figura 7.12 "Task.1_Prueba_2_skill_A_In Work"

Si se busca en Aras el proceso “Task.2_Prueba_2_skill_A_In Work”, se puede comprobar que está relacionado con un proceso precedente que es “Task.1_Prueba_2_skill_A_In Work” coincidiendo con la información que se había extraído.

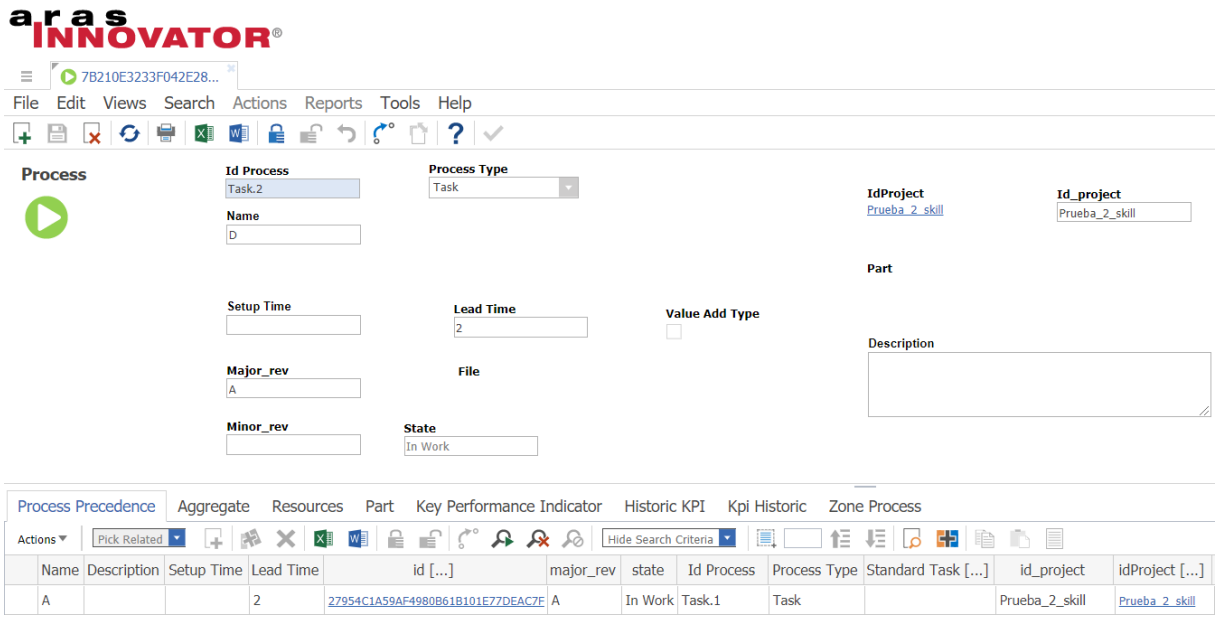

Figura 7.13 “Task.2_Prueba_2_skill_A_In Work”

Con esto quedaría demostrado que se puede extraer la información de Aras, pero ¿cómo se podría añadir información o modificar la que se tiene?

7.2 Importación desde Aras

Por hacer varias demostraciones, se van a hacer tanto modificaciones sobre los individuos existentes como instanciaciones de nuevos individuos.

En primer lugar, se ha visto que el atributo “description”, a pesar de estar definido en la ontología, no se ha extraído de Aras porque su valor estaba vacío y no aportaba información. Pues bien, se va a añadir una descripción a “Task.1_Prueba_2_skill_A_In Work” para demostrar que este atributo está bien trazado.

Para ello, se clica en “Add Data property assertions”, se selecciona el “Data property” “description” y el “Type” “string”, se añade el texto y se pulsa “Aceptar”. En este caso, se ha escrito “Situar en útil y realizar escariado previo a encasquillado”.

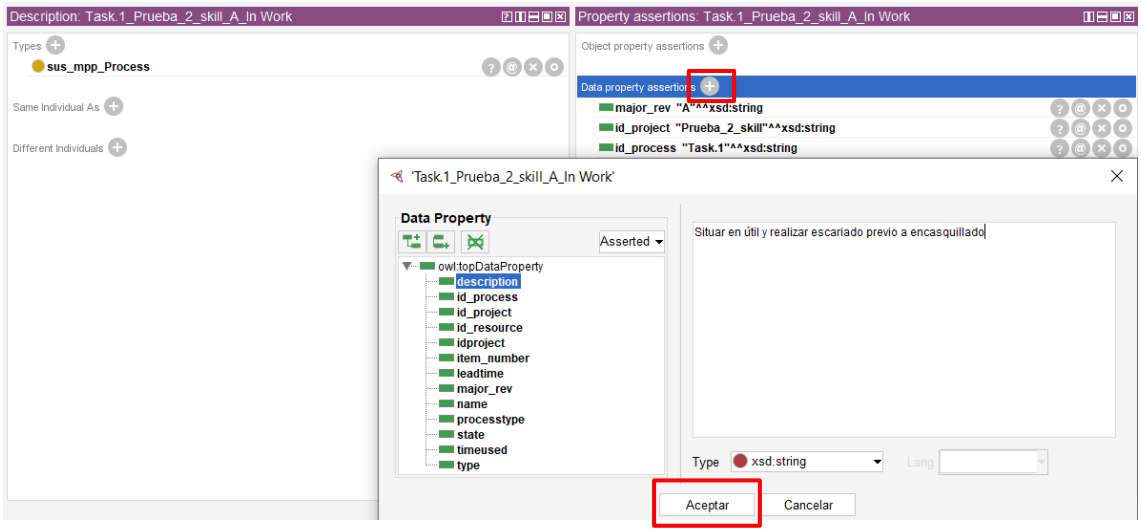

Figura 7.14 Adición de descripción al individuo “Task.1_Prueba_2_skill_A_In Work”

En segundo lugar, se ha detectado que había una errata en la definición del “Operator_a_A_In Work” y se va a aprovechar para corregirla desde Protege en vez de desde Aras.

