Proyecto Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

Diseño de un sistema colaborativo para la gestión de modelos de Ingeniería basado en ontologías. Aplicación a modelos de producción aeroespacial.

Autor: Daniel Gómez Reyes

Tutor: Jesús Racero Moreno

Dep. de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Sevilla, 2018

Proyecto Fin de Máster
Máster Universitario en Ingeniería Aeroáutica

Diseño de un sistema colaborativo para la gestión de modelos de Ingeniería basado en ontologías. Aplicación a modelos de producción aeroespacial.

Autor:
Daniel Gómez Reyes

Tutor:
Jesús Racero Moreno
Profesor titular

Dep. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Máster: Diseño de un sistema colaborativo para la gestión de modelos de Ingeniería basado en ontologías. Aplicación a modelos de producción aeroespacial.

Autor: Daniel Gómez Reyes

Tutor: Jesús Racero Moreno

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

Agradecimientos

Agradecer todo el apoyo recibido por parte de mi familia y amigos en esta última etapa académica que cierra mi paso como estudiante por la ETSI aunque nunca se deje de aprender.

Agradecer también a mi tutor en este Trabajo de Fin de Máster, Jesús Racero Moreno, por haberme dado la oportunidad de llevarlo a cabo, su dedicación y por tener la paciencia de "aguantarme" durante estos últimos meses.

Daniel Gómez Reyes

Sevilla, 2018

Resumen

Este trabajo consiste en el estudio la influencia de un modelo ontológico en el desarrollo de un proyecto de fabricación. Se abordarán la definición de la ontología, las herramientas utilizadas para su elaboración y su almacenamiento, así como la gestión tanto de los modelos como de las instancias que pondrán en evidencia los posibles errores en la definición de la ontología y destacando también su carácter interoperable.

Abstract

This work conists in the study of the influence of an ontological model on the development of a manufacturing project. The definition of the ontology, the tools used for its elaboration and its storage will be approached as well as the model management and the instances which will test potential errors in the ontology definition and highlight its interoperable nature.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

El crecimiento de la complejidad de los sistemas requiere la implementación de nuevos modelos de desarrollo manteniendo los costes, tiempo y calidad bajo control. El enfoque tradicional centrado en documentación y pruebas no es compatible con los nuevos sistemas ingenieriles, distribuidos y colaborativos. Model Based Systems Engineering (MBSE) afronta esta complejidad buscando integrar soluciones basadas en modelos y relacionadas entre sí.

En los últimos años, Model Based System Engineering (MBSE) ha cobrado gran interés como técnica para la definición y desarrollo de sistemas y proyectos complejos y extensos. Las técnicas aplicadas han sido efectivas minimizando errores y ayudando a los ingenieros al desarrollo de sistemas consistentes.

Cada vez en mayor cantidad de proyectos se opta por implementar la denominada MBSE, es decir, Ingeniería basada en Modelos, dejando atrás el formato físico que proporcionaban los documentos y favoreciendo la colaboración y comunicación entra las distintas partes que conforman el proyecto, incluidos los clientes del mismo, así como una mejora en la calidad gracias a la temprana identificación de requisitos y especificaciones lo que conlleva un aumento de la productividad debido también a la propia interacción entre los componentes del proyecto [1].

Sin embargo, surge el **problema de la falta de homogeneidad** en cuanto a definición de modelos en muchos casos similares pero con distinta notación, más aún cuando en un mismo proyecto toman parte distintas entidades, cada una desarrollando su tipo de aplicación y/o modelo con una herramienta diferente, lo que **dificulta una comunicación fluida y rápida**. Como solución a esta problemática, nace el **interés de diseñar técnicas que ayuden a la interoperabilidad** entre los distintos modelos independientemente de la herramienta que se use para la definición del mismo.

Los primeros trabajos en relación a MBSE se centran en definir y estandarizar metodologías de diseño, como por ejemplo SysML que, basado en UML, busca utilizar técnicas de modelado de la ingeniería del software para la definición del sistema.

Aunque SysML se ha convertido en referencia en el diseño de modelos, los ingenieros siguen usando técnicas tradicionales de modelado (Cmap, IDEF, entre otros). Estas técnicas siguen siendo tan válidas como las últimas propuestas por lo que **uno de los problemas que reside e intenta dar respuesta este trabajo consiste en proporcionar un marco de interoperabilidad entre herramientas de modelado que además permita, de forma colaborativa, integrar diseños existentes y extenderlos**, dicho marco está comprendido por modelos **ontológicos**, los cuales favorecen la interoperabilidad entre herramientas gracias a la inclusión de una semántica que enriquece la ontología. Si bien la ontología representa de forma unívoca una estructura, la semántica asociada permite que sea abordada por diferentes herramientas.

Por otra parte, aparece la **problemática de la gestión de la gran cantidad de modelos que conforman un proyecto**. En este aspecto cobran una gran importancia los sistemas PLM (Product Lifecycle Management), herramientas que gestionan por completo el ciclo de vida de un producto y, como extensión, la de un proyecto. Estos sistemas tienen la capacidad de almacenar modelos, poder reutilizar dichos modelos, permiten su customización para adaptarse a los ususarios y proporcionan un entorno colaborativo.

1.2 Objetivos

El objetivo del trabajo consiste en **el análisis, diseño y customización de un sistema PLM para el soporte de modelos basados en ingeniería**. La base para la definición de modelos de ingeniería son **ontologías**, el sistema PLM permitirá proporcionar la interoperabilidad necesaria para la definición de modelos con diferentes técnicas. De tal forma que, independientemente de la herramienta que se use para la definición del modelo, sus clases, elementos, etc., la ontología sea capaz de reconocer cada uno de esos elementos y clasificarlos correctamente. También se llevará a cabo un estudio de cómo afecta la definición de una determinada ontología al desarrollo de un proyecto y hasta qué punto limita o favorece su evolución.

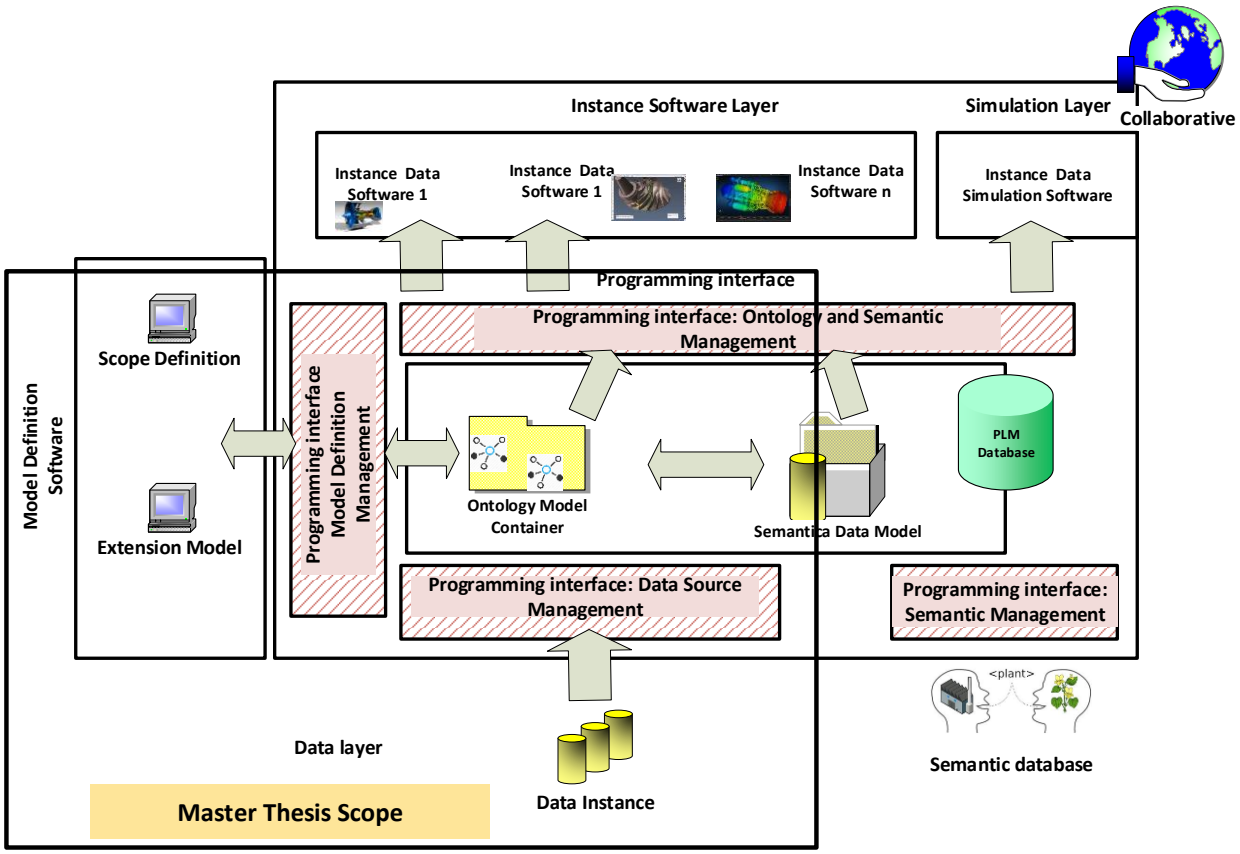


Figura 1–1. Scope del proyecto

Para alcanzar nuestro objetivo final, se deben de cumplir los hitos parciales que siguen:

- Realización de revisión de la literatura, donde se realizará un estudio sobre la evolución de la metodología aplicada al mundo ingenieril desde la ingeniería secuencial hasta MBSE y la implantación de ontologías, así como las distintas variantes.
- Diseño metodológico que se vaya a desarrollar, donde se expondrán la arquitectura del mismo así como las posibles revisiones.
- Implementación del diseño, donde se explicará el modelo de datos desarrollado en la herramienta.
- Aplicación y comprobación con una serie de instancias con las que se elaborará un informe con los resultados obtenidos.

2 REVISIÓN DE LA LITERATURA

2.1 MBSE

2.1.1 Evolución metodología

La metodología con la que desarrollar los distintos proyectos ingenieriles ha ido evolucionando con el paso del tiempo desde la ingeniería secuencial, pasando por la concurrente y adentrándonos cada vez más en la denominada ingeniería colaborativa.

Como su nombre indica, la ingeniería secuencial se basa en que cada fase del proceso se lleva a cabo consecutivamente, es decir, la siguiente etapa no se inicia hasta haber finalizado completamente la anterior. Es un método que destaca por su simpleza, y disciplina puesto que el esfuerzo de gestión no es excesivo; sin embargo, también se caracteriza por ser lento y la inexistencia de cooperación entre departamentos.

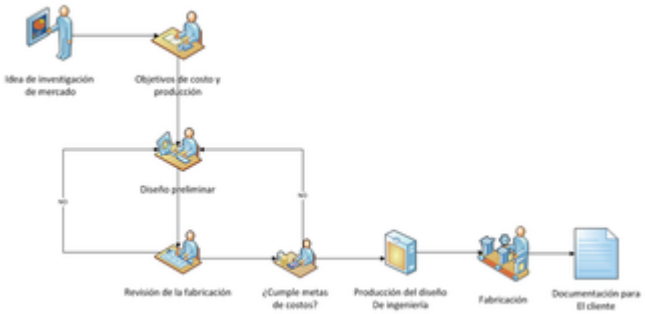


Figura 2–1. Ingeniería secuencial [2]

Partiendo de las carencias de la ingeniería secuencial y con el objetivo de reducir costes y tiempo en el desarrollo de los productos, surgió el concepto de ingeniería concurrente, en la que todos los procesos se rigen por dos ideas principales [3]:

- La totalidad de los componentes y elementos del producto final deben ser tenidos en cuenta desde las fases iniciales del proceso.
- Todas las actividades de diseño previas a la fabricación del producto han de ocurrir simultáneamente.

"La ingeniería concurrente es un esfuerzo sistemático para un diseño integrado, concurrente del producto y de su correspondiente proceso de fabricación y servicio. Pretende que los encargados del desarrollo desde un principio, tengan en cuenta todos los elementos del Ciclo de Vida del Producto (CVP), desde el diseño conceptual hasta su disponibilidad, incluyendo calidad, costo y necesidad de los usuarios" [4].

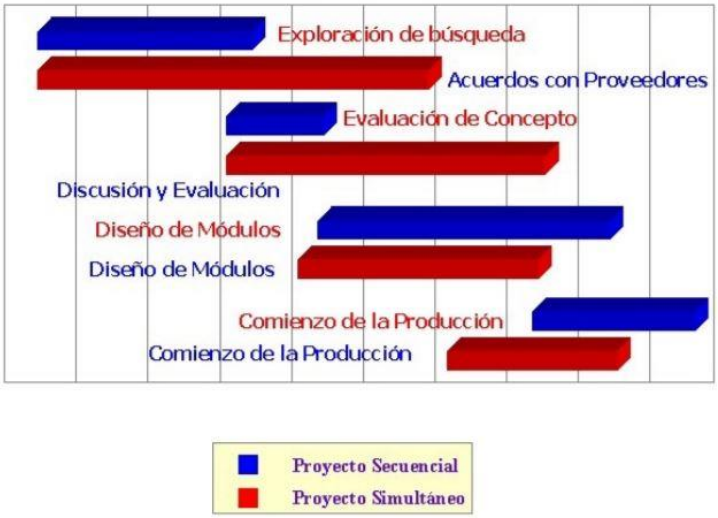


Figura 2–2. Ingeniería concurrente vs. Ingeniería secuencial

La ingeniería concurrente engloba cuatro conceptos fundamentales [2]:

1. Ciclo de vida del producto. Se define como el número de etapas por las que pasa un producto o componentes desde su creación hasta su fin de vida.
2. Modelos de diseño del producto. De entre los muchos disponibles, se destacan el Modelo del ciclo básico de diseño y el modelo de etapas.
3. Arquitectura del producto. Concretada mediante el establecimiento de una serie de reglas como definición de módulos, interfaces y plataformas.
4. Flujo de información en el proceso de diseño. Es fundamental la comunicación entre todos los miembros que forman parte en la elaboración del producto, incluyendo también a todas aquellas personas que se deberían satisfacer con el mismo.

Si bien es cierto que la ingeniería concurrente supuso un paso significativo, siguen existiendo barreras a la hora de organizar proyectos, trabajar en equipo y el aprovechamiento de las nuevas tecnologías (IoT). Es por ello, que aparece el término de ingeniería colaborativa, el cual está directamente relacionado con la Model-Based Systems Engineering (MBSE), es decir, sistemas de ingeniería basados en modelos.

MBSE es un paradigma que enfatiza la aplicación de principios rigurosos de modelado visual a las actividades de Ingeniería de Sistemas a lo largo del Ciclo de Vida de Desarrollo del Sistema (SDLC). Dichas actividades incluyen, entre otras: análisis de requisitos, validación y verificación, análisis funcional y asignaciones, análisis de funcionamiento y especificación de la estructura del sistema [5].

Durante la última mitad de siglo ha habido aportes fundamentales para el establecimiento de enfoques basados en modelos para la descripción, análisis y diseño de sistemas. Dichos sistemas se definen como un conjunto de objetos o entidades que poseen atributos y están relacionadas entre sí (tanto los objetos como los atributos). Si bien, se ha buscado en primer lugar una formalización matématica del modelado de sistemas, por ejemplo con la teoría de modelos de primer orden de Tarski [6]; no obstante, en la práctica se opta por un enfoque menos formal y más intuitivo mediante el uso de lenguajes de modelado gráficos. Como consecuencia, empezaron a surgir diferentes metodologías MBSE y que hasta hoy día siguen evolucionando, entre ellas destacan [7], [8]:

- Metodologías MBSE de IBM [7] [8]

Entre las distintas metodologías desarrolladas por IBM se encuentra “Harmony” enfocada a sistemas de ingeniería. Harmony se usa para el desarrollo de sistemas integrados y software. Fue diseñado para no depender de herramientas ni vendedores específicos. Asimismo el proceso es similar al de “V&V”, como se puede obersarvar en la Figura 2–3. Proceso de desarrollo Harmony , y tiene los siguientes objetivos clave:

- Indentificar las funcionalidades del sistemas requeridas

- Identificar estados y modos de sistemas asociados
- Asignar las funcionalidades y modos del sistema a una arquitectura física

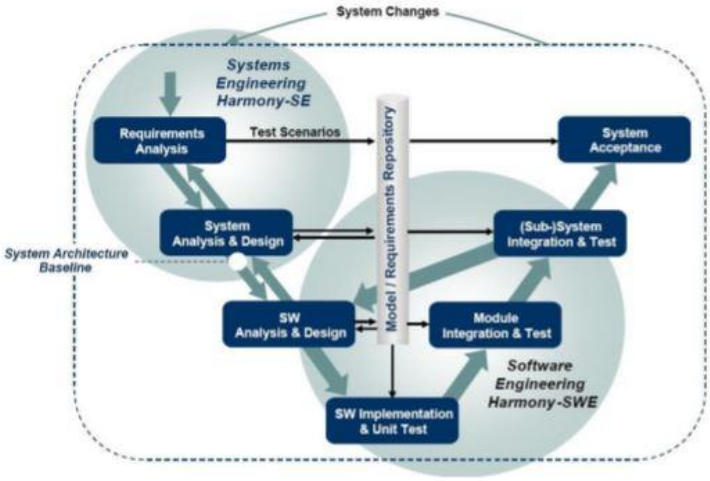


Figura 2–3. Proceso de desarrollo Harmony [7] [8]

Tanto el flujo de trabajo como los productos dentro de Harmony son sometidos a los siguientes procesos elementales:

- Análisis de requisitos
- Análisis funcional del sistema
- Diseño de la arquitectura

- INCOSE Metodología de sistemas de ingeniería orientada a objetos (OOSEM) [7] [8]

OOSEM integra un enfoque basado en modelos descendentes ("top-down") que usa OMG SysML para afianzar las especificaciones, análisis, diseños y verificación de sistemas. Sus principales objetivos son:

- Captura y análisis de requisitos e información de diseño para especificar sistemas complejos
- Integración con software, hardware y otros métodos orientados a objetos
- Dar soporte a la evolución del sistema

Los principios básicos de OOSEM incluyen prácticas esenciales para sistemas de ingeniería como: Desarrollo de Producto Integrado (IPD) y ciclo de vida V&V. Asimismo, incluye las siguientes actividades:

- Análisis de las necesidades de las partes interesadas
- Definición de requisitos del sistema
- Definición de la arquitectura lógica
- Optimización y evaluación de alternativas
- Validación y verificación del sistema

- Metodología MBSE de Vitech [7] [8]

Metodología basada en sistemas de ingeniería concurrente, sus actividades son:

- Análisis de requisitos

- Análisis de comportamiento funcional
- Síntesis de arquitectura
- Validación y verificación del diseño

Esta metodología usa un enfoque por capas ("The Onion Model") para el diseño del sistema y una base de datos común de diseños ("System Design Repository").