Puede verse como en el valor del atributo “name” aparece “Operator_c” en vez de “Operator_a”. Para corregirlo, se clica sobre “Edit” en la línea del atributo en cuestión, se cambia el texto y se pulsa “Aceptar”.

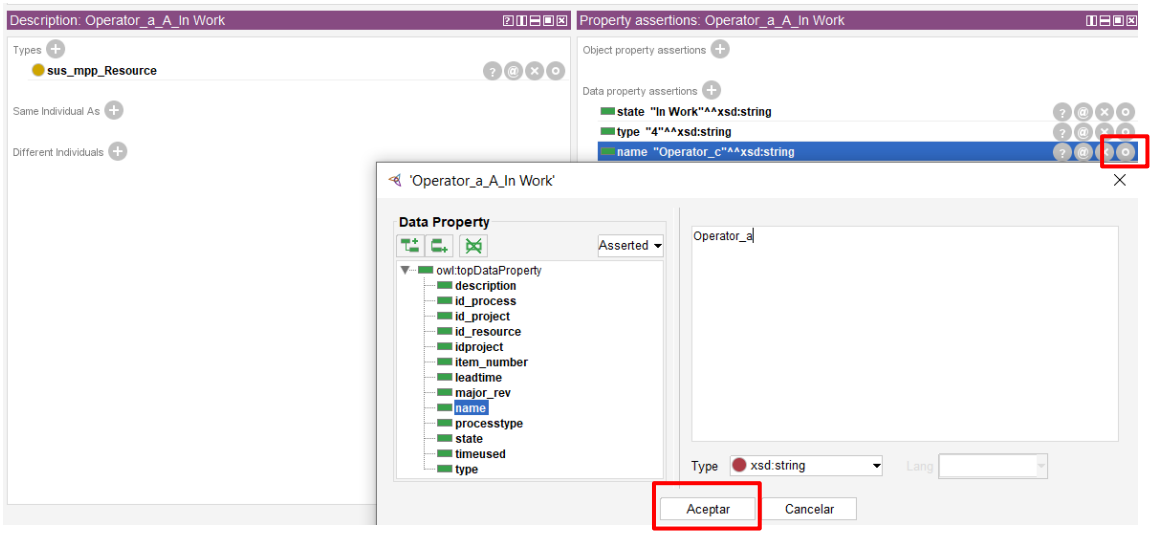

Figura 7.15 Corrección del nombre del individuo “Operator_a_A_In Work”

Ahora se van a instanciar nuevos individuos y, para terminar, se van a crear relaciones entre los individuos nuevos y entre los nuevos y los ya existentes.

Primero se va a instanciar un individuo de clase “sus_mpp_Process”, denominado “Remachado_Prueba_2_skill_A_In Work”, tal y como se mostró en el punto 3.4.2.

Los atributos que se han añadido a este individuo han sido:

- “id_process” = Remachado
- “id_project” = Prueba_2_skill
- “major_rev” = A
- “state” = In Work
- “name” = Remachado s/ I+D-L-113
- “description” = Remachar según I+D-L-113
- “leadtime” = 1.0

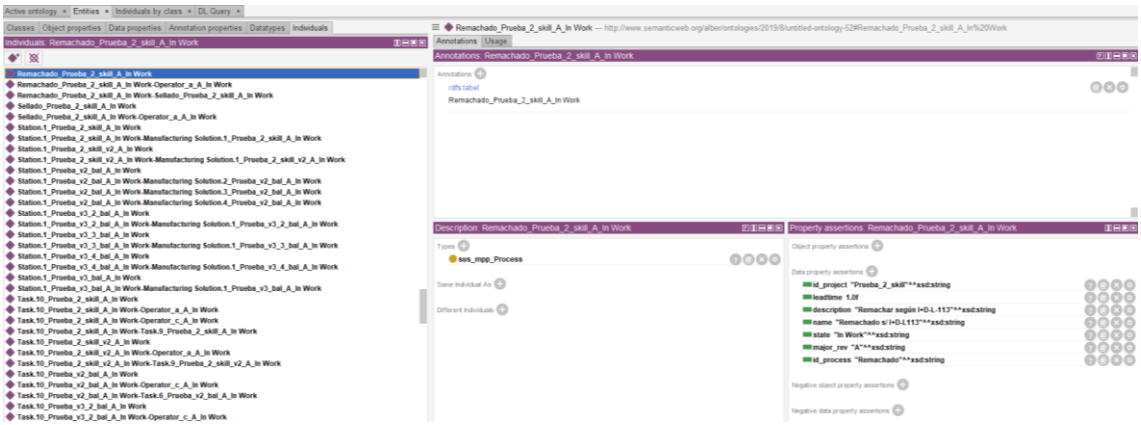

Figura 7.16 Instanciación del individuo “Remachado_Prueba_2_skill_A_In Work”

A continuación, se va a instanciar una relación de tipo “sus_mpp_Process_resource” entre “Remachado_Prueba_2_skill_A_In Work” y “Operator_a_A_In Work”, para que el nuevo proceso tenga un recurso asignado.

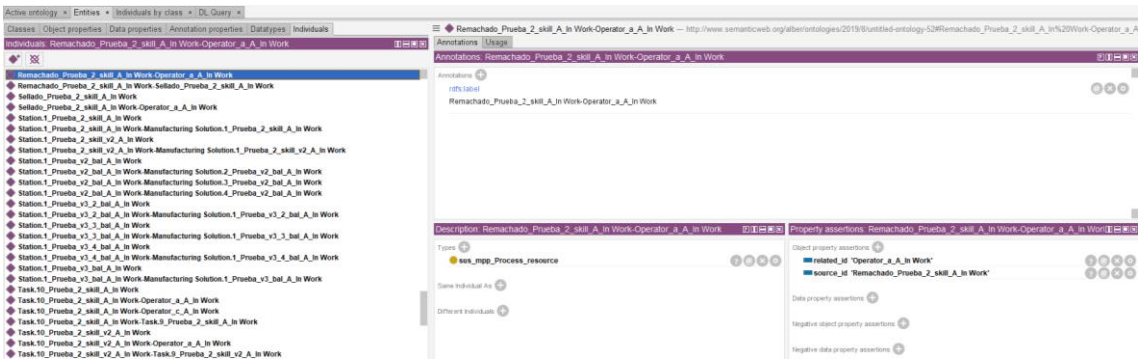

Figura 7.17 Instanciación del individuo “Remachado_Prueba_2_skill_A_In Work-Operator_a_A_In Work”

Ahora se va a instanciar otro individuo de clase “sus_mpp_Process”, denominado “Sellado_Prueba_2_skill_A_In Work”.

Los atributos que se han añadido esta vez han sido:

- “id_process” = Sellado
- “id_project” = Prueba_2_skill
- “major_rev” = A
- “state” = In Work
- “name” = Sellado s/ I+D-P-146
- “description” = Sellar según I+D-P-146
- “leadtime” = 1.0

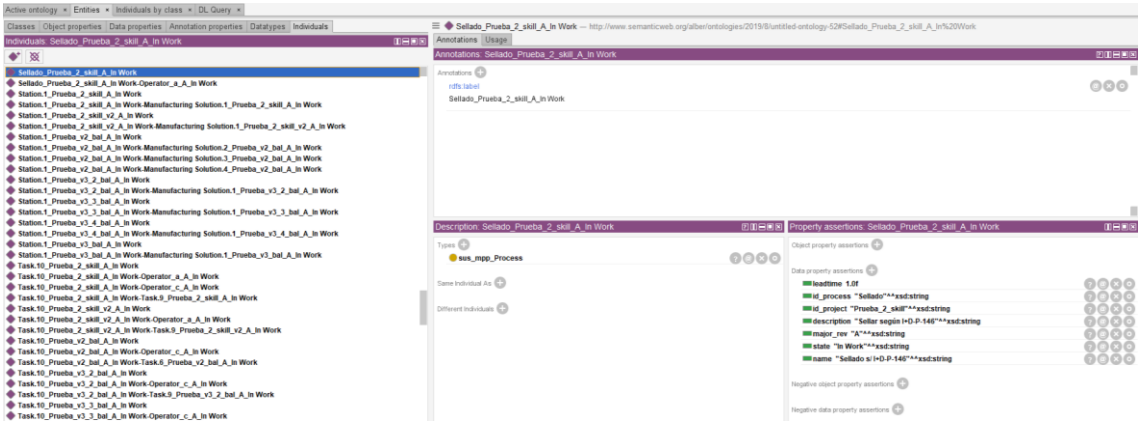

Figura 7.18 Instanciación del individuo “Sellado_Prueba_2_skill_A_In Work”

A continuación, como en el caso anterior, se instancia una relación de tipo “sus_mpp_Process_resource” entre “Sellado_Prueba_2_skill_A_In Work” y “Operator_a_A_In Work”, para que el nuevo proceso tenga un recurso asignado.

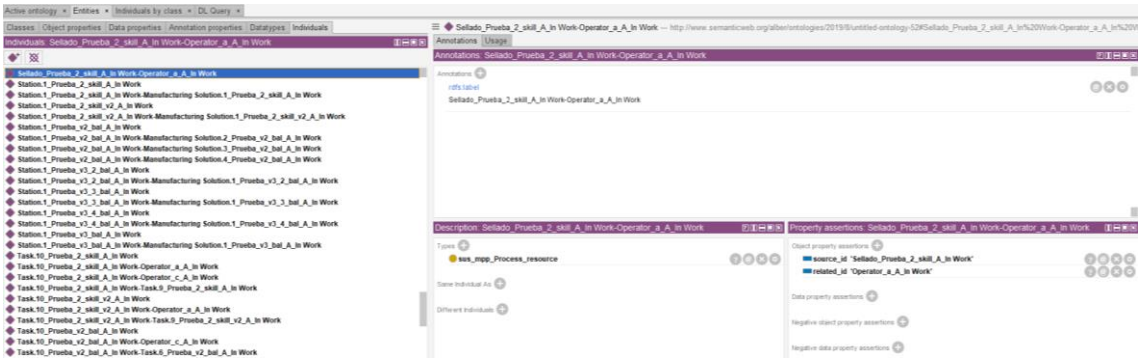

Figura 7.19 Instanciación del individuo “Sellado_Prueba_2_skill_A_In Work-Operator_a_A_In Work”

Para terminar, se va a añadir una relación entre los dos procesos que se han creado nuevos.

Suponiendo que lo que se va a remachar son dos superficies y se requiere aplicar una capa de sellante de interposición previa al remachado, se puede instanciar una relación entre procesos de tipo "sus_mpp_Process_precedence" de manera que se cree una restricción para que no se pueda planificar la operación "Remachado_Prueba_2_skill_A_In Work" antes que "Sellado_Prueba_2_skill_A_In Work".