- Iniciativa MBSE INCOSE [7] [8]

Iniciativa organizada por INCOSE en 2007 con el objetivo de establecer e integrar metodologías MBSE en los sistemas de ingeniería existentes. Un aspecto fundamental en este proyecto es la transición de procesos centrados en documentos a procesos centrados en modelos. En concreto, con esta iniciativa, se pretende que MBSE sea la aplicación formal de modelos que de soporte a los sistemas de ingeniería desde sus primeras etapas en el diseño conceptual, continuando en las fases de desarrollo y hasta el final de las etapas de ciclo de vida del producto. Dicha iniciativa se encuentra investigando sobre los siguientes temas:

- Interoperabilidad de modelos y simulaciones y cómo dichos modelos interaccionan entre sí a lo largo del cilco de vida del sistema.
- Modelado para sistemas espaciales
- Modelado telescópico
- Arquitectura de aparatos biomédicos
- Sistemas de observación de la Tierra (GEOSS) [9]

2.1.2 Técnicas de modelado

Como se mencionó anteriormente, la mejor forma para expresar estos MBSE es mediante lenguajes gráficos. El precursor de este tipo de lenguajes es el **Lenguaje Unificado de Modelado** (UML) el cual es el estándar para el desarrollo de software. Es un lenguaje gráfico que provee de semánticas y notaciones para la resolución de problemas orientados a objetos. Los artefactos de UML pueden ser aplicados en sistemas de ingeniería, entre ellos [8]: casos de uso, diagramas de clase, diagramas de paquetes, diagramas de secuencia y diagramas de transición de estados.

Los diagramas de casos de usos representan una visión externa del sistema, donde las interacciones entre los "actores" con el sistema reflejan gráficamente los requisitos funcionales. En la Figura 2–4, se muestran los elementos básicos de un diagrama de caso de uso UML. Estos elementos son el "actor" (representado por un monigote con su denominación), el sistema (rectángulo con su nombre en el interior) y las interacciones entre el actor y el sistema (denotadas por líneas) y las funciones por las que el actor usa al sistema (representado por un óvalo en el interior del sistema).

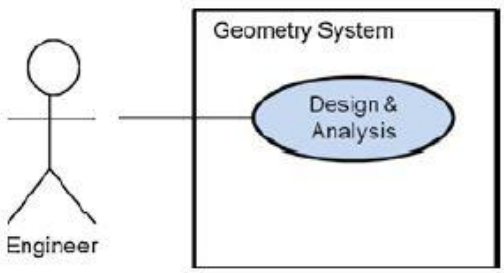


Figura 2–4. Diagrama caso de uso UML

Las clases no son más que abstracciones[1] de las entidades del sistema. En el desarrollo de software, se pueden obtener objetos como instancias de una clase. Dichas clases contienen una serie de atributos y métodos que pueden ser llamados en el diagrama. Los métodos se distinguen de los atributos por los paréntesis que siguen al nombre.

Un modelo representado por clases se puede organizar mediante el uso de paquetes, los cuales agrupan dichos diagramas así como elementos del modelo. Estos paquetes y sus agrupaciones conforman componentes en el modelo, cuyas dependencias son claramente identificadas a través de los diagramas de paquetes. Esta forma de modelar componentes y sus relaciones favorece la creación de una estructura que puede ser usada como parte de la especificación de la arquitectura del sistema.

A partir de este Lenguaje Unificado de Modelado, surge el denominado **SysML**, lenguaje gráfico más enfocado al modelado de sistemas de ingeniería. Como se dijo anteriormente, UML se creó orientado al desarrollo de software, por tanto, carece de la semántica y notación que dan soporte a los sistemas de ingeniería en el análisis, diseño, verificación y validación de hardware, software, datos, paramétricos, personal, procedimientos y establecimientos; asimismo, SysML es más sencillo en el sentido de que no incluye determinados diagramas y construcciones centradas en creación de software. La Figura 2–5, ilustra los tipos de diagramas usados por SysML: hay tres tipos principales de diagramas: **comportamiento, estructura y requisitos**. Como puede observarse, el bloque de diagrama de requisitos es un tipo nuevo de diagrama no presente en UML y que muestra los requisitos del sistema y sus relaciones con otros elementos, de mucha utilidad en el proceso V&V. En el siguiente nivel, destacan modificaciones a los diagramas de actividades (control de funcionamiento de operaciones, implementación de estamentos "if"), y de clases apareciendo diagrama de definición de bloques (muestra la estructura del sistema como conjunto de componentes así como las propiedades de cada uno, sus operaciones y relaciones) e interno de bloques (muestra la estructura interna de los componentes del sistema, incluyendo sus partes y conectores).

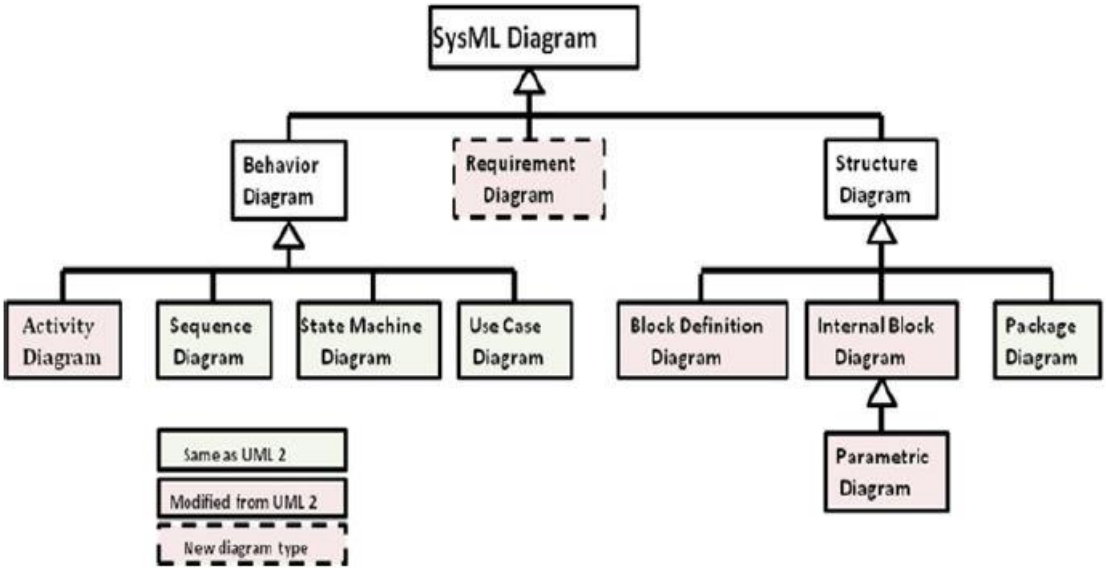


Figura 2–5. Jerarquía SysML y su relación con UML [10]

2.2 Ontología

Un enfoque distinto aunque complementario a lo citado anteriormente, se basa en el uso de **ontologías** cuya característica principal se basa en un aporte semántico superior, facilitando la **unificación de estructuras de almacenamiento de forma que puedan ser utilizadas por diferentes aplicaciones informáticas**.

Proviniente originalemte de la filosofía, su uso ha sido cada vez más común en el campo de la informática donde

[1] Se entiende por abstracción en programación orientada a objetos como el principio por el cual un programador "esconde" toda la información que no sea relevante acerca de un objeto de forma que reduce la complejidad e incrementa la eficiencia.

se han barajado multitud de definiciones hasta llegar a la que actualmente cuenta con mayor apoyo, establecida en Studer [11] y que la define como:

"Una ontología es una especificación formal y explicita de una conceptualización compartida"

Desglosando dicha definición es posible alcanzar un mayor entendimiento acerca de la misma:

- Conceptualización: Hace referencia a un modelo abstracto de algún fenómeno en el mundo a través de la identificación de los coneptos relevantes de dicho fenómeno.
- Explícita: El tipo de conceptos y relaciones se definen explícitamente.
- Formal: Entendible para las máquinas.
- Compartida: La ontología representa un conocimiento consensuado por un grupo, no por un individuo.

Como se ha mencionado, las ontologías aportan una semántica de un área concreta y definen, a distintos niveles, el significado de los términos y relaciones entre ellos. De esta forma, las ontologías están formadas por los siguientes componentes principales [12]:

- Clases: representan principalmente conceptos, éstos pueden ser algo sobre lo que se dice algo (tipo de objeto, descripción de tarea, estrategia, etc). Las clases de una ontología suelen organizarse en taxiomas. Si bien es cierto que a veces la ontología se diluye en el término taxonomía (ontología completa), no se deben confundir con las ontologías.
- Atributos: representación de la estructura interna de los conceptos. En función de su origen, pueden ser específicos o heredados. Los específicos son propios del concepto, mientras que los heredados se establecen por las relaciones taxonómicas en las que el concepto desempeña el papel de "hijo" o "especificación".
- Relaciones: representan las interacciones entre conceptos del dominio. Se define formalmente como un subconjunto de un producto de n conjuntos, es decir, R: C1 x C2 x C3 x…x Cn. Entre los tipos de relaciones existentes, se encuentran como relaciones binarias las taxonómicas ("es un") y las partonómicas ("parte de").
- Funciones: tipo especial de relaciones en las que el n-ésimo elemento es único para los n-1 elementos precedentes. Formalmente, se definen como: F: C1 x C2 x…x Cn-1 → Cn. Como ejemplo, la función "madre de".
- Axiomas: expresiones que siempre son ciertas. Se incluyen en una ontología para definir el significado de sus componentes, definir restricciones complejas sobre valores de atributos, argumentos de relaciones, etc.
- Instancias: ocurrencias en el mundo real de los conceptos. Se usan para representar elementos específicos.

La aplicación de estas ontologías en los sistemas informáticos vino de la mano del creador de la Web Semántica y presidente del consorcio W3C Berners-Lee, el cual pretendía que las máquinas entediesen el contenido que aparecía en la Web y pudiesen utilizarlo, es decir, facilitando el procesado en integración de la información. Como resultado, surge el Lenguaje de Ontologías Web (OWL), el cual está pensado para usarse cuando la información contenida en documentos vaya a ser procesada por aplicaciones, en vez de simplemente ser presentada a los clientes [13]. OWL aporta una mayor interoperabilidad al contenido web que RDF ya que provee vocabulario adicional junto con semántica formal. Asimismo, OWL solventa algunas de las limitaciones de RDF, por ejemplo, este último no da soporte a la representación de desuniones de clases, restricciones de cardinalidad o relaciones especiales, entre otros.

OWL se divide en tres sublenguajes en función del nivel de detalle que se quiera alcanzar, denominados OWL Lite, OWL DL y OWL Full [13]. OWL Lite esta destinado a aplicaciones que sólo requieran una clasificación jerárquica y restricciones simples (cardinalidad entre 0 o 1). OWL DL está enfocado a aplicaciones que requerían la mayor expresividad posible conservando completitud operacional y resolubilidad. Por último, OWL Full se utiliza cuando se quiere la máxima expresividad y libertad sintáctia de RDF sin garantías computacionales.

La representación más común de un OWL se realiza mediante un esquema RDF, el cual no es más que un fichero XML con una estructura específica, como sigue:

En primer lugar, de forma opcional aunque recomendada, aparece una declaración con la versión del XML y la codificación usada.

En segundo lugar, aparece el nodo de mayor nivel *rdf:RDF* que engloba la ontología incluyendo clases y propiedades de objeto y que cierra el fichero. En la declaración de este nodo, se encuentran los espacios usados en el documento. Los básicos son los siguientes:

Prefijo	Nombre de Espacio	Notas
owl	"http://www.w3.org/2002/07/owl"	El nombre de espacio de OWL en la sintaxis RDF/XML
xsd	"http://www.w3.org/2001/XMLSchema"	El nombre de espacio del esquema XML
rdf	"http://www.w3.org/2000/01/rdf-schema#"	El nombre de espacio del esquema RDF

Tabla 2-1. Nombres de espacios

Asimismo, dentro del nodo *rdf* y anterior a la declaración de clases, puede aparecer un nodo *owl:Ontology* que recoge información acerca de la versión, versiones anteriores, compatibilidad, anotaciones, entre otros.

A continuación, se muestra una figura con la estrutura a modo de ejemplo:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
     <owl:Ontology>
        <owl:versionInfo/>
     </owl:Ontology>
     <owl:ObjectProperty/>
     <owl:ObjectProperty/>
     <owl:Class/>
     <owl:Class/>
</rdf:RDF>
```

Figura 2–6. Estructura OWL en sintaxis RDF

Principalmente, un fichero OWL define clases y atributos. Las clases constan de atributos que pueden ser literales o que tomen valores básicos (real, enteros, cadena) o bien valores procedentes de otras clases definidas, en cuyo caso, se tratarían de propiedades de objeto.

Finalmente, existe la posibilidad de definir subclases. Una subclase es una copia de la clase padre a la que se pueden incluir más atributos.

2.2.1 Herramientas

Las herramientas de desarrollo de ontología ofrecen un ámbito para cargar, instanciar y consultar ontologías. A continuación, se presentan algunas.

2.2.1.1 Ontolingua Server

Ontolingua Server [13] fue la primera herramienta ontológica construida a mediados de los 90 por el Laboratorio de Sistemas de Conocimiento (KSL) de la Universidad de Stanford [15]. Ontolingua Server ofrece un ambiente colaborativo para navegar, crear, editar, modificar y usar ontologías. Farquhar [14] identifica tres modos de

interacción con esta herramienta relacionados con colaboradores remotos, aplicaciones remotas y aplicaciones independientes. En el primer caso, ayuda a los usuarios a navegar, crear y mantener ontologías mediante el uso de navegadores web y permite a los usuarios colaborar en una sesión compartida. En el segundo modo, las aplicaciones remotas dan soporte a usuarios para consulta y modificación de ontologías así como acceso a datos y metadatos. Por último, Ontolingua ayuda a los usuarion a traducir ontologías al formato que requieran.

2.2.1.2 Protégé

Protégé es una herramienta de software libre que ayuda al usuario a construir modelos de dominio y aplicaciones basadas en conocimietos usando ontologías. **Protégé es una herramienta independiente [15] y es ampliamente usada debido a su disponibilidad de ayuda online** [16]. Protégé ofrece tanto una interfaz como un formato de salida customizable que además puede ser adaptado con cualquier lenguaje formal [17]. Asimismo, Protégé facilita la integración con otras aplicaciones, herramientas, bases de datos y es capaz de dar soporte a lenguajes gráficos como OWL y RDFS [18].

El proceso de creación de una ontología con Protégé es sencillo e intuitivo, como se muestra a continuación:

Una vez en la ventana principal, bajo la pestaña "Entities" se encontrarán las distintas pestañas de clases, propiedades de objetos, propiedades de datos y anotaciones que existan en el modelo ontológico. Para crear una nueva clase, hay que siturarse sobre la pestaña correspondiente y, a partir de la clase raíz *owl:Thing*, basta con clicar en el icono  , el cual también permite crear subclases. Si la intención es crear clases al mismo nivel, basta con clicar sobre  .

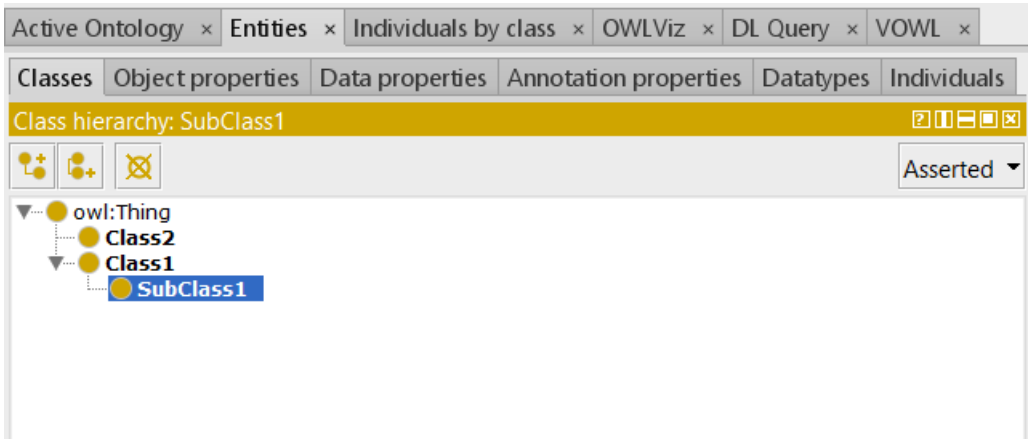


Figura 2–7. Pestaña de Clases

Asimismo, es posible establecer relaciones semánticas entre clases del tipo *Equivalent To*, *Subclass of,* etc., así como establecer atributos en la ventana de anotaciones.

Para relacionar clases, hay que dirigirse a la pestaña de "Object properties" y se deberá crear una nueva propiedad siguiendo un proceso análogo al de las clases. Situándose sobre el nodo raíz *owl:topObjectProperty* se podrán crear tantas como se quieran. Una vez creada, en la ventana de "Description" se deberá establecer la información de dicha propiedad, por ejemplo, señalando su dominio y su rango. Para ello, aparecerá una ventana en la que será posible seleccionar las clases previamente creadas.

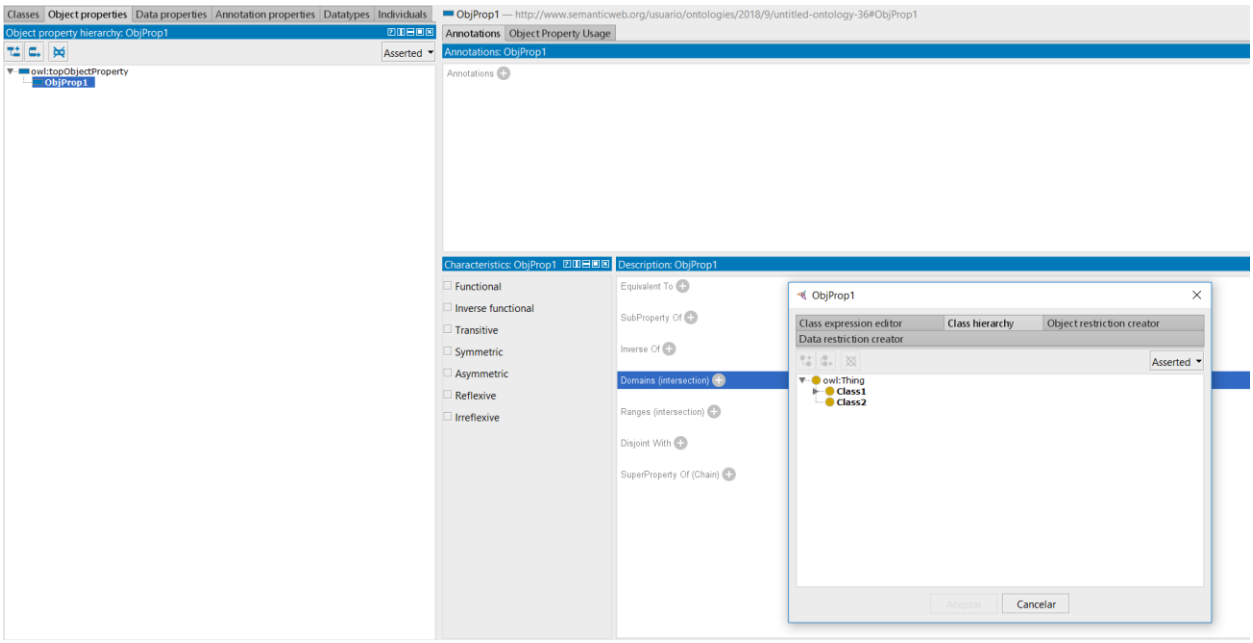


Figura 2–8. Creación Propiedades de Objeto

Una vez finalizada la ontología, basta con clicar sobre “File>Saves as” y elegir el formato que se desee así como la ubicación del destino.

2.2.1.3 Enterprise Architect

Enterprise Architect es una herramienta de modelado basada en OMG UML, siendo capaz de albergar modelados del ciclo de vida completo de cualquier proyecto gracias al alto nivel de seguimiento de las especificaciones de análisis, diseño, implementación, test y manutención de modelos mediante el uso de UML, SysML, BPMN y otros estándares [19]. Además, favorece la interacción entre las distintas partes de proyectos compartidos gracias a sus servidores “nube”.

La herramienta incluye la especificación de modelos MBSE mediante SysML y dispone de capacidad para definir ontologías y exportarlas en formato OWL.

Para la creación de una ontología en EA, el proceso es el siguiente:

En primer lugar, tras ejecutar el programa, se selecciona sobre “New File” en la pantalla principal.

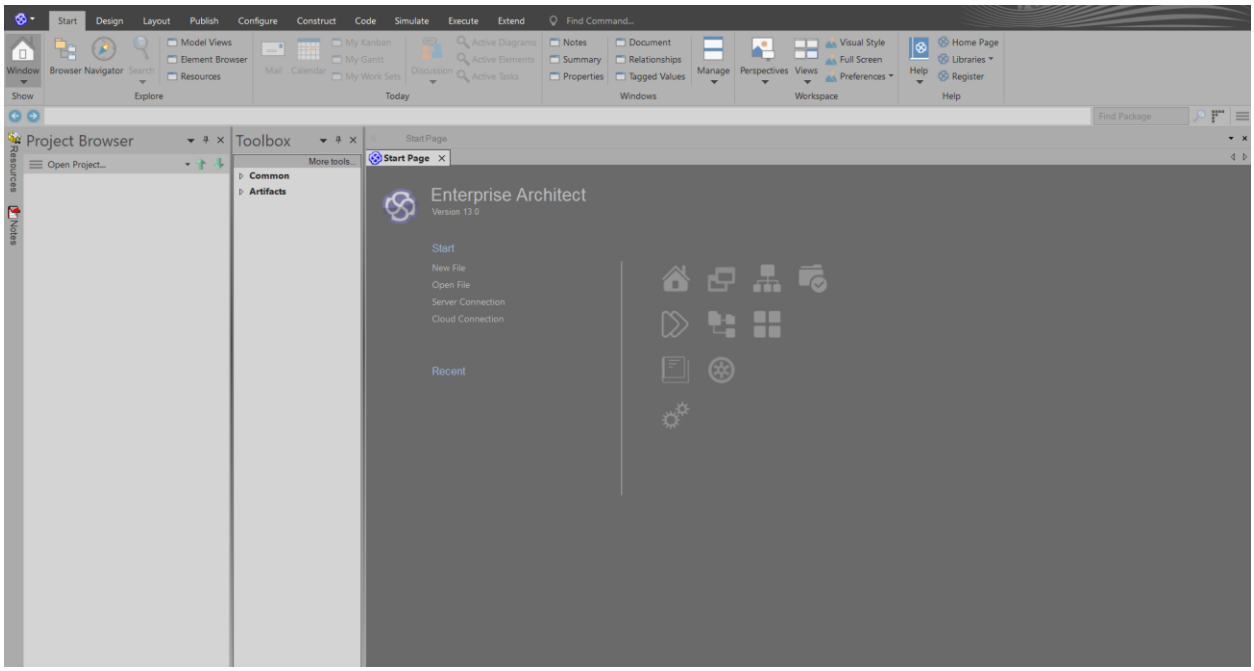


Figura 2–9. Pantalla principal Enterprise Architect

Tras ello, aparecerá una ventana donde se podrá elegir la ubicación y el nombre del archivo que se va a crear. Una vez establecidos, será el turno de elegir en qué modulo de EA se va a trabajar. Para ello, en este caso, basta con clicar en la casilla "OWL Ontology" bajo la sección "ODM" (Figura 2–10).

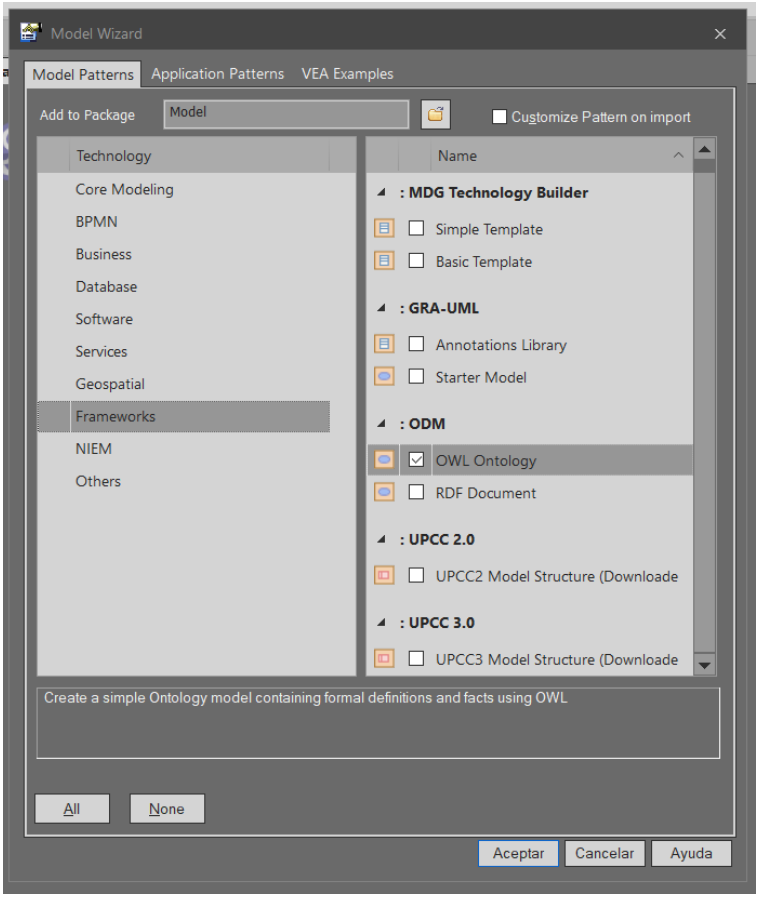


Figura 2–10. Selección módulo

Tras aceptar dicha selección, la pantalla principal anterior se actualizará al módulo seleccionado, como se muestra en la siguiente figura:

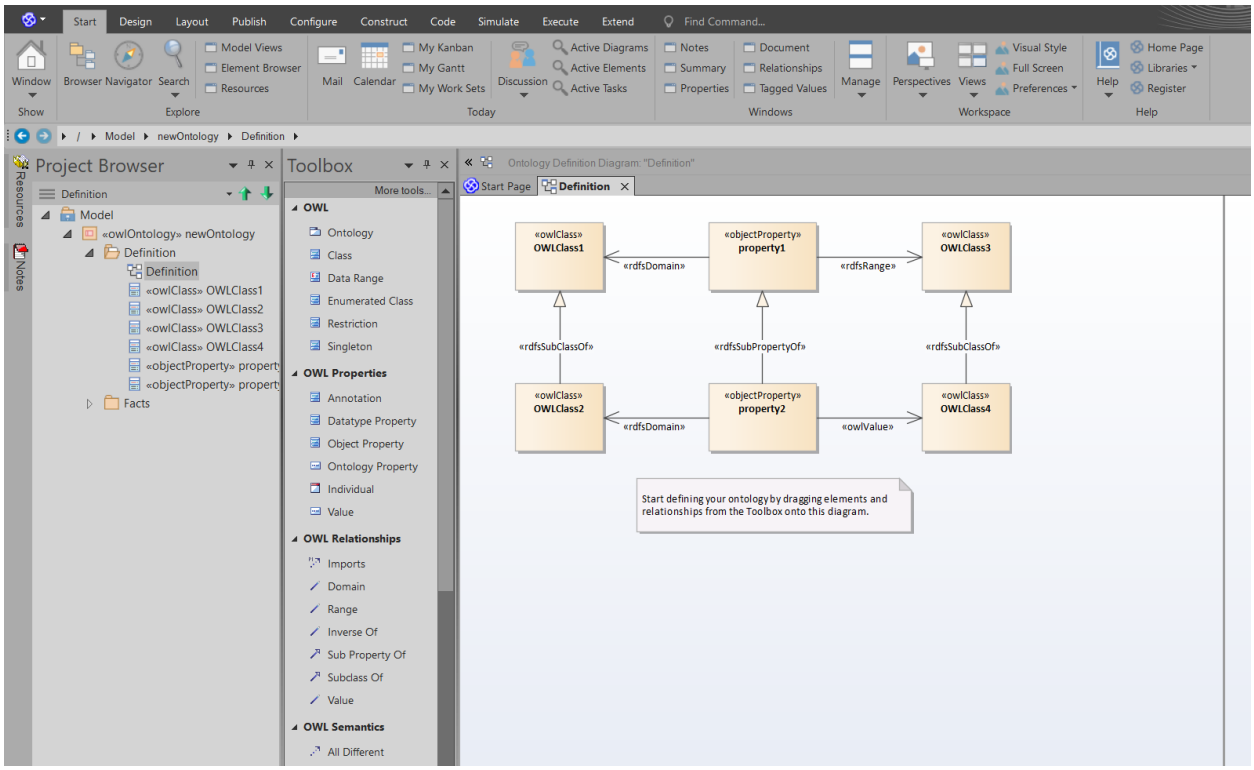


Figura 2–11. Módulo OWL

Es habitual que aparezca un pequeño diagrama a modo de ejemplo que podrá ser borrado sin problema alguno. Bajo la sección “Toolbox” se encuentran todas las herramientas de las que dispone este módulo en concreto, si bien se pueden acceder al resto clicando sobre “More tools”. Para usar cualquiera de ellas, basta con clicar sobre la que se desee y volver a clicar sobre la ventana de definición. De esta forma, podrán crearse clases, que representen grupos de individuos; propiedades que pueden ser de objeto o de tipo de dato, mediante las cuales se representan relaciones haciendo uso del triplete *Sujeto-Predicado-Objeto*. El sujeto y el objeto son definidos mediante clases y el predicado se trata de una propiedad.

Figura 2–12. Ejemplo triplete

Otra herramienta disponible es el uso de semántica, la cual establece relaciones entre clases:

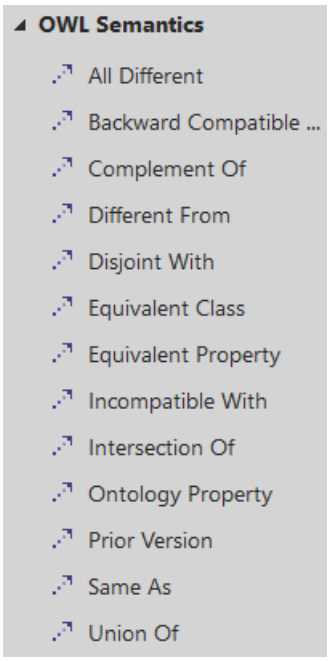


Figura 2–13. Semántica

Una vez finalizado, para exportar el archivo basta con clicar con el botón derecho sobre <<*owlOntology*>> en el árbol de definición y seleccionar "Extensions>ODM>Export OWL/RDF".

2.2.2 OntoSTEP

Hasta ahora se han comentado las características de las ontologías, su lenguaje gráfico y herramientas de modelado que soportan dicho lenguaje, si bien con todo esto es posible el desarrollo de un modelo propio enriquecido semánticamente más allá de las posibilidades de las técnicas de modelado mencionadas con anterioridad en 2.1.2. Técnicas de modelado, existen determinados estándares a partir de los cuales es posible su adaptación mediante ontologías con el objetivo de mejorarlos, este es el caso de OntoSTEP.

STEP (Standard for Exchange of Product model data) [20], como su nombre indica, facilita el intercambio de datos del producto entre software CAX y surge como consecuencia del auge de los sistemas PLM, los cuales son capaces de modelar, capturar, manipular, intercambiar y usar información a lo largo de todo el proceso de decisión del ciclo de vida del producto en todo el dominio de aplicación. En los últimos años, se ha desarrollado otro estándar STEP-NC [21] que mejora la interoperabilidad de sistemas a través de la integración de la información de la maquinaria que realiza el proceso (CNC).

Los Protocolos de Aplicación (AP) de STEP son modelados mediante el lenguaje EXPRESS [22]. Este lenguaje fue desarrollado para representar modelos de productos y ser capaz de describir la información requerida para el diseño, fabricación y mantenimiento de los productos. El mayor incoveniente de STEP y su modelado mediante EXPRESS es el formato y la asuencia de un formalismo lógico que facilita la incorporación de una semántica formal. Barbau [23] propone un enfoque que permite la traducción de STEP definida con EXPRESS a OWL, lo que se denomina OntoSTEP.

2.3 Semántica