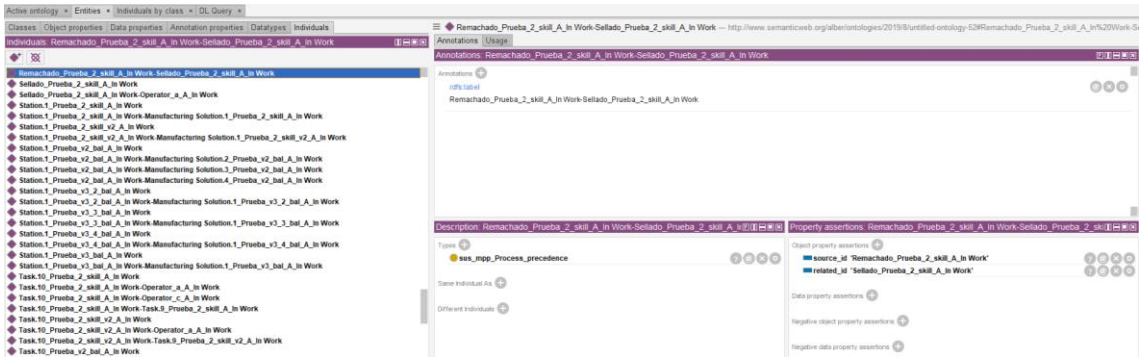

Figura 7.20 Instanciación del individuo "Remachado_Prueba_2_skill_A_In Work-Sellado_Prueba_2_skill_A_In Work".

Una vez hecho esto, ya se puede generar el nuevo XML que se va a utilizar para transmitir la información a Aras.

Tan solo quedaría meter como input la dirección en la que está guardado el XML y arrancar la aplicación.

Esta vez, cuando aparezca la interfaz habrá que clicar el botón "XmlToAras".

Automáticamente, la aplicación actualizará toda la información en Aras.

Se van a tomar unas capturas para demostrar para demostrar que los cambios se han efectuado correctamente.

En primer lugar, se muestra "Task.1_Prueba_2_skill_A_In Work" y se comprueba cómo se ha añadido el atributo "description" y cómo se ha corregido el atributo "name" del "Operator_a_A_In Work" con el que está relacionado.

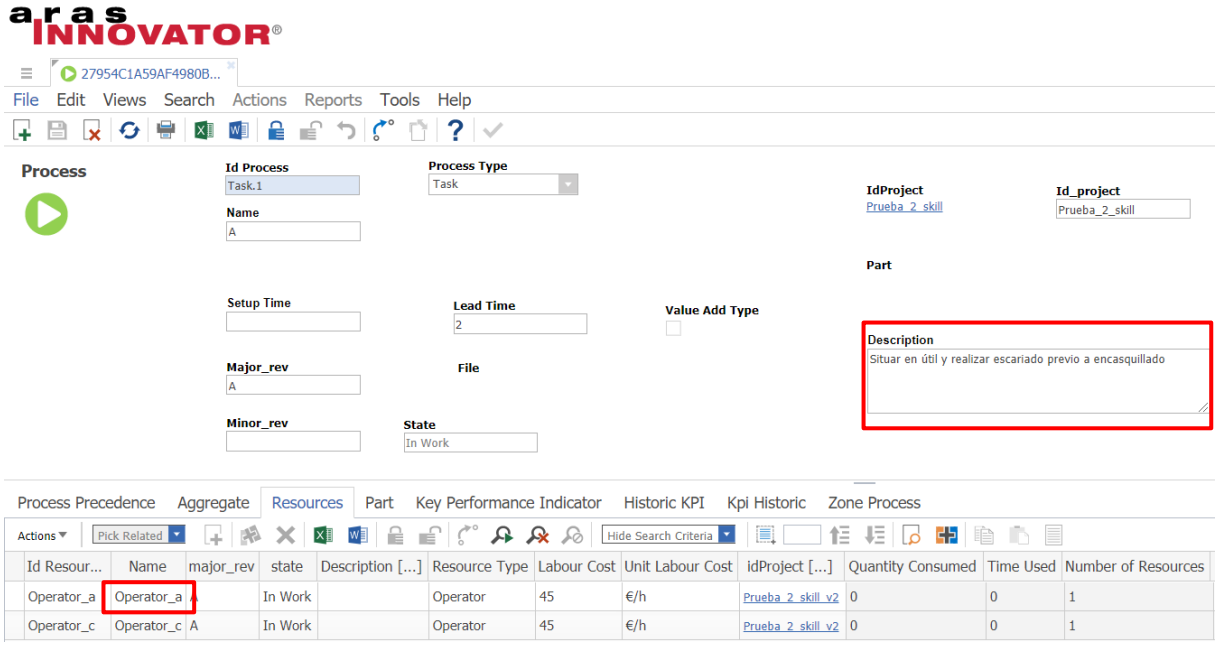

Figura 7.21 "Task.1_Prueba_2_skill_A_In Work"

Ahora se busca el proceso "Sellado_Prueba_2_skill_A_In Work" y se comprueba que se han añadido los siete atributos correctamente.

En la zona inferior se pueden buscar las relaciones de este proceso y comprobar que está relacionado con el

recurso “Operator_a_A_In Work”.