Comúnmente se entiende por semántica la ciencia que estudia el significado de palabras y expresiones, clasificando palabras de similiar y distinto significado. De tal forma que es posible interpretar diferentes modelos

siempre y cuando se disponga de las equivalencias de palabras correspondientes.

De este modo, se puede entender la aparición de herramientas semánticas como las mencionadas en los apartados 2.2.1.2 y 2.2.1.3, donde ambos softwares son capaces de expresar en un modelo ontológico mediante el uso de funciones como *Equivalent to* o *Same as* que dos objetos son el mismo.

No obstante, cuando se habla de semántica como un tipo de modelo, también hacen referencia a aquellos que permiten obtener información de un sistema haciendo preguntas "sencillas", como por ejemplo, la temperatura de un determinado sensor, sin tener que entender detalles como qué sistema de control monitoriza ese sensor o qué sensor es el que controla esa temperatura. Por tanto, los modelos semánticos son muy útiles a la hora de relacionar el mundo "físico" con el real.

La forma de proveer acceso a esta información debe ser consistente, para ello, en la implementación de un modelo semántico se usan tripletes, que no son más que una relación *Sujeto-Predicado-Objeto*, siguiendo con el ejemplo anterior:

- Tanque1 <tiene temperatura> Sensor 1
- Tanque 1 <es parte de> Paltaforma 1
- Plataforma 1<es parte de> Región 1

Estos tripletes conforman una ontología para la región 1, por tanto, puede decirse que un modelo semántico está incluido en la definición de la ontología.

Otra ventaja de los modelos semánticos se trata del mantenimiento del modelo en sí. Un modelo puede implementarse siguiendo distintos diagramas:

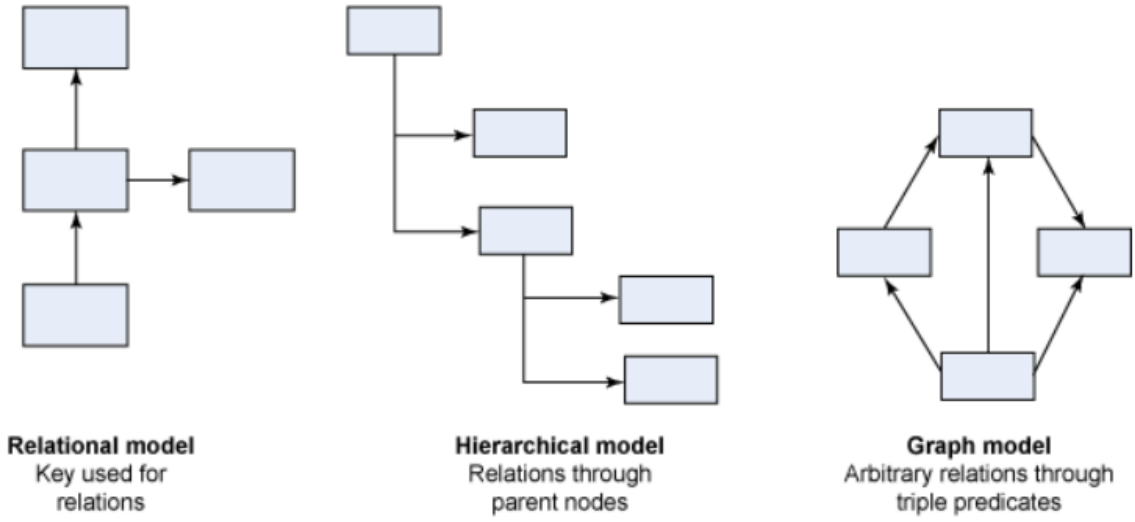


Figura 2–14. Tipos de diagrama [24]

Los modelos relacionales establecen relaciones mediante las claves (primarias y secundarias), como ventaja es que son modelos sencillos de entender; sin embargo, modificar alguna relación requiere modificar la estructura base del propio modelo, lo que puede no ser trivial en situaciones con una base de datos extensa. Por su parte, los modelos jerárquicos sufren limitaciones similares cuando son necesarios realizar cambios y son poco flexibles a la hora de procesar el modelo "horizontalmente".

Por último, el modelo en grafo (usado en modelos semánticos) facilita tanto las consultas como el mantenimiento del propio modelo, ya que si fuese necesario añadir una nueva relación, sería sencillo hacerlo mediante la adición de un triplete a la representación. Asimismo, es posible abordar el grafo desde distintas perspectivas permitiendo la aparición de preguntas no planteadas en el diseño inicial.

Por este motivo, el esquema RDF no es más que un marco de trabajo para la representación de grafos, basados en tripletes de la forma *Sujeto-Predicado-Objeto*, de tal forma que es posible describir taxonomía de clases, formalizar relaciones entre las mismas para, finalmente, terminar definiendo ontologías.

3 DISEÑO METODOLÓGICO

En este capítulo se describe el diseño metodológico seguido en el proyecto para el desarrollo de la herramienta. Citando el estado en el que se encontraba en el inicio y detallando los cambios introducidos.

3.1 Introducción

En primer lugar, destacar que el componente principal de este desarrollo se trata de ARAS innovator, un software PLM basado en una arquitectura SOA (Service-Oriented Arquitecture). En el caso que se está abordando, la intención es usar ARAS como un gestor de modelos, el cual sea capaz de leer ontologías y almacenarlas correctamente en la base de datos, ser capaz de modificar dichas ontologías si fuese necesario y volver a generar la ontología modificada. Una vez introducida la ontología final en ARAS, se realizarán una serie de instancias para comprobar su funcionamiento (Figura 3–1).

Tanto la importación como exportación de ontologías se llevaría a cabo integrando un código C# en el propio ARAS que permita ambas acciones, en el Capítulo 4 se desarrollarán las funciones más significativas así como el modelo de datos seguido en el código, que nos permitirán importar ontologías en ARAS.

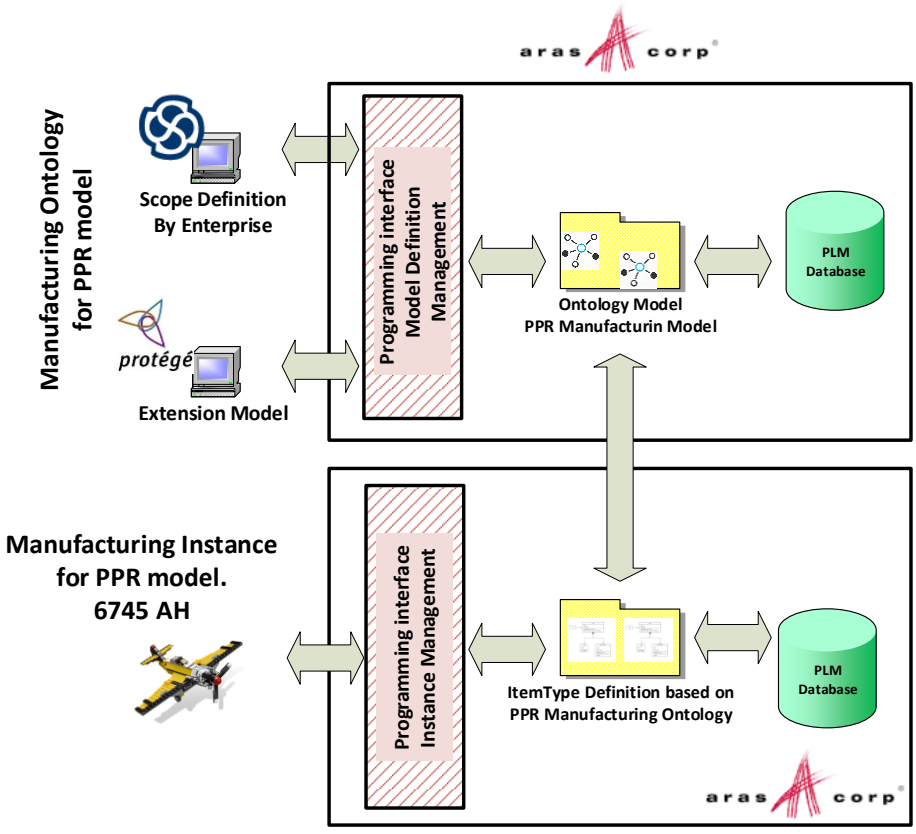


Figura 3–1. Proceso a seguir

En el estado actual, el código implementado en ARAS es capaz de procesar ontologías básicas generadas por las herramientas IDEF y CMAP. No obstante, y como consecuencia de la popularidad de otra herramienta denominada Enterprise Architect (EA), se opta adaptar el gestor inicial de modelos para que sea posible importarlos desde EA.

3.2 ARAS

ARAS en un software open source con las siguientes funcionalidades de PLM:

- Customizable. Capacidad para adaptarse y ser configurado en base a las necesidades del usuario, permitiendo gestionar el conocimiento.
- Escalable. Capacidad para reutilizar información con el fin de mejorar o ampliar futuros modelos.
- Interoperabilidad. Capacidad de los sistemas PLM para comunicarse e intercambiar información con otros sistemas. De modo que, permite la utilización de herramientas externas para el desarrollo de los modelos.
- Gestión del ciclo de vida. El ciclo de vida dicta en qué fase se encuentra un objeto así como los usuarios que pueden acceder y qué acciones pueden ejercer sobre dicho objeto.
- Colaboración. Un sistema PLM favorece un entorno colaborativo en el que compartir el conocimiento de las distintas partes involucradas en un proyecto.

La interfaz de ARAS se muestra en la siguiente Figura 3–2. Interfaz ARASfigura:

Figura 3–2. Interfaz ARAS

En la parte izquierda se muestra la Tabla de Contenidos (TOC), donde se encuentran las funcionalidades de ARAS, más concretamente en la pestaña *Administration*. Bajo ella, las más relevantes en este caso son:

- ItemTypes. Todo objeto (incluyendo relaciones entre los mismos) en ARAS es un ItemType y se encuentra recogido en esta sección. Cada ItemType tiene una serie de atributos que pueden ser modificados; así como unas restricciones de quién puede ver, crear, modificar o eliminar este tipo de objeto. Destacar que en esta sección, se encuentra el objeto "base". Por ejemplo, si creamos un ItemType "Modelo" con sus atributos y queremos crear varios modelos, debemos configurar para que dicho ItemType aparezca como una sección en la Tabla de Contenidos y allí crear cuantos modelos se quiera.
- Forms. Sección donde se encuentran los formularios de cada ItemType, se entiende por formulario la interfaz que aparece al crear un objeto de un tipo de ItemType. Cada formulario se crea automáticamente al crear un ItemType.
- Method. En esta sección se recogen los códigos disponibles en ARAS.

3.2.1 Customización ARAS

Como se ha comentado, todo objeto en ARAS es un ItemType. Para crear uno, hay que dirigirse a la sección de la Tabla de Contenidos (TOC) denominada *ItemTypes*, al clicar sobre el botón "Create new Item"  aparecerá una ventana como la que sigue:

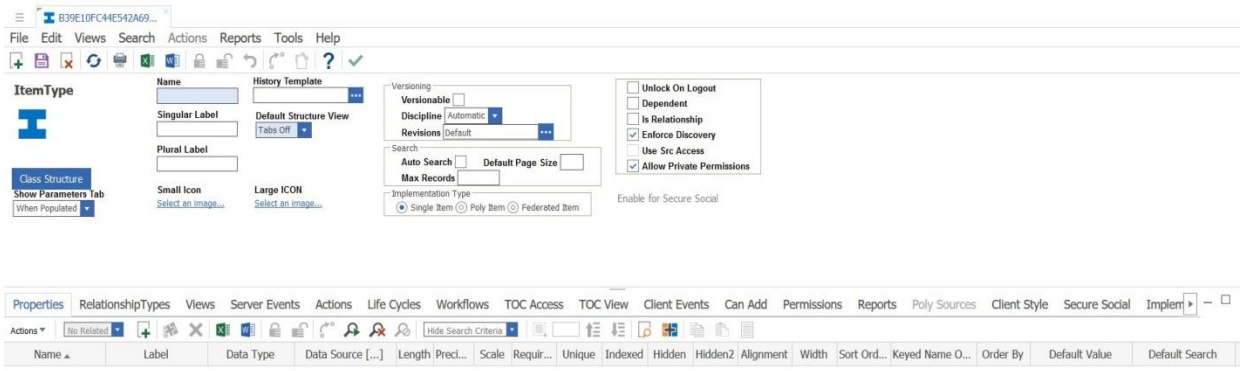


Figura 3–3. Creación ItemType

Es posible crear un nuevo ItemType desde cero o como copia de otro existente, haciendo uso de la capacidad de reutilizar información ya almacenada. Esto es posible debido a que ARAS posee por defecto item relacionados con la fabricación como *Part* o *Product*.

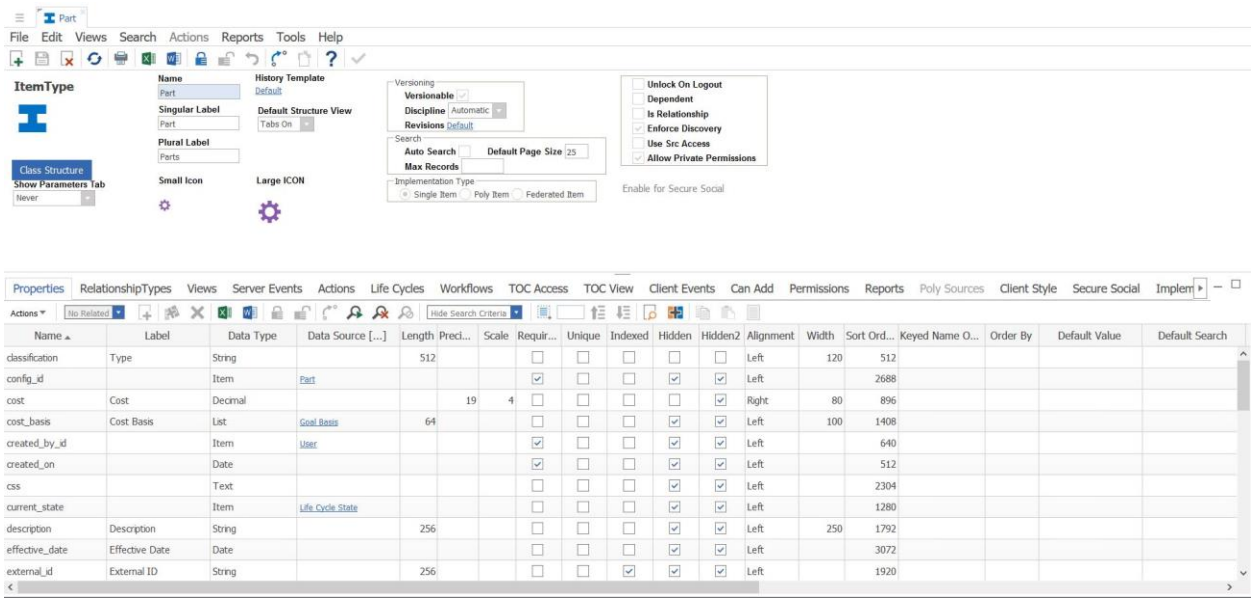


Figura 3–4. ItemType *Part*

En la mitad superior de la ventana, para cada ItemType, las etiquetas "Name", "Singular Label" y "Plural Label" deben ser rellenadas. Asimismo es posible modificar el icono que representa dicho objeto, así como clasificarlo como versionable o no. En la mitad inferior, existen varias pestañas relacionadas con el ItemType, entre ellas se encuentran las siguientes:

- Properties. En esta pestaña se encuentran los atributos asociados al item. Asociadas a cada atributo existen una serie de propiedades como su etiqueta ("Label"), el tipo de dato, la fuente y si es requisito.

- Relationships. Las relaciones entre ítems aparecen en esta pestaña. Al crear una nueva relación bajo esta pestaña, ARAS también crea un nuevo ItemType con la casilla “Is Relationship” marcada en el menú principal de Itemtypes, para diferencias aquellos ítems que son o no relaciones.

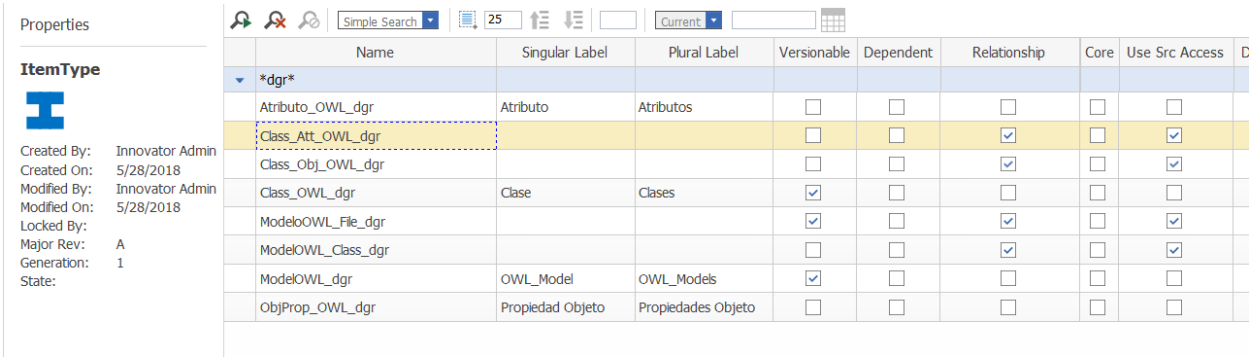


Figura 3–5. Menú principal ItemType

- Lyfe Cycles. El ciclo de vida del item aparecerá en esta pestaña.
- TOC Access. La información en esta pestaña describe dónde aparecerá el nuevo item en la Tabla de Contenidos y qué usuarios tendrán acceso a ellas. Para crear secciones en la Tabla de Contenidos, es neceario modificar la lista *Category* que se encuentra en la sección *List* de la TOC.
- Can add. Bajo esta pestaña se ubican los permisos para añadir o crear el item de este tipo en su sección de la TOC.

Una vez finalizado la configuración del ItemType, se guarda clicando sobre el botón “Save, Unlock & Close”  . Tras ello, este nuevo tipo de item aparecerá en la Tabla de Contenidos bajo la sección que se haya dispuesto en TOC Access. Como ejemplo, cuando se cree un nuevo item *Part*, una venta como la siguiente se mostrará:

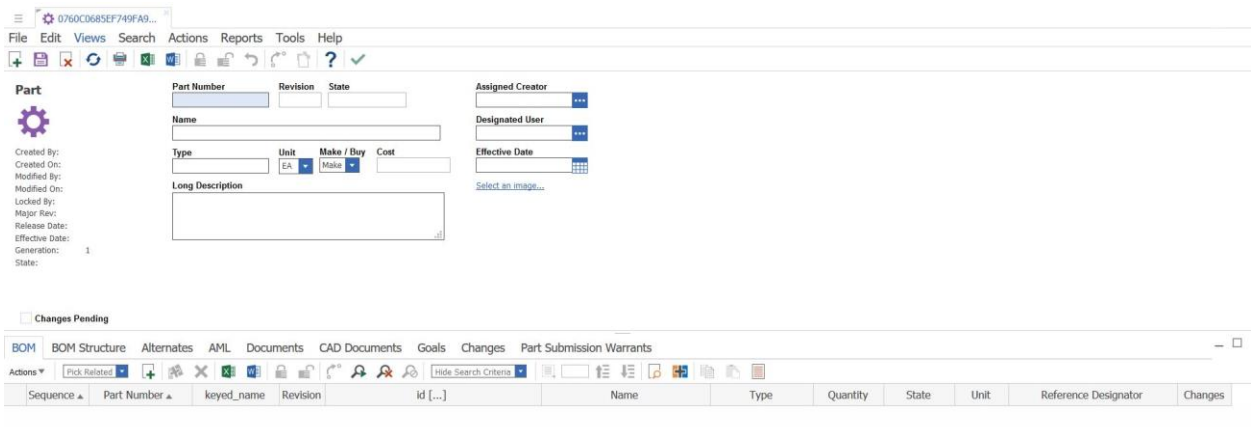


Figura 3–6. Ejemplo nueva parte

Esta ventana posee una estructura similar a la de ItemType: en la mitad superior se encuentra los datos identificativos del objeto, mientras que la mitad inferior recoge información acerca de las relaciones con otros ItemTypes. Asociado a estas relaciones, ARAS hace posible que se establezca cardinalidad entre las entidades relacionadas, para ello es necesario dirigirse a la sección de la Tabla de Contenidos *Relationship Types*, donde aparecen todas las relaciones creadas, clicando sobre una de ellas nos aparecerá una ventana como la que sigue:

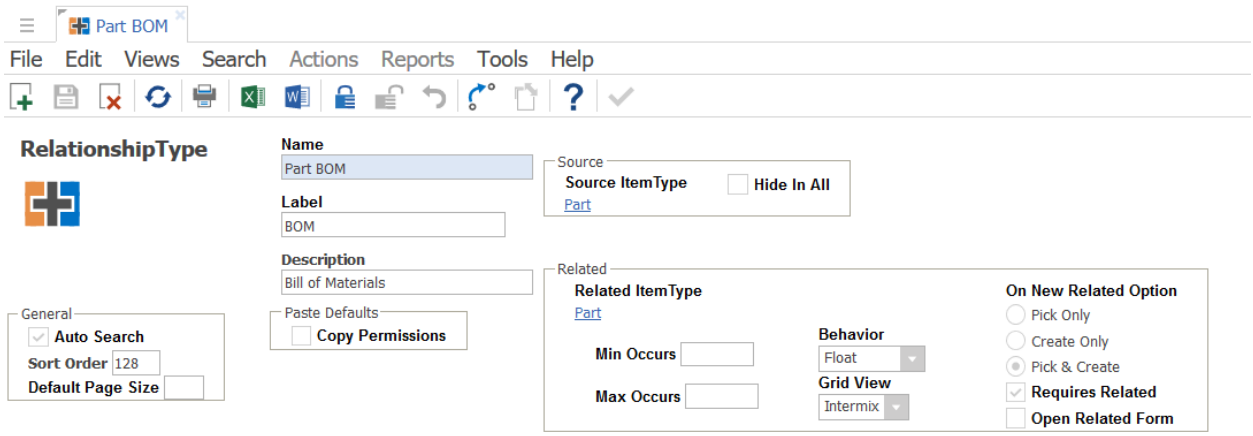
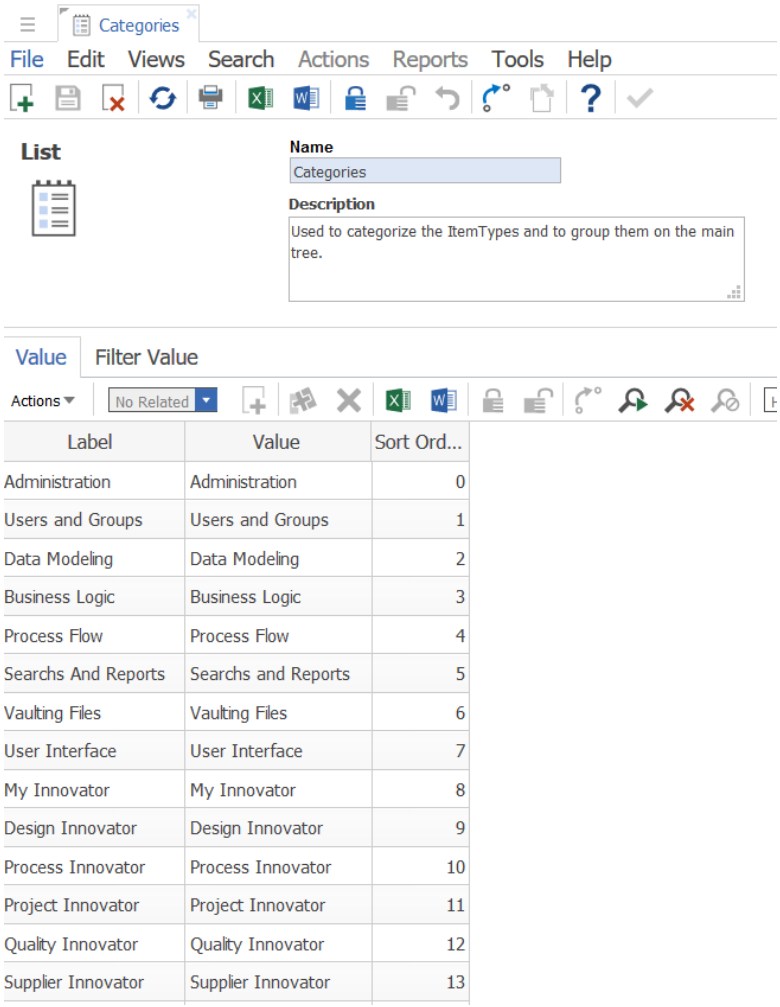


Figura 3–7. Relaciones de *Part*

La cardinalidad se stablece en el item relacionado en las casillas de *Min Occurs y Max Occurs*, pudiendo tomar números enteros. En el caso de dejarla en blanco, ARAS lo interpreta como ilimitado.

En último lugar, para modificar la Tabla de Contenidos añadiendo secciones, hay que dirigirse a la sección *Lists* y buscar la lista *Categories*:



Label	Value	Sort Ord...
Administration	Administration	0
Users and Groups	Users and Groups	1
Data Modeling	Data Modeling	2
Business Logic	Business Logic	3
Process Flow	Process Flow	4
Searchs And Reports	Searchs and Reports	5
Vaulting Files	Vaulting Files	6
User Interface	User Interface	7
My Innovator	My Innovator	8
Design Innovator	Design Innovator	9
Process Innovator	Process Innovator	10
Project Innovator	Project Innovator	11
Quality Innovator	Quality Innovator	12
Supplier Innovator	Supplier Innovator	13

Figura 3–8. Lista *Categories*

En la cual se pueden añadir nuevas secciones, incluyendo su nombrel (*Label)*, si se quiere que aparezca como subsección de otra o no (*Value)* y la posición (*Sort Order*) en la Tabla de Contenidos, siendo 0 el primer lugar.

3.3 Diseño

El procedimiento a seguir para el almacenamiento completo de la ontología es el siguiente (Figura 3–11).

En primer lugar, con ayuda de Enterprise Architect se genera un primer esquema de la ontología que se va a desarrollar, definiendo sus clases y propiedades de objetos, que no son más que las relaciones entre clases. Si bien es cierto que a las clases es posible añadirle propiedades o atributos, a la hora de exportar dicho modelo en formato XML, dichos atributos no aparecen representados. Más adelante se propone una solución para complementar esta limitación.

Una vez se obtiene el archivo XML con la ontología, el siguiente paso es almacenarla esta primera versión en nuestra base de datos ARAS. En la cual se habrán creado los ItemTypes correspondientes a:

- Modelo. Donde se almacenará cada modelo importado.
- Clases. Donde se almacenarán las clases de cada modelo.
- Propiedades de Objetos. Donde se almacenarán las propiedades de objetos de cada modelo, entendiéndolas como las relaciones entre clases.
- Atributos. Donde se almacenarán los atributos de cada clase.

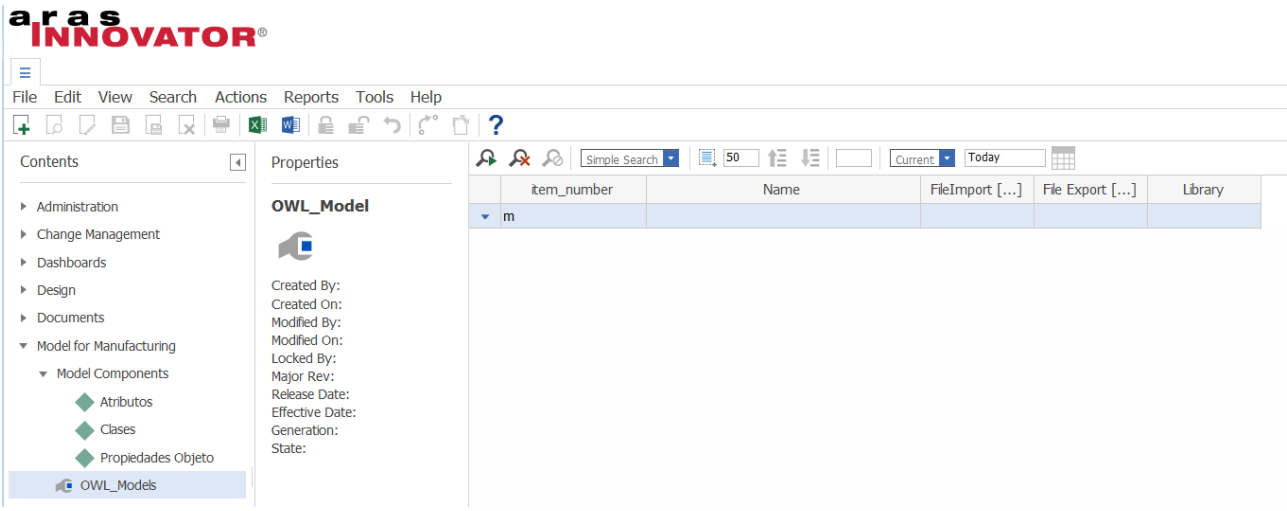


Figura 3–9. ItemTypes asociados a la ontología

Los cuales se relacionan según se muestra en el siguiente modelo de datos:

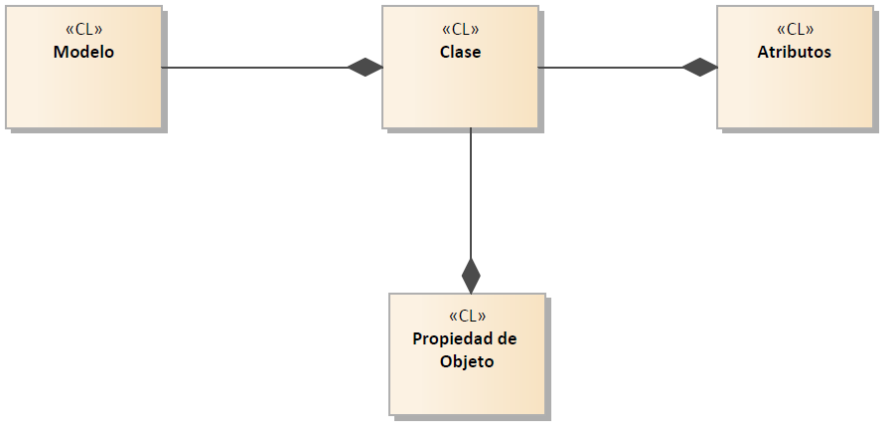


Figura 3–10. Modelo de Datos

Este modelo muestra la relación entre los ItemTypes creados anteriormente, se observa cómo el modelo está compuesto de clases, las cuales a su vez poseen una serie de atributos y propiedades de objeto. Es importante recalcar que la clase *Atributos* contendrá los atributos de las clases que se definan en la ontología; por su parte, cada uno de estos ItemTypes poseerá atributos propios asociados a ARAS, entre ellos:

- Id: Almacena el identificativo unívoco del ItemType.
- Name: Almacena el nombre del ItemType.
- Revision: Almacena el número de revisiones llevadas a cabo.
- State: Gestiona el estado en el que se encuentra el ItemType.

Una vez importada esta primera versión, se procederá de nuevo a la generación de un fichero XML con esta ontología. El cual será modificado con la herramienta de Protégé, mediante la que se añadirán los distintos atributos a las clases previamente creadas.

El motivo de realizar el paso intermedio de la importación en ARAS en lugar de utilizar el archivo XML generado por Enterprise Architect, se debe a que Protégé no es capaz de procesar correctamente este último debido a que EA incluye en los nodos de cada clase el atributo *rdf:ID=""*.

Por último, una vez añadidos todos los atributos a cada clase, se volvería a procesar y almacenar la ontología en ARAS.

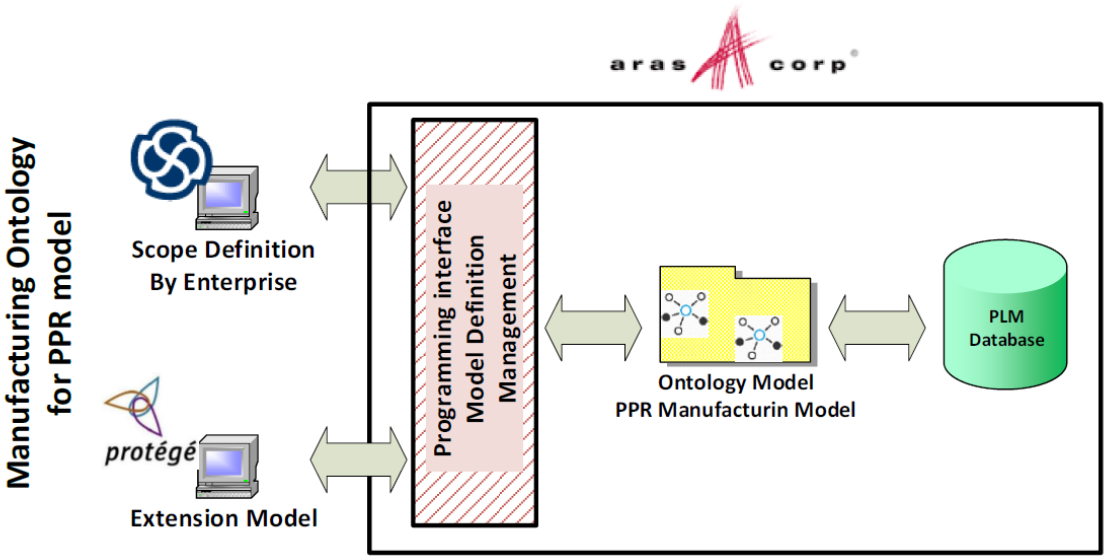


Figura 3–11. Arquitectura

3.4 Revisiones

Como cualquier proyecto que se precie, conforme se avance en el mismo se encontrarán aspectos redundantes y mejorables, lo que podrá llevar a la realización de posibles revisones de la ontología que defina el proceso de fabricación, gracias al uso de un software PLM. ARAS permite la definición de un ciclo de vida que controle el estado en el que se encuentran sus ItemTypes. En este caso, dicho ciclo se ciñe al establecido por defecto de ARAS:

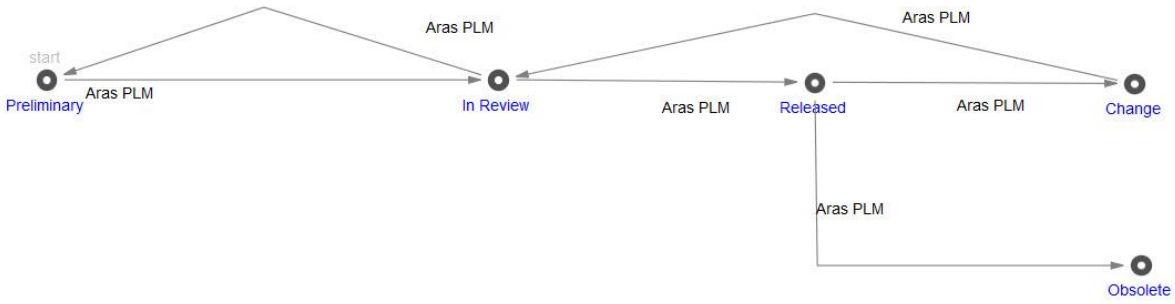


Figura 3–12. Ciclo de vida ItemType

Según el cual, todo ItemType se rige por él. El primer estado es el preliminar, tras el cual el objeto pasaría a estar en revisión, con el fin de comprobar su correcto funcionamiento. Tras su validación, pasaría a estar en un estado liberado, en el cual ya podría ser utilizado por el usuario. En el caso que fuese necesario realizar algún cambio, pasaría al correspondiente estado. Una vez realizadas las modificaciones, se trasladaría al estado de revisión de nuevo. El último estado se trata del obsoleto, en el caso de que el item llegara al final de su vida útil o exista una mejor versión del mismo. Cabe destacar que en cada estado existirán unos permisos de tal forma que determinados usuarios podrán ejercer determinadas acciones sobre el objeto. En este caso, se observa cómo el paso de un estado a otro es responsabilidad del usuario *Aras PLM*.

Con estas revisiones, se controlan tanto las mejoras de los ItemTypes del PPR como de la propia ontología.

Sin embargo, uno de los principales objetivos no es la realización de revisiones, sino de estudiar cómo con un diseño dado y una ontología previamente definida, afectan al desarrollo de un proyecto, es decir, se mencionarán dichos aspectos mejorables, pero no existirá una revisión del diseño o de la ontología como tal con su posterior estudio.

4 IMPLEMENTACIÓN

En este capítulo se detalla la implementación del diseño explicado en el capítulo anterior. Como se mencionó, se trabajará sobre ARAS, por tanto, a continuación se mostrarán tanto el modelo de datos desarrollado para llevar a cabo el procesamiento de los modelos ontológicos así como los casos de uso de las funciones principales que ayudarán a entender el funcionamiento de la herramienta.

4.1 Modelo de datos

El modelo de datos se engloba dentro del subsistema ARAS (Figura 4–1), que contiene todas las clases que intervienen en el código desarrollado, dicho esquema se muestra en Figura 4–2, donde se recogen todas las clases que intervienen.

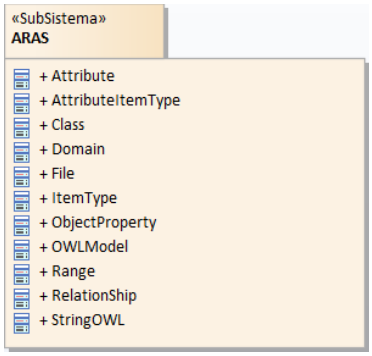


Figura 4–1. Subsistema ARAS

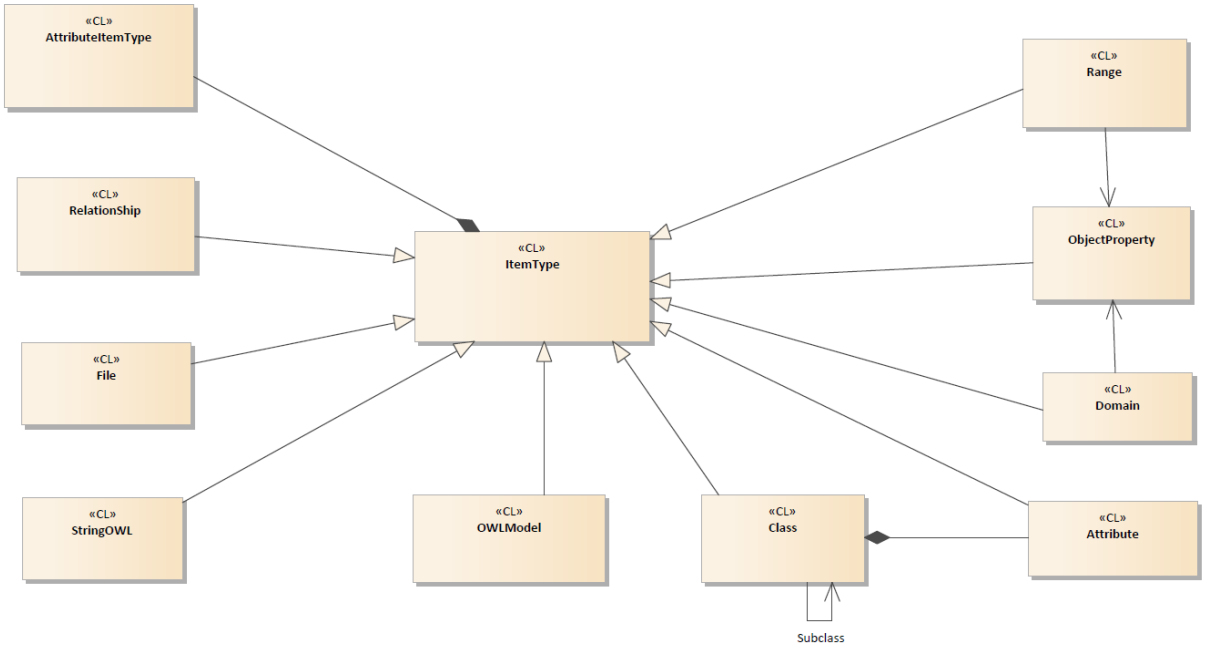


Figura 4–2. Modelo de datos

A continuación, se explicará de forma individualizada la función de cada una.

- ItemType. Clase principal del modelo. Como se mencionó en 3.2 ARAS, todo objeto es un ItemType, esta clase recoge las funcionalidades para incluir, modificar o eliminar items de la base de datos de ARAS. Por ello, el resto de clases se crearán de tal forma que hereden estas funciones.