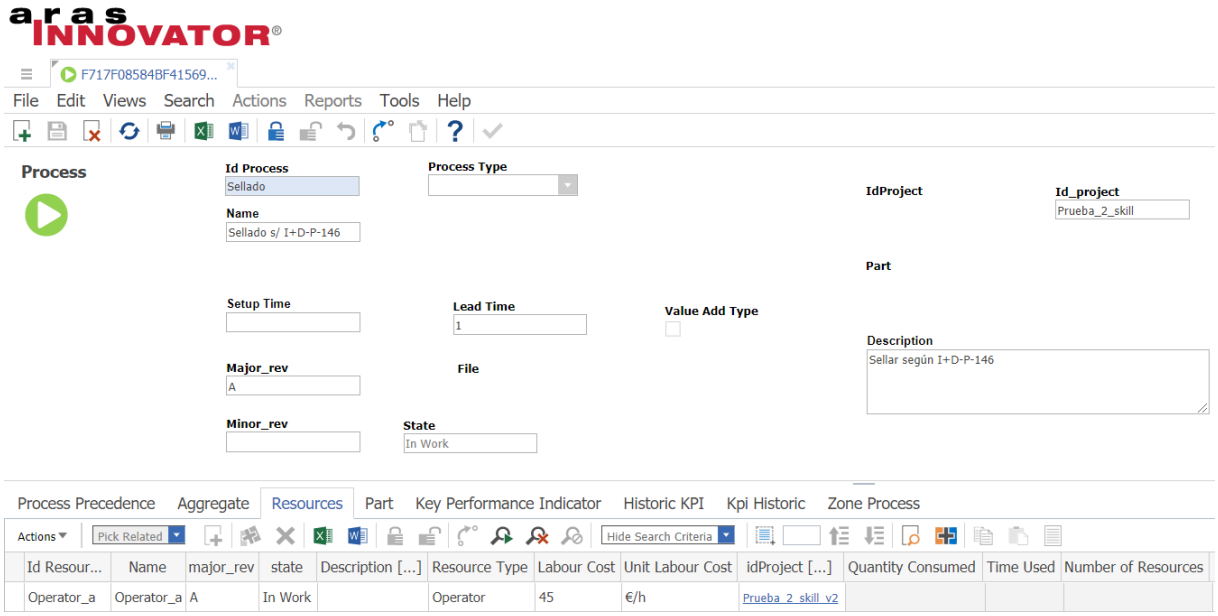

Figura 7.22 “Sellado_Prueba_2_skill_A_In Work”

Para terminar, se busca el proceso “Remachado_Prueba_2_skill_A_In Work” y se comprueba que se han añadido los siete atributos correctamente.

En la zona inferior se pueden buscar las relaciones de este proceso y comprobar que está relacionado con el recurso “Operator_a_A_In Work”, tal y como se ha solicitado.

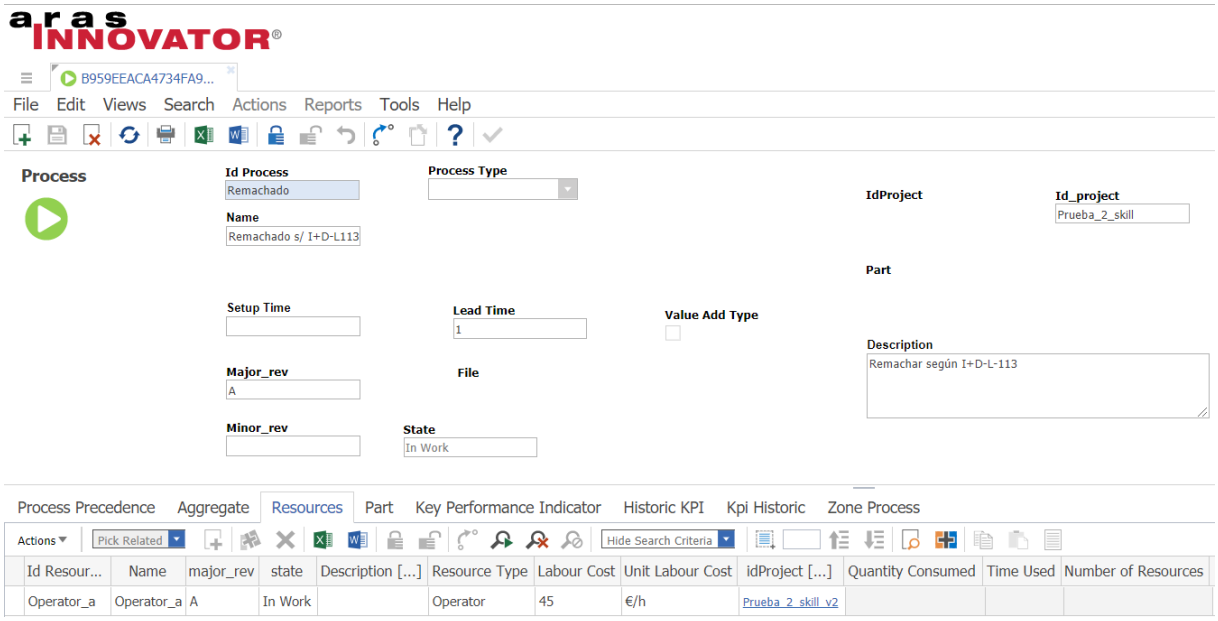

Figura 7.23 “Remachado_Prueba_2_skill_A_In Work” (1/2)

Por último, sólo quedaría comprobar que también tiene una relación con el proceso “Sellado_Prueba_2_skill_A_In Work”.

Para ello, se mira en la pestaña de procesos precedentes si existe alguna restricción y se comprueba que antes de “Remachado_Prueba_2_skill_A_In Work” tiene que realizarse “Sellado_Prueba_2_skill_A_In Work” y que las características que definen dicho proceso son correctas.