- AttributeItemType. Clase que complementa a ItemType, puesto que todo objeto posee una serie de atributos, esta clase desempeña dicha función, define atributos y establece sus valores.
- RelationShip. Clase destinada a la defínción, modificación y/o elminación de relaciones entre ItemTypes.
- File. Clase asociada al procesamiento de archivos externos en caso de que se quieran subir a la base de datos.
- StringOWL. Clase destinada a separar cadenas de caracteres.
- Attribute. Clase destinada al procesamiento de los atributos de las clases del modelo.
- Range. Clase destinada a la inicialización de una variable tipo rango, asociada a las propiedades de objeto.
- Domain. Clase destinada a la inicialización de una variable tipo dominio, asociada a las propiedades de objeto.
- ObjectProperty. Clase destinada principalmente al procesado de los nodos "ObjectProperty" del XML, tanto su importación como exportación a/desde ARAS.
- Class. Clase destinada principalmente al procesado de los nodos "Class" del XML, tanto su importación como exportación a/desde ARAS.
- OWLModel. Clase destinada principalmente al procesado del modelo contenido en el XML, englobando el procesamiento de clases, atributos y propiedades de objeto; tanto su importación como exportación a/desde ARAS.

4.2 Casos de Uso

En este apartado, se exponen los casos de uso principales que son:

- Importación modelo XML. Se muestra el diagrama de actividades asociado a la importación del modelo contenido en el fichero. Se comienza con la lectura del mismo, su almacenamiento en memoria y, por último, la transferencia de la información a la base de datos.

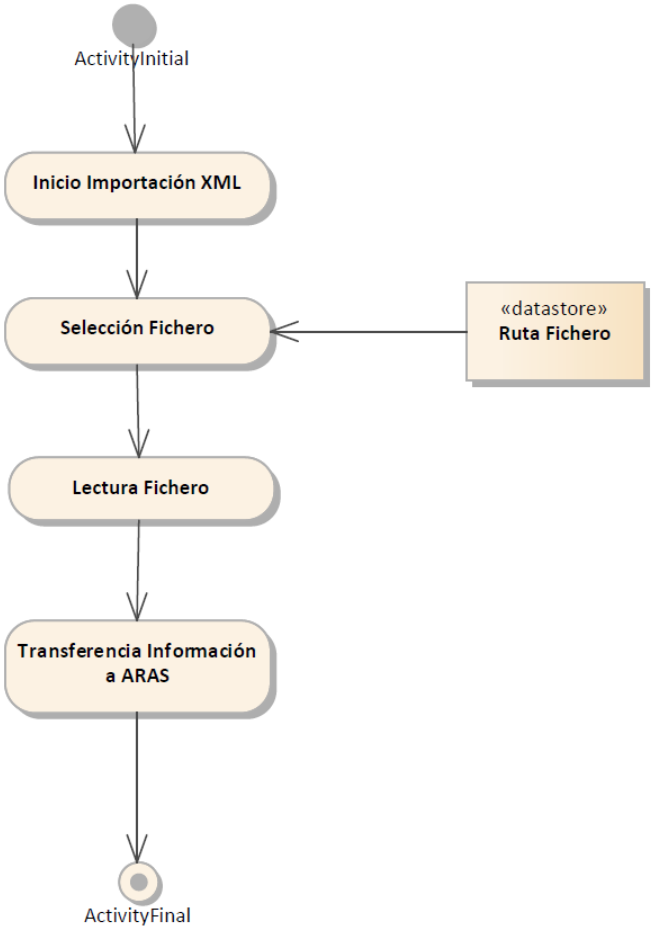


Figura 4–3. Importación XML

Para una mejor compresión del proceso, se representa seguidamente la función de lectura por separado. Ésta consiste en el procesado de cada nodo según el tipo que sea, puesto que de ello depende la información que almacena. Los nodos pueden ser tipo *Class*, *ObjectProperty* y otros, siendo estos últimos no relevantes en cuanto a la información que aportan.

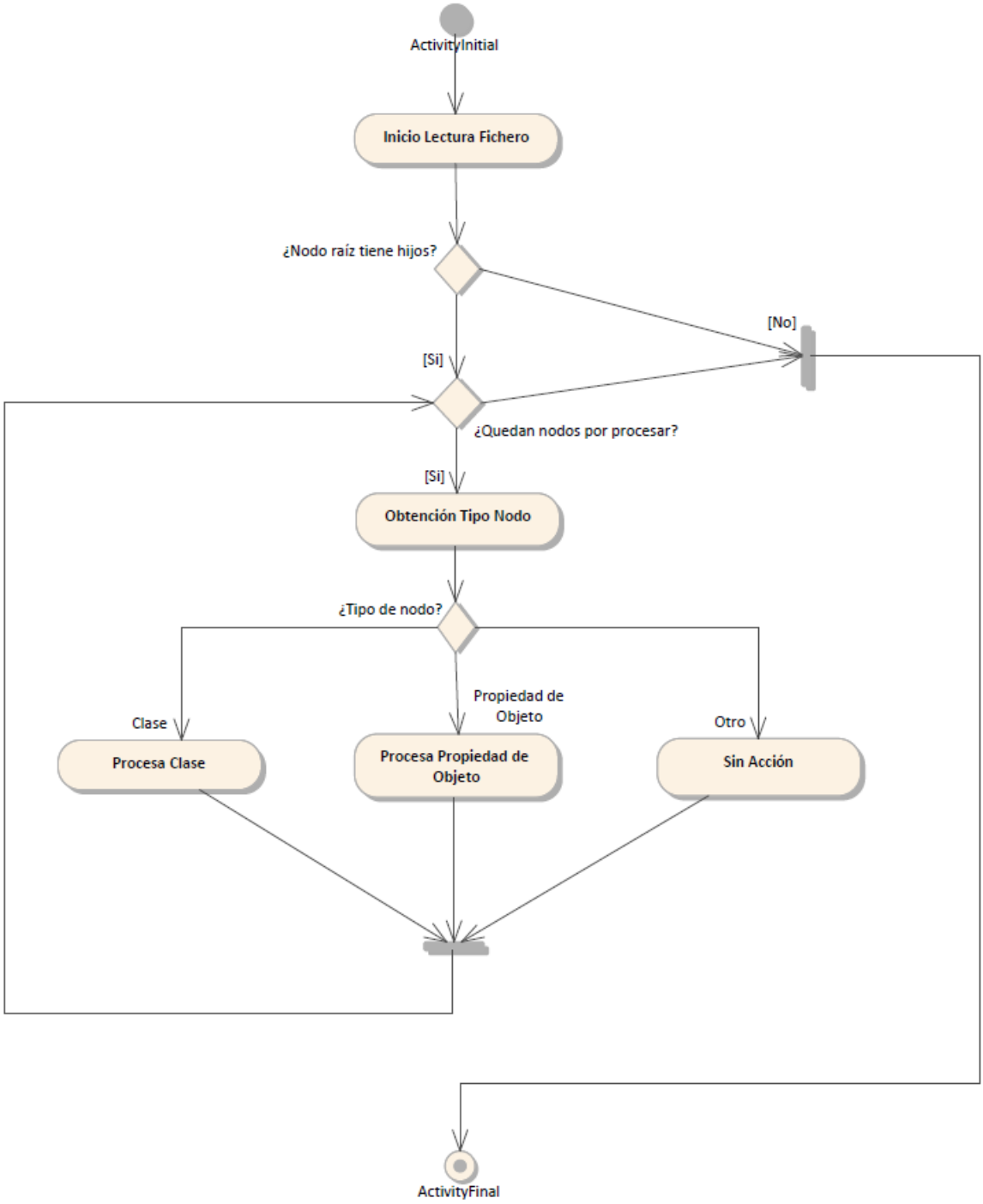


Figura 4–4. Lectura XML

- Exportación modelo XML. Se muestra el diagrama de actividades asociado a la exportación del modelo contenido en el fichero. Se comienza con la carga del modelo correspondiente, accediendo así a sus clases y propiedades de objeto, las cuales se plasman en el archivo XML.

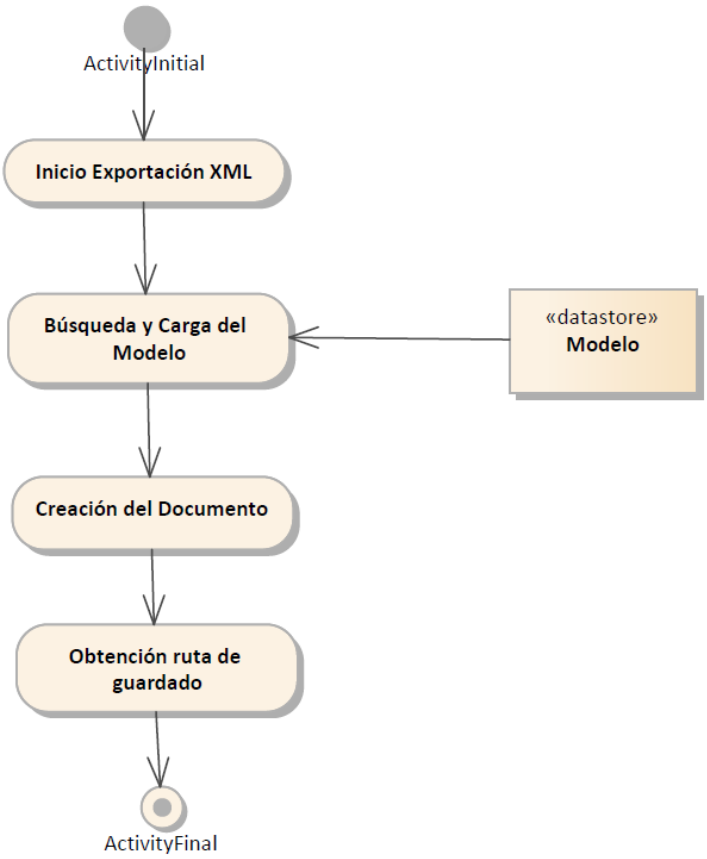


Figura 4–5. Exportación XML

El diagrama de actividades de la acción “Creación del Documento” es el siguiente:

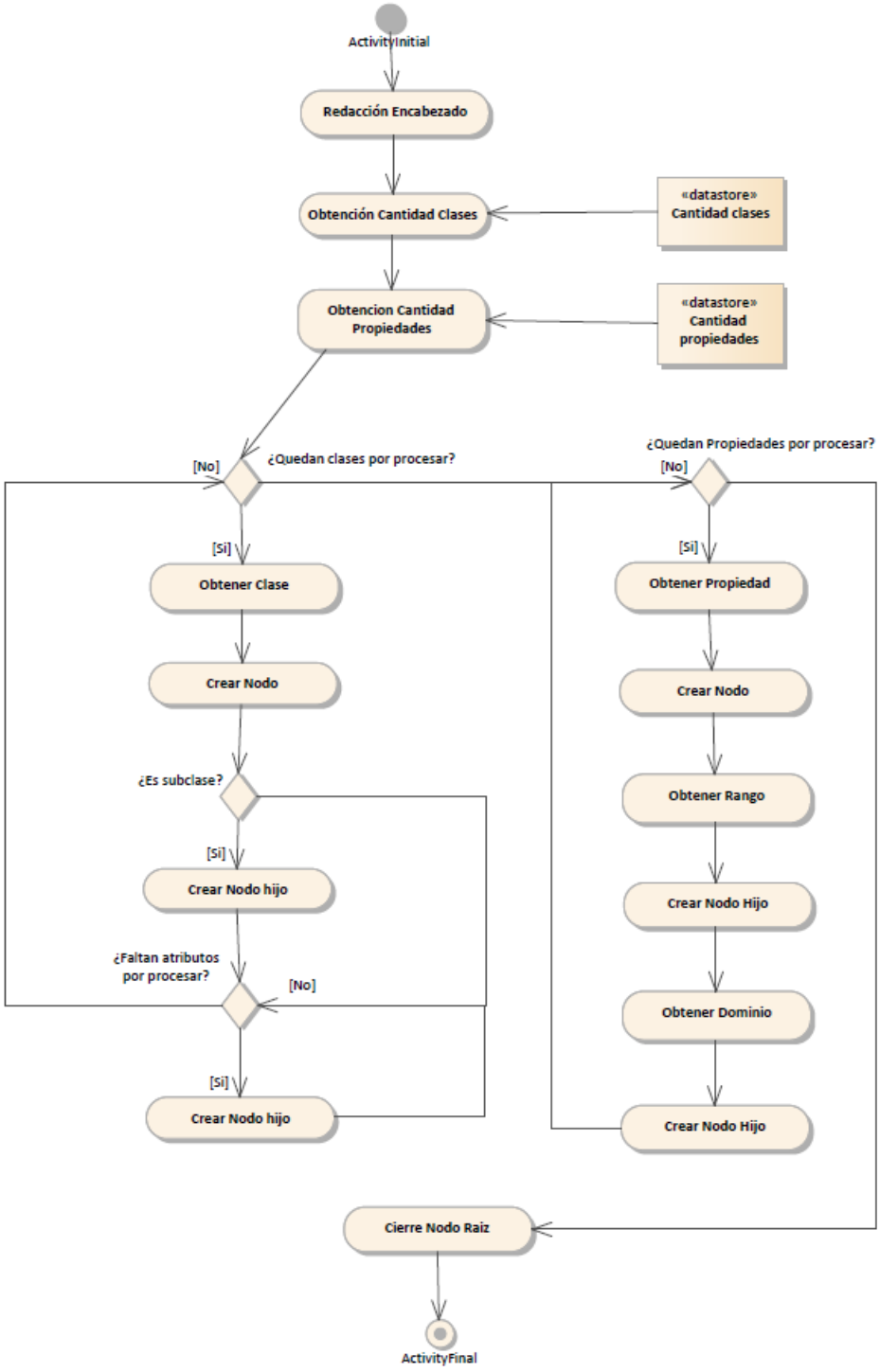


Figura 4–6. Creación documento

5 APLICACIÓN

En este capítulo se detalla la aplicación del diseño expuesto en el capítulo 3. En primer lugar, se mostrará la ontología definida para la estructura PPR, siguiendo todo el proceso incluyendo el paso por las distintas herramientas. A continuación, se expondrá la semántica básica utilizada para poder procesar instancias provenientes de diferentes clientes. Seguidamente, se detallará el porducto tomado como ejemplo, incluyendo su estructura, procesos y recursos. Por último, se presentarán las instancias realizadas y el correspondiente informe, que reflejará la información que se almacena en la base de datos tras pasar el "filtro" de la ontología.

5.1 Definición ontología PPR

Como se mencionó en el apartado 3.3. Diseño, la herramienta usada para la definición de la ontología es Enterprise Architech, la cual seguirá la estructura establecida en el modelo de datos (Figura 3–10). El objetivo es definir una estructura PPR (Product-Process-Resources) que represente los productos, procesos y recursos así como las relaciones entre ellos, de tal forma que la estrcutrura de fabricación quede completamente definida y sea unívoca para todo usuario. Dicha estructura vendrá dada por el modelo de datos de diseño, donde un modelo estará compuesto de clases, que poseerán una serie de atributos y estarán relacionadas con otras clases a través de unas propiedades de objeto.

Fruto de ello, se muestra el diagrama de la Figura 5–1. En el cual se distinguen clases <<*owlClass*>> y propiedades de objetos <<*objectProperty*>>. Las clases tienen una serie de atributos representados en la parte inferior de su caja. Por tanto, cuando se realice una instanciación, la información que será almacenada en la base de datos será la que aparece en el diagrama, es decir, para cada clase se procesarán los atributos que aparecen en su caja y, por su parte, para las propiedades de objeto, sólo se almacenará el dominio y el rango. Cualquier otra información que sea proporcionada, será "ignorada" debido a la definición de la ontología.

Como se puede observar, el producto final es la clase que contiene al resto, pues es el objetivo del proyecto. En este caso, un producto tiene asociado un plan de procesos. Dicha relación se establece, en EA, como una propiedad de objeto, relacionando ambas partes bien como dominio o rango, siendo el rango la clase que se representa como "propiedad" de la principal (dominio). El plan de procesos está formado por el conjunto de procesos necesarios para obtener el producto. Según se ha definido, cada proceso puede ser un cierto tipo (ensamblado, transporte, definido por el ususario,…) identificado mediante la etiqueta de "Subclase". Asimismo, un proceso puede utilizar uno o varios recursos en su desarrollo, recursos que de la misma forma que un proceso, puede ser de varios tipos. Además, un proceso debe llevarse a cabo en un lugar establecido, una localización y podría ocurrir que la realización de un proceso requiriese de otros subprocesos. Otra propiedad asociada sería la de contabilizar determinados parámetros asociados al procesos, KPI (Key Performance Indicators) para llevar un control de la eficiencia del mismo. Por último, todo proceso consume una o más partes y obtiene otra.

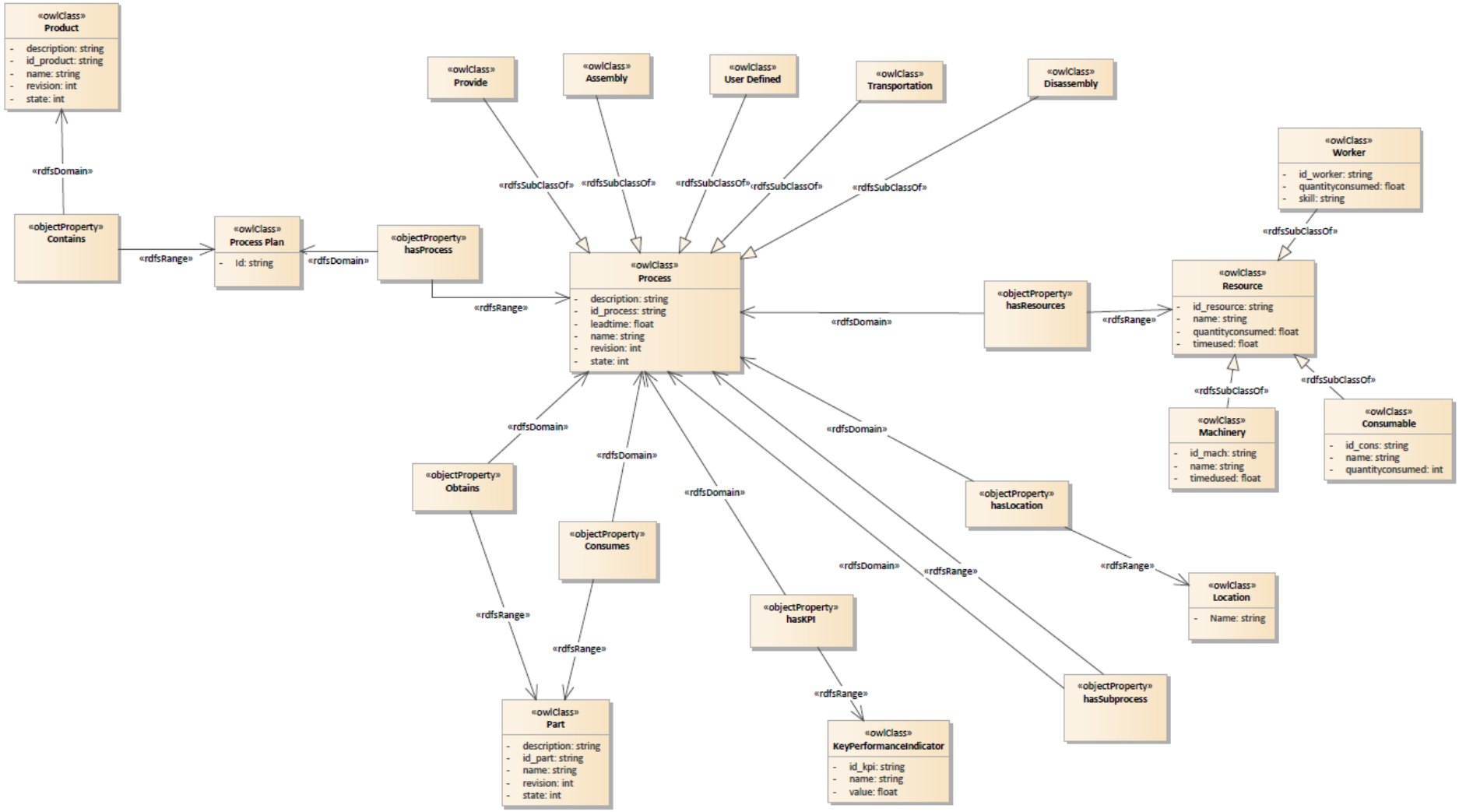


Figura 5–1.Ontología de la Estructura PPR

La exportación del fichero XML tiene el siguiente formato:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:ns0="http://myontologies/newOntology">
<owl:Ontology rdf:ID="newOntology">
<owl:Class rdf:about="http://localhost/default#Assembly" rdf:ID="Assembly">
<rdfs:subClassOf rdf:resource="http://localhost/default#Process" />
</owl:Class>
<owl:Class rdf:about="http://localhost/default#Consumable" rdf:ID="Consumable">
<owl:Class rdf:about="http://localhost/default#Disassembly" rdf:ID="Disassembly">
<owl:Class rdf:about="http://localhost/default#KeyPerformanceIndicator" rdf:ID="KeyPerformanceIndicator" />
<owl:Class rdf:about="http://localhost/default#Location" rdf:ID="Location" />
<owl:Class rdf:about="http://localhost/default#Machinery" rdf:ID="Machinery">
<owl:Class rdf:about="http://localhost/default#Part" rdf:ID="Part" />
<owl:Class rdf:about="http://localhost/default#Process" rdf:ID="Process" />
<owl:Class rdf:about="http://localhost/default#ProcessPlan" rdf:ID="Process Plan" />
<owl:Class rdf:about="http://localhost/default#Product" rdf:ID="Product" />
<owl:Class rdf:about="http://localhost/default#Provide" rdf:ID="Provide">
<owl:Class rdf:about="http://localhost/default#Resource" rdf:ID="Resource" />
<owl:Class rdf:about="http://localhost/default#Transportation" rdf:ID="Transportation">
<owl:Class rdf:about="http://localhost/default#UserDefined" rdf:ID="User Defined">
<owl:Class rdf:about="http://localhost/default#Worker" rdf:ID="Worker">
<owl:ObjectProperty rdf:about="http://localhost/default#Consumes" rdf:ID="Consumes">
<rdfs:range rdf:resource="http://localhost/default#Part" />
<rdfs:domain rdf:resource="http://localhost/default#Process" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://localhost/default#Contains" rdf:ID="Contains">
<owl:ObjectProperty rdf:about="http://localhost/default#Obtains" rdf:ID="Obtains">
<owl:ObjectProperty rdf:about="http://localhost/default#hasKPI" rdf:ID="hasKPI">
<owl:ObjectProperty rdf:about="http://localhost/default#hasLocation" rdf:ID="hasLocation">
<owl:ObjectProperty rdf:about="http://localhost/default#hasProcess" rdf:ID="hasProcess">
<owl:ObjectProperty rdf:about="http://localhost/default#hasResources" rdf:ID="hasResources">
<owl:ObjectProperty rdf:about="http://localhost/default#hasSubprocess" rdf:ID="hasSubprocess">
<owl:Thing rdf:ID="Fact1" />
<owl:Thing rdf:ID="Fact2" />
<owl:Thing rdf:ID="Fact3" />
<owl:Thing rdf:ID="Fact4" />
</rdf:RDF>
```

Figura 5–2. Exportación Enterprise Architect

Como es posible observar, no aparecen los atributos definidos asociados a las clases, sólo su referencia e identificador. Por su parte, en las propiedades de objeto sí aparecen rango y dominio. Es por este motivo que se requiere la herramienta Protégé, pues como ella se añadirán dichos atributos.

Previo al paso por Protégé, es necesario introducir el modelo en ARAS, para "filtar" ese atributo *rdf:ID* que dificulta la utilización de Protégé.

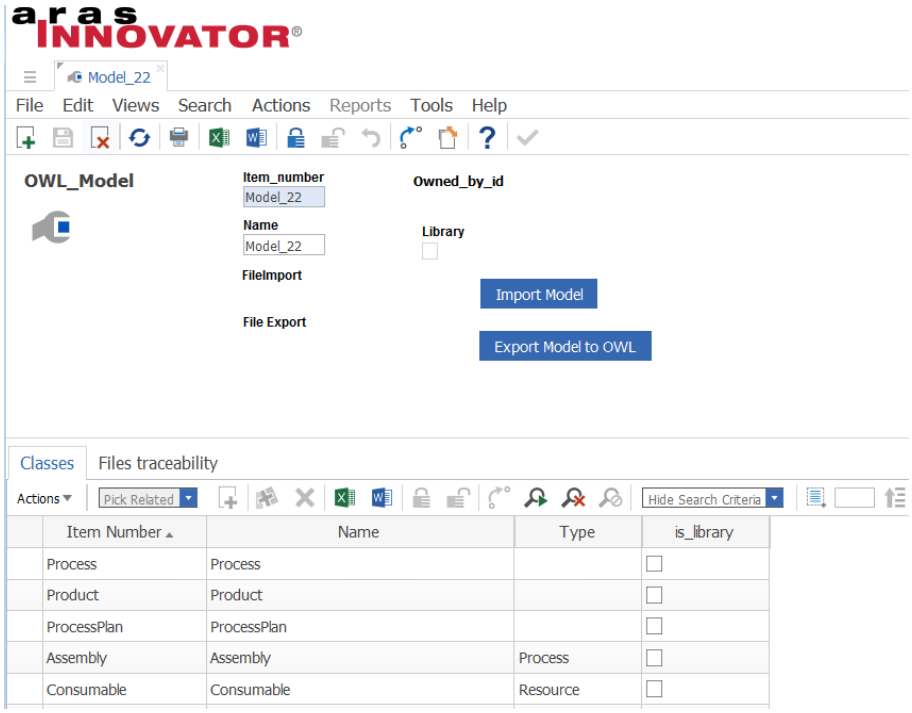


Figura 5–3. Menú modelo importar/exportar

El resultado de la importación y exportación de ARAS, se muestra en la siguiente figura:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://localhost/default#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">

<owl:Class rdf:about="http://localhost/default#Assembly">
 <rdfs:subClassOf rdf:resource="http://localhost/default#Process"/>
</owl:Class>
<owl:Class rdf:about="http://localhost/default#Consumable">
<owl:Class rdf:about="http://localhost/default#Disassembly">
<owl:Class rdf:about="http://localhost/default#KeyPerformanceIndicator">
<owl:Class rdf:about="http://localhost/default#Location">
<owl:Class rdf:about="http://localhost/default#Machinery">
<owl:Class rdf:about="http://localhost/default#Part">
<owl:Class rdf:about="http://localhost/default#Process">
<owl:ObjectProperty rdf:about="http://localhost/default#Consumes">
    <rdfs:range rdf:resource="http://localhost/default#Part"/>
    <rdfs:domain rdf:resource="http://localhost/default#Process"/>
    <rdfs:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://localhost/default#Obtains">
<owl:ObjectProperty rdf:about="http://localhost/default#hasKPI">
<owl:ObjectProperty rdf:about="http://localhost/default#hasLocation">
<owl:ObjectProperty rdf:about="http://localhost/default#hasResources">
<owl:ObjectProperty rdf:about="http://localhost/default#hasSubprocess">
<owl:Class rdf:about="http://localhost/default#ProcessPlan">
<owl:ObjectProperty rdf:about="http://localhost/default#hasProcess">
<owl:Class rdf:about="http://localhost/default#Product">
<owl:ObjectProperty rdf:about="http://localhost/default#Contains">
<owl:Class rdf:about="http://localhost/default#Provide">
<owl:Class rdf:about="http://localhost/default#Resource">
<owl:Class rdf:about="http://localhost/default#Transportation">
<owl:Class rdf:about="http://localhost/default#UserDefined">
<owl:Class rdf:about="http://localhost/default#Worker">
</rdf:RDF>
```

Figura 5–4. Exportación ARAS

Puesto que simplemente se ha importado y exportado el modelo, la información que contiene, en cuanto a clases y propiedades de objeto, permanece intacta. En este formato, ya es posible abrirlo con Protégé, donde se podrán añadir los atributos correspondientes. Para añadir atributos, se debe seguir el siguiente proceso:

En primer lugar, se selecciona la clase que se quiera modificar:

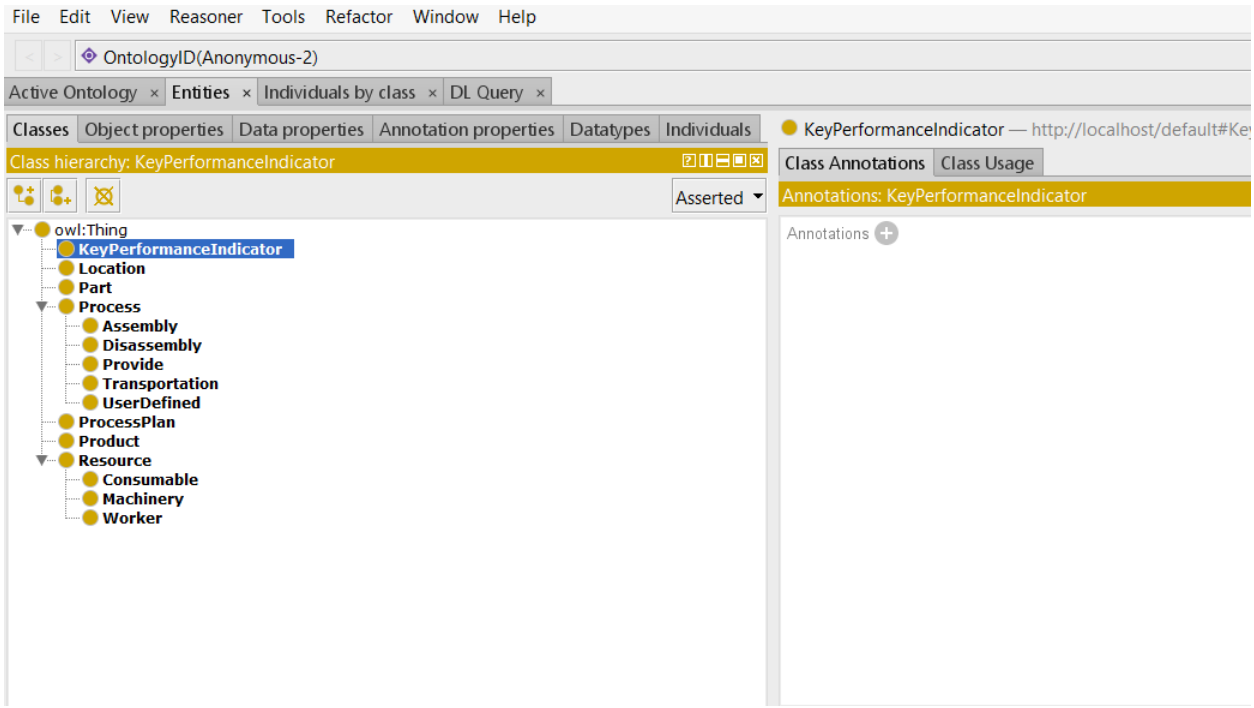


Figura 5–5. Menú principal Protégé

A continuación, para añadir el atributo se debe seleccionar primero la clase y en la ventana de la derecha denominada “Class Anotation”, clicar sobre el símbolo . Tras ello, aparecerá la siguiente ventana:

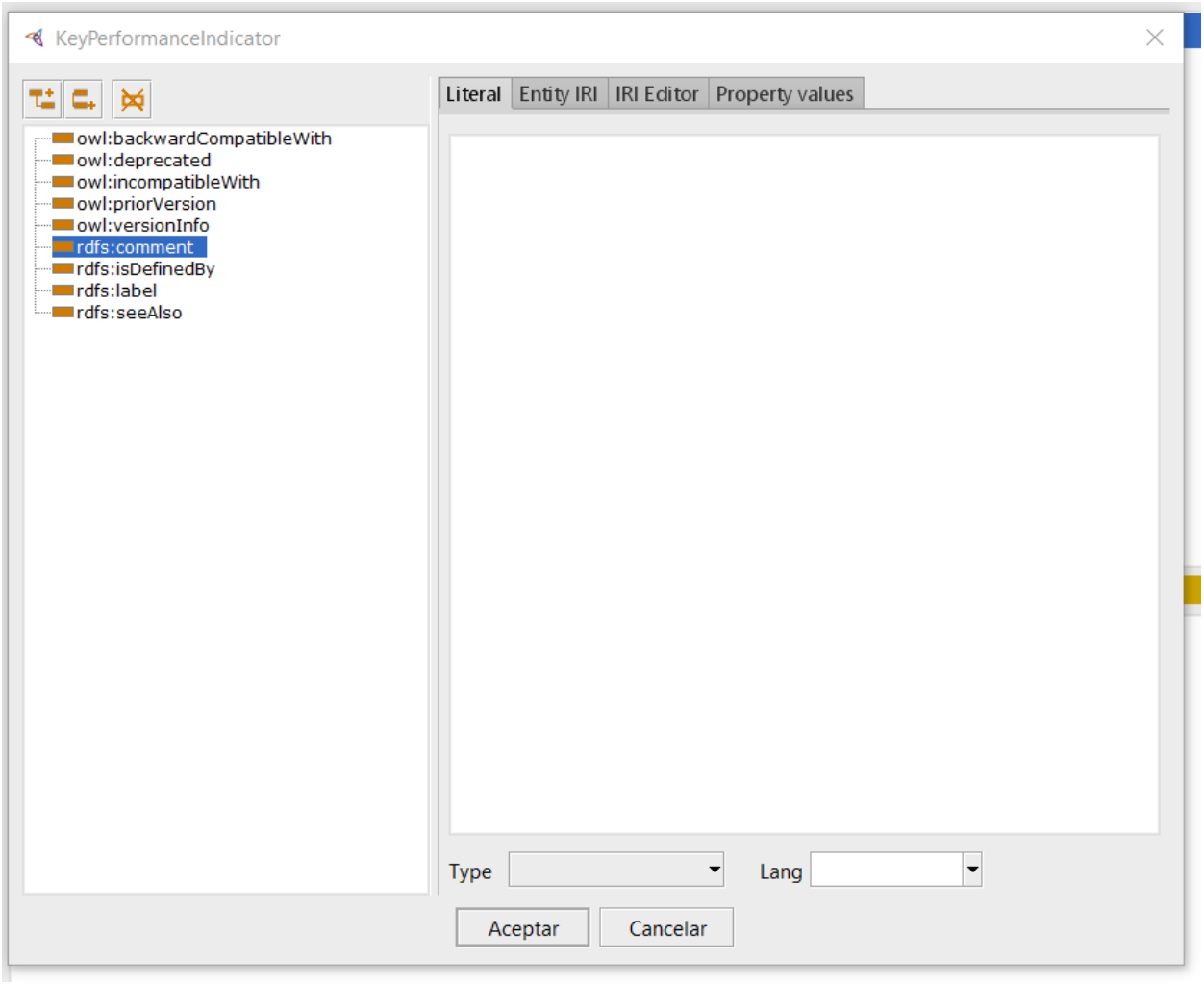


Figura 5–6. Ventana anotaciones

Donde se muestran las distintas anotaciones predeterminadas. Además de estas, es posible añadir las que el usuario desee; por ejemplo, para añadir una anotación del tipo "tiene", basta con clicar sobre el botón  , que provocará la aparición de una segunda ventana donde se podrá especificar el nombre de la anotación:

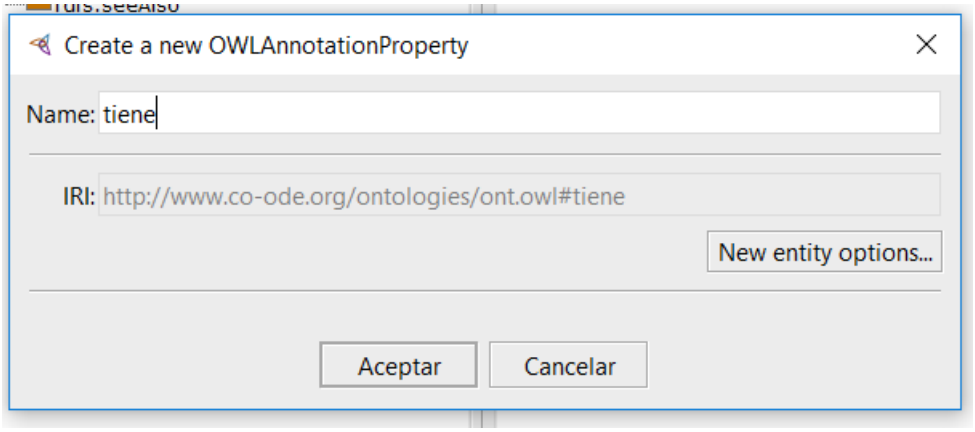


Figura 5–7. Creación de nueva anotación

Tras aceptar, ya aparecerá en la ventana de anotaciones y podremos especificar el tipo de anotación (xsd:*string, xsd:double, …*) y su nombre (*Literal*).

Al finalizar y exportar, obtendríamos el siguiente XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://localhost/default#"
     xml:base="http://localhost/default"
     xmlns:ont="http://www.co-ode.org/ontologies/ont.owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     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#">
    <owl:Ontology/>
    <!--
    <!-- http://www.co-ode.org/ontologies/ont.owl#has -->
    <owl:AnnotationProperty rdf:about="http://www.co-ode.org/ontologies/ont.owl#has"/>
    <!--
   <!-- http://localhost/default#Consumes -->
   <owl:ObjectProperty rdf:about="http://localhost/default#Consumes">
    <!-- http://localhost/default#Contains -->
   <owl:ObjectProperty rdf:about="http://localhost/default#Contains">
    <!-- http://localhost/default#Obtains -->
    <owl:ObjectProperty rdf:about="http://localhost/default#Obtains">
    <!-- http://localhost/default#hasKPI -->
    <owl:ObjectProperty rdf:about="http://localhost/default#hasKPI">
    <!-- http://localhost/default#hasLocation -->
    <owl:ObjectProperty rdf:about="http://localhost/default#hasLocation">
    <!-- http://localhost/default#hasProcess -->
    <owl:ObjectProperty rdf:about="http://localhost/default#hasProcess">
    <!-- http://localhost/default#hasResources -->
    <owl:ObjectProperty rdf:about="http://localhost/default#hasResources">
    <!-- http://localhost/default#hasSubprocess -->
    <owl:ObjectProperty rdf:about="http://localhost/default#hasSubprocess">
    <!-- http://localhost/default#Process -->
    <owl:Class rdf:about="http://localhost/default#Process">
        <ont:has rdf:datatype="http://www.w3.org/2001/XMLSchema#string">description</ont:has>
        <ont:has rdf:datatype="http://www.w3.org/2001/XMLSchema#string">id_process</ont:has>
        <ont:has rdf:datatype="http://www.w3.org/2001/XMLSchema#float">leadtime</ont:has>
        <ont:has rdf:datatype="http://www.w3.org/2001/XMLSchema#string">name</ont:has>
        <ont:has rdf:datatype="http://www.w3.org/2001/XMLSchema#int">revision</ont:has>
        <ont:has rdf:datatype="http://www.w3.org/2001/XMLSchema#int">state</ont:has>
    </owl:Class>
    <!-- http://localhost/default#ProcessPlan -->
    <owl:Class rdf:about="http://localhost/default#ProcessPlan">
    <!-- http://localhost/default#Product -->
    <owl:Class rdf:about="http://localhost/default#Product">
    <!-- http://localhost/default#Provide -->
    <owl:Class rdf:about="http://localhost/default#Provide">
    <!-- http://localhost/default#Resource -->
    <owl:Class rdf:about="http://localhost/default#Resource">
    <!-- http://localhost/default#Transportation -->
    <owl:Class rdf:about="http://localhost/default#Transportation">
    <!-- http://localhost/default#UserDefined -->
    <owl:Class rdf:about="http://localhost/default#UserDefined">
    <!-- http://localhost/default#Worker -->
    <owl:Class rdf:about="http://localhost/default#Worker">
</rdf:RDF>
```

Figura 5–8. Exportación Protégé

Se puede observar que ahora sí aparecen los atributos asociados a cada clase. Por tanto, el último paso consistiría en importar de nuevo el modelo en ARAS.