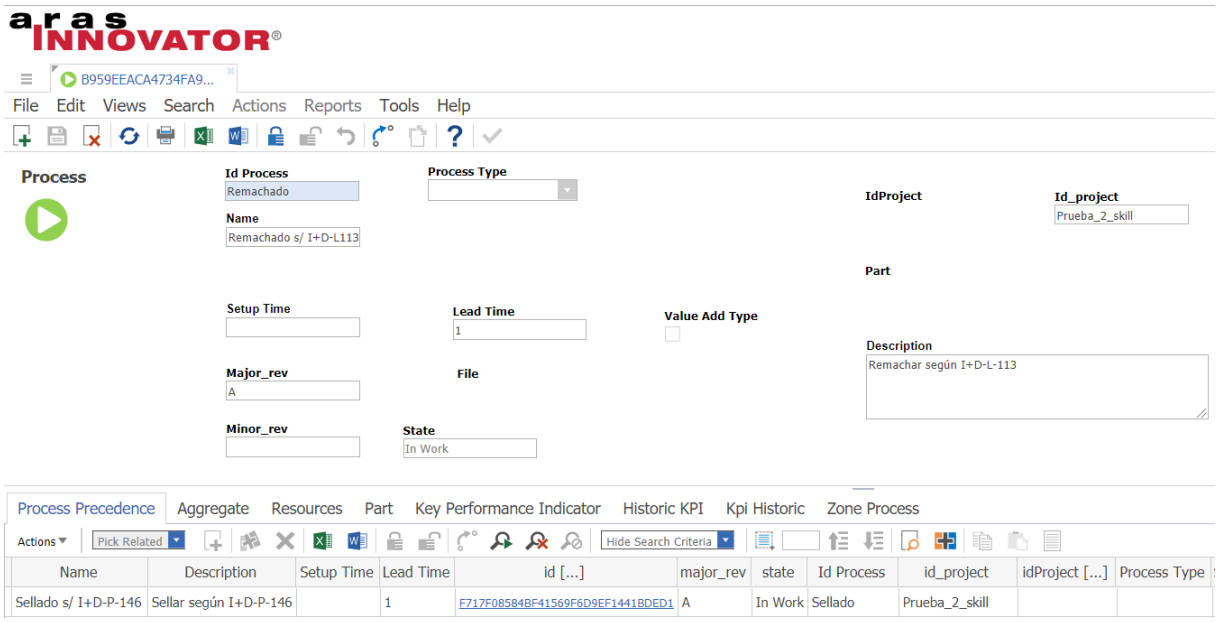

Figura 7.24 “Remachado_Prueba_2_skill_A_In Work” (2/2)

Con esto se habría mostrado que la herramienta funciona y quedaría cerrado el ciclo demostrando que cualquier aplicación, en este caso se ha hecho con Aras, puede hablarse con otra transmitiendo la información a través de una ontología.

8 CONCLUSIONES

Podría empezarse este apartado diciendo que las ontologías no aseguran la interoperabilidad total, pero sí el entendimiento para que la interoperabilidad sea posible.

La idea de este proyecto era demostrar que se puede solucionar el problema de la interoperabilidad haciendo uso de las ontologías. Es decir, demostrar que dos aplicaciones diferentes se pueden entender si son capaces de expresar toda la información en un formato común que sirva para transmitir dicha información.

Efectivamente, se ha demostrado que es posible la comunicación entre aplicaciones diferentes y que la transmisión de la información se hace sin ambigüedades ni pérdida de significado, pero para ello hay que crear funciones que hablen el mismo idioma que la aplicación en cuestión, es decir, desarrollar códigos en el mismo lenguaje de programación.

Eso sí, el hecho de que la información se transmita a través de ontologías utilizando un formato común hace que sea suficiente con que cada aplicación entienda ese formato para poder comunicarse con cualquier otra. Es decir, si se crean funciones para que las aplicaciones entiendan la sintaxis XML, se logrará que todas esas aplicaciones se entiendan entre sí.

No obstante, para lograr resolver el problema de la interoperabilidad no basta con esto. Aún queda pendiente la falta de estandarización. A lo largo del trabajo, han surgido numerosos problemas relacionados con este aspecto como los nombres de los individuos o de las clases de los propios individuos. No se puede encontrar nada si no se sabe lo que se busca. Cada individuo que se crea en Aras lleva asociado un id, pero si no se conoce ese identificador la única manera de encontrarlo es a través de sus atributos clave, por eso cuando se ha creado el individuo en Protege, se ha denominado concatenando los valores de sus atributos clave. Al fin y al cabo, la única manera de resolver ambigüedades es la de crear individuos que sean únicos. La solución no puede ser más sencilla y efectiva, pero debe ser aceptada por todos los usuarios para que funcione correctamente. Con respecto a los nombres de las clases pasa exactamente igual, si no se sabe como le llama esa aplicación a un ItemType, no se va a poder buscar y, por consiguiente, tampoco encontrar. El hecho de tener que saber cómo se denomina un ItemType en una aplicación determinada no parece ser un problema mayor, pero el hecho de que ese mismo ItemType en otra aplicación se llame de manera diferente sí. Además, la competencia actual hace que la evolución y el desarrollo de aplicaciones cada vez sea mayor y se tienda más a una desestandarización que a una

estandarización. Aún así, se ha visto que haciendo uso de la característica modular y la naturaleza extensible que tienen las ontologías se podrían crear clases equivalentes de manera que se tuviera una única definición para un dominio dado que valiera para todas las aplicaciones. Esta podría ser una vía para explorar en futuras investigaciones.

Por último, cabe destacar la utilidad que tendría el hecho de que las aplicaciones de diferente índole se hablaran entre sí. Hoy en día, en el sector aeroespacial se está empezando a susituir planos y listas de partes por modelos full 3D. En estos modelos está recogida toda la información necesaria para fabricar una pieza. Características como el material en el estado de compra, el material en el estado de entrega, la dimensión de la materia prima, la protección superficial, los atributos de la pieza, las tolerancias geométricas, las notas técnicas… Toda la información relativa a una pieza está contenida en su modelo full 3D, no hay más documentación aplicable.

En empresas que se dedican a subcontratar la fabricación de piezas, controlar esta información es vital. En este sector, el propietario suele dividir el producto en paquetes, pero cada paquete puede estar compuesto del orden de 10^3 referencias y analizar la información de esas 10^3 referencias conlleva mucho tiempo, además, tiempo de personal indirecto, lo que se traduce en mucho dinero. Si el modelo full 3D pudiera transmitir toda la información de manera automática a la base de datos de esta compañía, se tendría un ahorro brutal.

Pero hay más, en empresas que se dedican a la fabricación de piezas además de analizar la información para ofertar por ellas, tienen que establecer un plan de fabricación. Para ello, se podría obtener del modelo full 3D la información referente a la fabricación a través de una ontología que ya tuviera integrada los costes asociados y poder cotizar automáticamente un paquete de piezas. Pero es más, si se tuviera un ERP con operaciones tipo creadas, con la información asociada a la fabricación extraída a través de la ontología se podrían generar órdenes de producción de manera automática.

En definitiva, se podría concluir con que las ontologías no solo definen un dominio y almacenan datos, sino que generan conocimiento a partir de la lógica y sirven para transmitir información entre diferentes sistemas haciendo que la interoperabilidad sea posible.

Anexo

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Prueba_2_skill_A_In Work">
<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Mpp_project" />
<idproject rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prueba_2_skill</idproject>
<major_rev rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</major_rev>
<name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prueba_2_skill</name>
<state rdf:datatype="http://www.w3.org/2001/XMLSchema#string">In Work</state>
<rdfs:label>Prueba_2_skill_A_In Work</rdfs:label>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Prueba_2_skill_v2_A_In Work">
<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Mpp_project" />
<idproject rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prueba_2_skill_v2</idproject>
<major_rev rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</major_rev>
<name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prueba_2_skill_v2</name>
<state rdf:datatype="http://www.w3.org/2001/XMLSchema#string">In Work</state>
<rdfs:label>Prueba_2_skill_v2_A_In Work</rdfs:label>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operation.7_Prueba_v2_bal_A_In Work">
<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process" />
<id_process rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operation.7</id_process>
<id_project rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prueba_v2_bal</id_project>
<major_rev rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</major_rev>
<name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operation.7</name>
<processtype rdf:datatype="http://www.w3.org/2001/XMLSchema#string">5</processtype>
<state rdf:datatype="http://www.w3.org/2001/XMLSchema#string">In Work</state>

```
    <rdfs:label>Operation.7_Prueba_v2_bal_A_In Work</rdfs:label>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operation.5_Prueba_2_skill_A_In Work">
    <rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process" />
    <id_process rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operation.5</id_process>
    <id_project rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prueba_2_skill</id_project>
    <major_rev rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</major_rev>
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operation.5</name>
    <processtype rdf:datatype="http://www.w3.org/2001/XMLSchema#string">5</processtype>
    <state rdf:datatype="http://www.w3.org/2001/XMLSchema#string">In Work</state>
    <rdfs:label>Operation.5_Prueba_2_skill_A_In Work</rdfs:label>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operator_a_A_In Work">
    <rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Resource" />
    <id_resource rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operator_a</id_resource>
    <major_rev rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</major_rev>
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operator_c</name>
    <state rdf:datatype="http://www.w3.org/2001/XMLSchema#string">In Work</state>
    <type rdf:datatype="http://www.w3.org/2001/XMLSchema#string">4</type>
    <rdfs:label>Operator_a_A_In Work</rdfs:label>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operator_c_A_In Work">
    <rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Resource" />
    <id_resource rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operator_c</id_resource>
    <major_rev rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</major_rev>
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Operator_c</name>
    <state rdf:datatype="http://www.w3.org/2001/XMLSchema#string">In Work</state>
    <type rdf:datatype="http://www.w3.org/2001/XMLSchema#string">4</type>
    <rdfs:label>Operator_c_A_In Work</rdfs:label>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operation.7_Prueba_v2_bal_A_In Work-
Task.7_Prueba_v2_bal_A_In Work">
    <rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_aggregate" />
    <related_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.7_Prueba_v2_bal_A_In Work" />
    <source_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operation.7_Prueba_v2_bal_A_In Work" />
    <rdfs:label>Operation.7_Prueba_v2_bal_A_In Work-Task.7_Prueba_v2_bal_A_In Work</rdfs:label>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operation.5_Prueba_2_skill_A_In Work-
Task.7_Prueba_2_skill_A_In Work">
    <rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_aggregate" />
    <related_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.7_Prueba_2_skill_A_In Work" />
    <source_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operation.5_Prueba_2_skill_A_In Work" />
    <rdfs:label>Operation.5_Prueba_2_skill_A_In Work-Task.7_Prueba_2_skill_A_In Work</rdfs:label>
</owl:NamedIndividual>
```

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Remachado s/I+D-L-113_Prueba_2_skill_A_In Work-Sellado s/I+D-P-146_Prueba_2_skill_A_In Work">

<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_precedence" />

<related_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Sellado s/I+D-P-146_Prueba_2_skill_A_In Work" />

<source_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Remachado s/I+D-L-113_Prueba_2_skill_A_In Work" />

<rdfs:label>Remachado s/I+D-L-113_Prueba_2_skill_A_In Work-Sellado s/I+D-P-146_Prueba_2_skill_A_In Work</rdfs:label>

</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.9_Prueba_v3_2_bal_A_In Work-Task.5_Prueba_v3_2_bal_A_In Work">

<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_precedence" />

<related_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.5_Prueba_v3_2_bal_A_In Work" />

<source_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.9_Prueba_v3_2_bal_A_In Work" />

<rdfs:label>Task.9_Prueba_v3_2_bal_A_In Work-Task.5_Prueba_v3_2_bal_A_In Work</rdfs:label>

</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.9_Prueba_v3_2_bal_A_In Work-Operator_c_A_In Work">

<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_resource" />

<related_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operator_c_A_In Work" />

<source_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.9_Prueba_v3_2_bal_A_In Work" />

<timeused rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0</timeused>

<rdfs:label>Task.9_Prueba_v3_2_bal_A_In Work-Operator_c_A_In Work</rdfs:label>

</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.12_Prueba_v3_2_bal_A_In Work-Operator_c_A_In Work">

<rdf:type rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#sus_mpp_Process_resource" />

<related_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Operator_c_A_In Work" />

<source_id rdf:resource="http://www.semanticweb.org/alber/ontologies/2019/8/untitled-ontology-52#Task.12_Prueba_v3_2_bal_A_In Work" />

<timeused rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0</timeused>

<rdfs:label>Task.12_Prueba_v3_2_bal_A_In Work-Operator_c_A_In Work</rdfs:label>

</owl:NamedIndividual>

Bibliografía

[i] F. Ahmed and S. Han. Interoperability of product and manufacturing information using ontology.

[ii] Penciuc, D., Durupt, A., Belkadi, F., Eynard, B., & Rowson, H. Towards a PLM interoperability for a collaborative design support system.

[iii] Amann, K. Product lifecycle management: Empowering the future of business.

[iv] https://www.product-lifecycle-management.com/

[v] https://www.batchmaster.co.in/blog/a-brief-guide-on-product-lifecycle-management/

[vi] Damjanović V., Behrendt W., Plößnig M., Holzapfel M. Developing Ontologies for Collaborative Engineering in Mechatronics.

[vii] Mas, F., Menéndez, J. L., Oliva, M., Gómez, A., & Ríos, J. Collaborative engineering paradigm applied to the aerospace industry.

[viii] Moones, E., Vosgien, T., Kermad, L., El Mhamedi, A., & Figay, N. Plm standards modelling for enterprise interoperability: A manufacturing case study for erp and mes systems integration based on isa-95.

[ix] Penciuc, D., Durupt, A., Belkadi, F., Eynard, B., & Rowson, H. Towards a PLM interoperability for a collaborative design support system.

[x] Panetto, H., Dassisti, M., & Tursi, A. ONTO-PDM: Product-driven ONTOlogy for Product Data Management interoperability within manufacturing process environment.

[xi] Mun, D., Hwang, J., Han, S., Seki, H., & Yang, J. (2008). Sharing product data of nuclear power plants across their lifecycles by utilizing a neutral model.

[xii] Mason, H. ISO10303-STEP A Key Standard for Global Market.

[xiii] ISO 10303-1. Industrial Automation Systems and Integration–Product Data Representation and Exchange–Part 1: Overview and Fundamental Principles.

[xiv] Rachuri, S., Subrahmanian, E., Bouras, A., Fenves, S. J., Foufou, S., & Sriram, R. D. Information

sharing and exchange in the context of product lifecycle management: Role of standards.

[xv] Padillo, A., Racero, J., Oliva, M., & Mas, F. (2017, July). Design and implementation of a prototype for information exchange in digital manufacturing processes in aerospace industry.

[xvi] Harjunkoski, I., & Bauer, R. Sharing data for production scheduling using the ISA-95 standard.

[xvii] Figay, N., Ghodous, P., Abdul Ghafour, S., Ferreira Da Silva, C., & Expósito, E. Pragmatic PLM process interoperability for aeronautic, space and defence DMN.

[xviii] Song, H., Roucoules, L., Eynard, B., & Lafon, P. Interoperability between a Cooperative Design Modeler and a CAD System: Software Integration versus Data Exchange.

[xix] Lukas, U. V., & Nowacki, S. (2005). High level integration based on the PLM services standard.

[xx] Wang, Y., Ge, J., Shao, J., & Han, S. Research on Web Service-Based Interoperability of Heterogeneous PDM Systems.

[xxi] Choi, S. S., Yoon, T. H., & Noh, S. D. XML-based neutral file and PLM integrator for PPR information exchange between heterogeneous PLM systems.

[xxii] Gunpinar, E., & Han, S. Interfacing heterogeneous PDM systems using the PLM Services.

[xxiii] Jiang, P., Shao, X., Qiu, H., Gao, L., & Li, P. A Web services and process-view combined approach for process management of collaborative product development.

[xxiv] https://www.w3.org/2001/sw/#owl

[xxv] https://www.w3.org/OWL/

[xxvi] https://www.w3.org/TR/2012/REC-owl2-overview-20121211/#Introduction

[xxvii] https://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.2

[xxviii] https://www.w3.org/2007/09/OWL-Overview-es.html#ref-rdf-schema

[xxix] https://www.w3.org/TR/rdf-schema/

[xxx] https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/#basic

[xxxi] A. Talhi, V. Fortineau, J-C Huet and S. Lamouri. Ontology for cloud manufacturing based Product Lifecycle Management. A. Talhi, V. Fortineau, J-C Huet and S. Lamouri.

[xxxii] https://www.manufacturingterms.com/Bill_of_materials_BOM.html

[xxxiii] http://www.naamsstandards.org/publications/nascontents.htm

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

[xxxv] https://notepad-plus-plus.org/

[xxxvi] Z.Usman, R. Young, N. Chungoora, C. Palmer, K. Case and J. Harding. A manufacturing core concepts ontology for producto lifecycle interoperability.

[xxxvii] M. Ahmad. An ontology-based approach for integrating engineering workflows for industrial assembly automation systems

[xxxviii] https://www.aras.com/