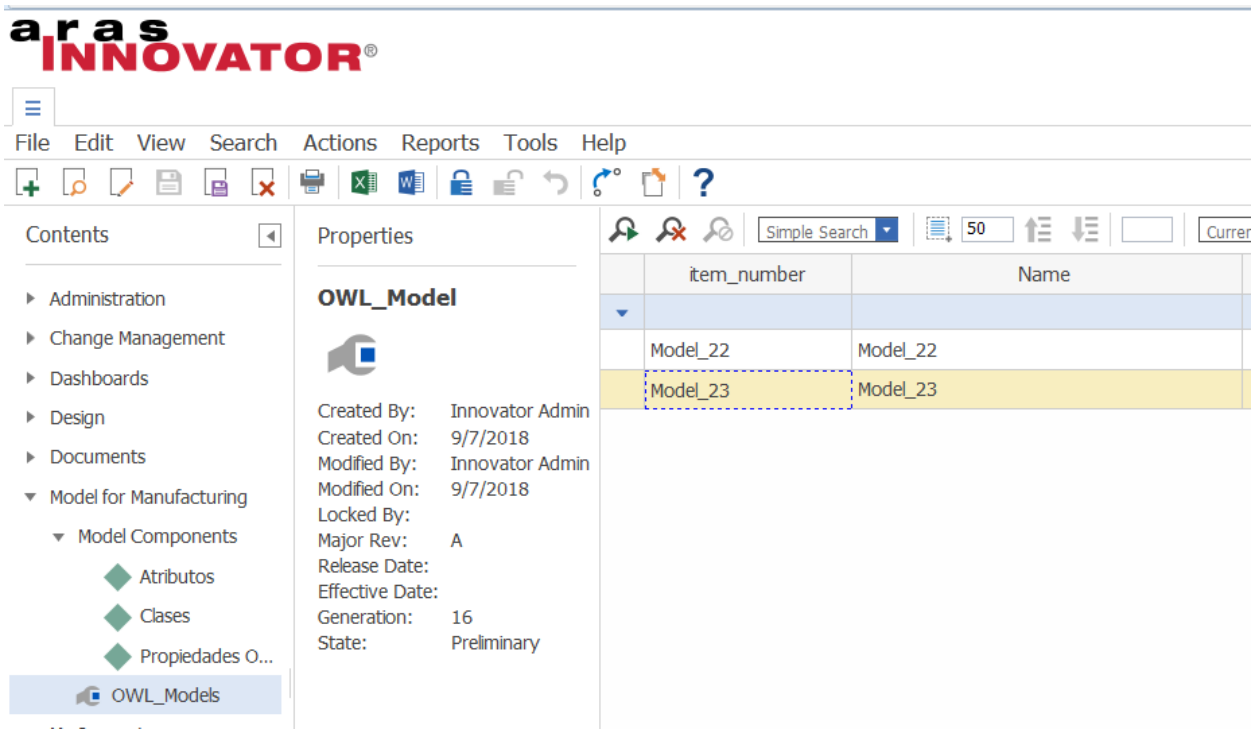


Figura 5–9. Menú de modelos OWL importados

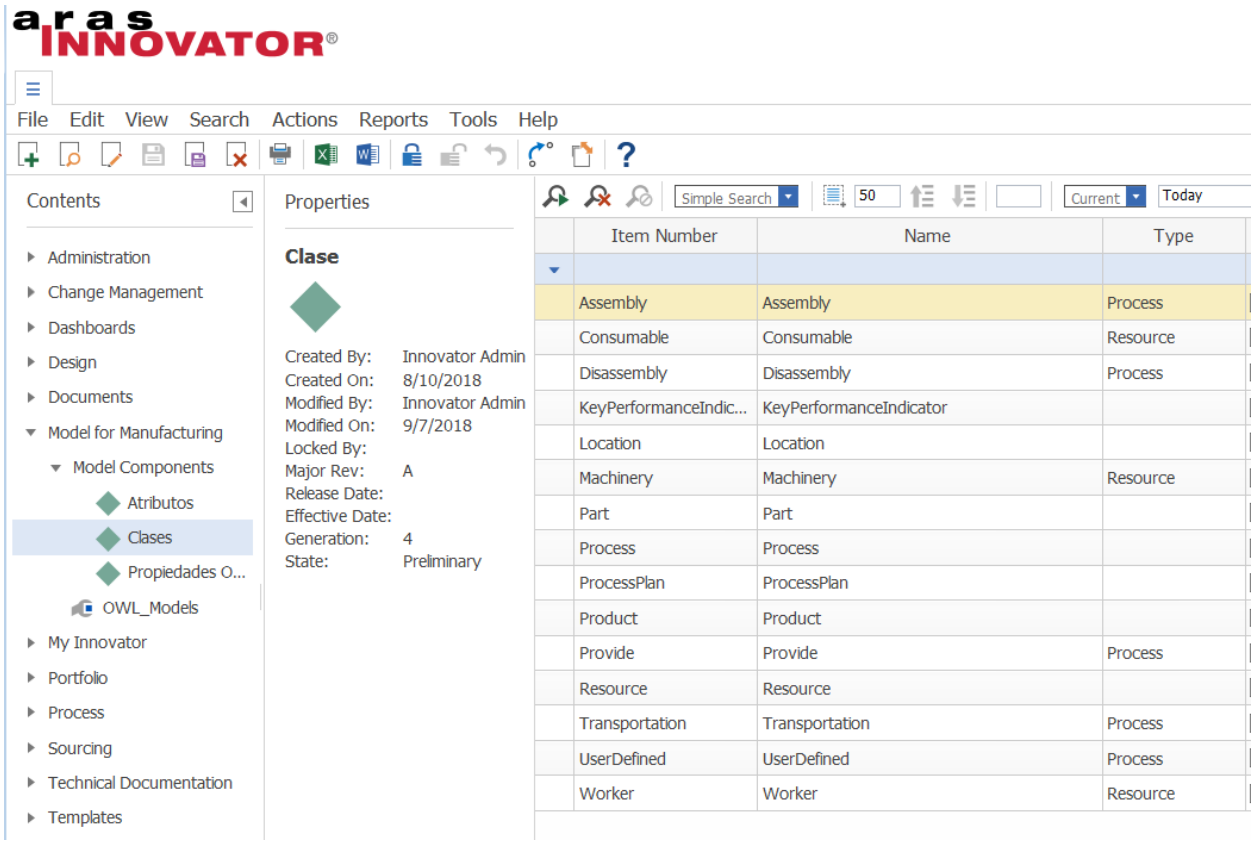


Figura 5–10. Menú de clases importadas

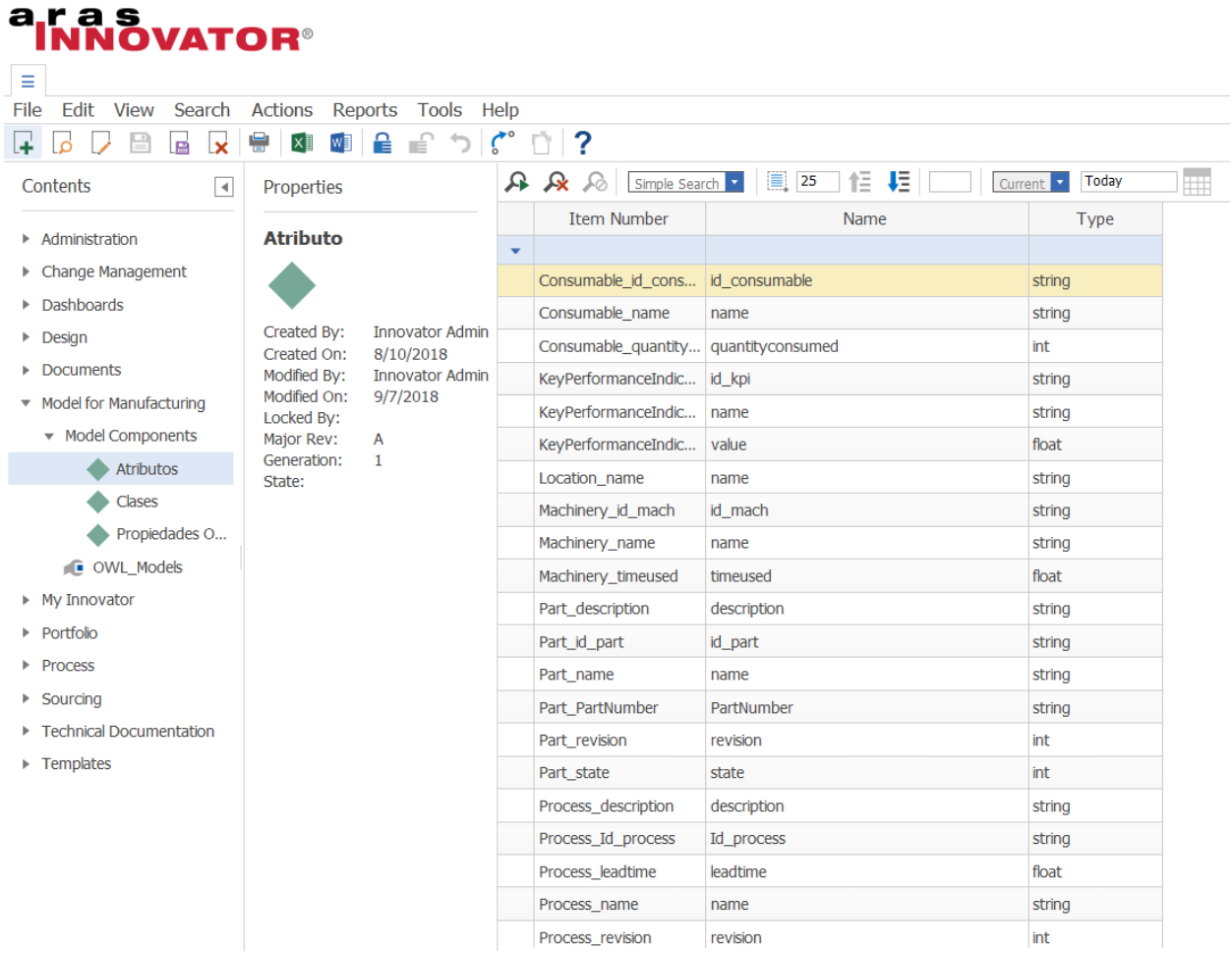


Figura 5–11. Menú de atributos de clase

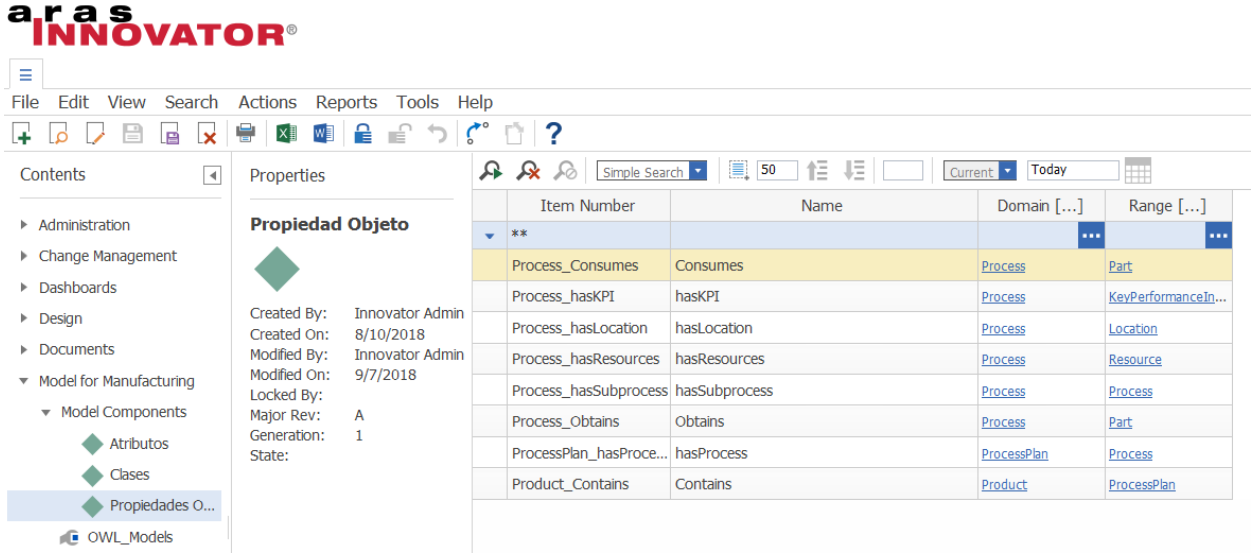


Figura 5–12. Menú de propiedades de objetos

5.2 Semántica

Una de las ventajas del uso de ontologías y uno de los principales factores que favorecen la interoperabilidad es su semántica, capaz de reconocer y unificar instancias provenientes de distintos softwares o herramientas bajo su definición unívoca de, en este caso, la estructura PPR original.

En el caso que nos ocupa, se llevarán a cabo dos instanciaciones de dos "clientes" diferentes. Por tanto, existe gran probaibilidad de que cada uno defina de forma distinta la denominación de los nodos del XML de la instancia.

Como ejemplo de una semántica básica se muestra la siguiente tabla:

Modelo	**Instancia 1**	**Instancia 2**
id	id	instance_number
name	nombre	name
type	tipo	type
value	valor	value
description	descripción	description
quantityconsumed	cantidad	quantity
timeused	tiempouso	timeused
state	estado	state
Part	Parte	Part
Process	Proceso	Process
Resource	Recurso	Resource
KPI	KPI	KPI
Location	Localizacion	Location
Process Plan	PlanProceso	ProcessPlan
hasProcessPlan	Producto_PlanProceso	Product_ProcessPlan
hasProcess	PlanProceso_Producto	ProcessPlan_Product
hasResources	Proceso_Recurso	Process_Resource
hasLocation	Proceso_Localizacion	Process_Location
hasKPI	Proceso_KPI	Process_KPI
Consumes	Proceso_Consume	Process_Consumes
Obtains	Proceso_Obtiene	Process_Obtains

Tabla 5-1. Tabla de semántica

5.3 Producto

El producto que se va a tomar como ejemplo es un turbohélice modelo 6745 de la marca Lego©, el cual se muestra a continuación.

Figura 5–13. Turbohélice 6745-AH

5.3.1 Captura de requisitos

Tras el análisis previo del turbohélice llevado a cabo con el software LEGO Designer®, se decide descomponer el modelo en las siguientes partes:

- Superficies sustentadoras. Compuestas por las alas y los estabilizadores (HTP y VTP).

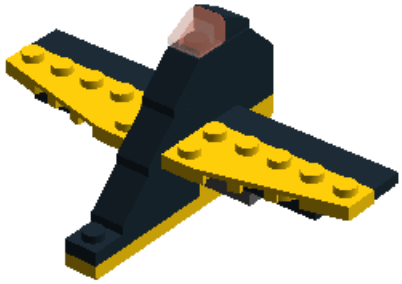


Figura 5–14. Grupo estabilizadores

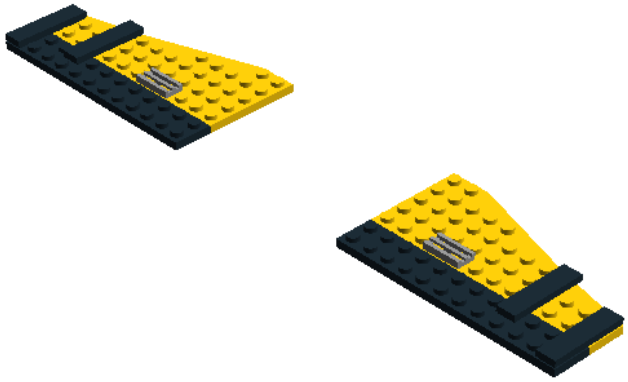


Figura 5–15. Grupo alar

- Estructura base. Compuesta por el fuselaje inferior y el superior.

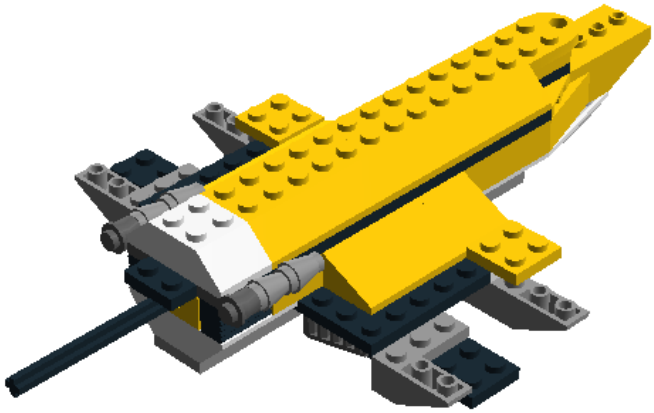


Figura 5–16. Estructura base

- Tren de aterrizaje.

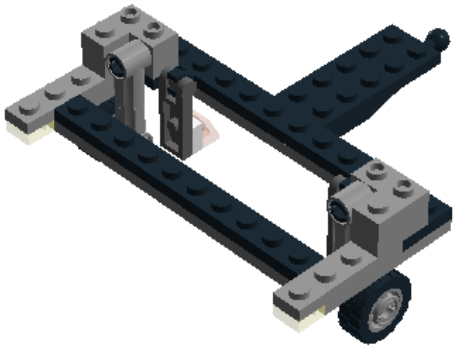


Figura 5–17. Tren de aterrizaje

- Cabina.

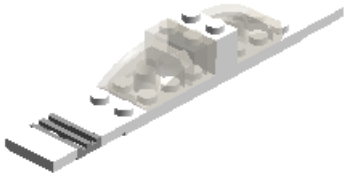


Figura 5–18. Cabina

- Grupo motor. Compuesto por el motor y la hélice.

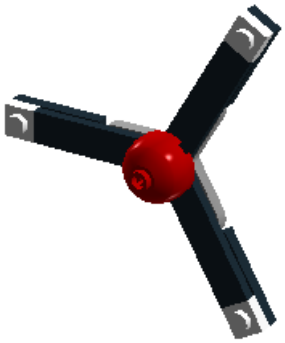


Figura 5–19. Hélice

Figura 5–20. Motor

Requisitos asociados al proyecto 6745-AH:

- Unidades de medida: La unidad de medida utilizada en este proyecto es el *brick* 1x1. Un *brick* 1x1 es un componente básico de montaje. Como complemento a este *brick*, se utiliza la unidad de medida *plate*, con una altura 1/3 de la del *brick*.

Asociado a cada parte, se especifican una serie de requisitos:

- Requisitos cola. La cola está formada por sendos estabilizadores horizontal y vertical. El estabilizador horizontal se descompone en dos "semialas" cada una de 5 *bricks* de anchura y 3 de longitud, si bien los componentes usados son *plates* al tratarse de una superficie aerodinámica. Por su parte, el estabilizador vertical tiene una anchura de un *brick* y una altura de 4 bricks. En la punta de dicho estabilizador es donde se coloca la luz. La unión entre ambos estabilizadores se realiza mediante la pieza 4514553.

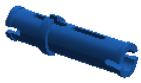


Figura 5–21. Componente 4514553

- Requisitos alas. Formada por dos semialas, cada una con una semienvergadura de 12 *bricks* y una cuerda en el encastre de 7 *bricks*. Al igual que el HTP estos *bricks* son del tipo *plate* al tratarse de superficies aerodinámicas. Se trata de un ala que tiene una cierta flecha en el borde de salida siendo recta en el borde de ataque, pues al ser un turbohélice no alcanza velocidades cercanas a M=1.

- Requisitos estructura. La estructura del turbohélice está formada por el fuselaje inferior y el superior.

    - Fuselaje inferior. Parte inferior de la estructura donde se ensambla el tren de aterrizaje. Tiene una longitud de 18 *bricks* y una achura de 16.
    - Fuselaje superior. Corresponde al bloque superior del turbohélice exceptuando la cabina de mando. A éste también se le une directamente la cola en la parte posterior, así como el grupo motor en la parte anterior. Tiene una longitud de 21 *bricks* y una achura de 12.

- Requisitos tren de aterrizaje. El tipo de tren de aterrizaje que tiene este turbohélice es uno convencional compuesto por un tren principal debajo del fuselaje a la altura del ala y un patín de cola. Se opta por este tipo debido a su pequeño tamaño pues exige menor potencia del motor.

- Requisitos de cabina. La cabina se conecta directamente al fuselaje superior mediante una unión simple que no requiere de sujeciones externas. Tiene una longitud de 13 *bricks* y una anchura de 2.

- Requisitos grupo motor. Compuesto por el motor y la hélice.

    - Requisitos hélice. Hélice de tres aspas de longitud 5 bricks, formado por componentes tipo *plate.* La sujeción viene dada por el elemento 4216657.

Figura 5–22. Componente 4216657

    - Requisitos motor. Motor de eje único en el que la hélice se conecta a la turbina a través de la reductora.

5.3.2 Estructura MBOM

En cuanto al MBOM (*Manufacturing Bill of Materials)*, en este se definen los materiales necesarios para la fabricación en un determinado orden. También es denominada como lista de materiales *as Planned* porque representa la estructura en que se industrializa o fabrica el producto. En este caso en el MBOM lo que se recoge es la planificación de ensamblados.

En esta estructura de producto, o estructura *as Planned*, se representan los grupos de ensamblados o cambios (productos intermedios) que van sufriendo los elementos en base a diferentes operaciones. De este modo los

productos pueden ser:

- Ensamblados
- Componentes

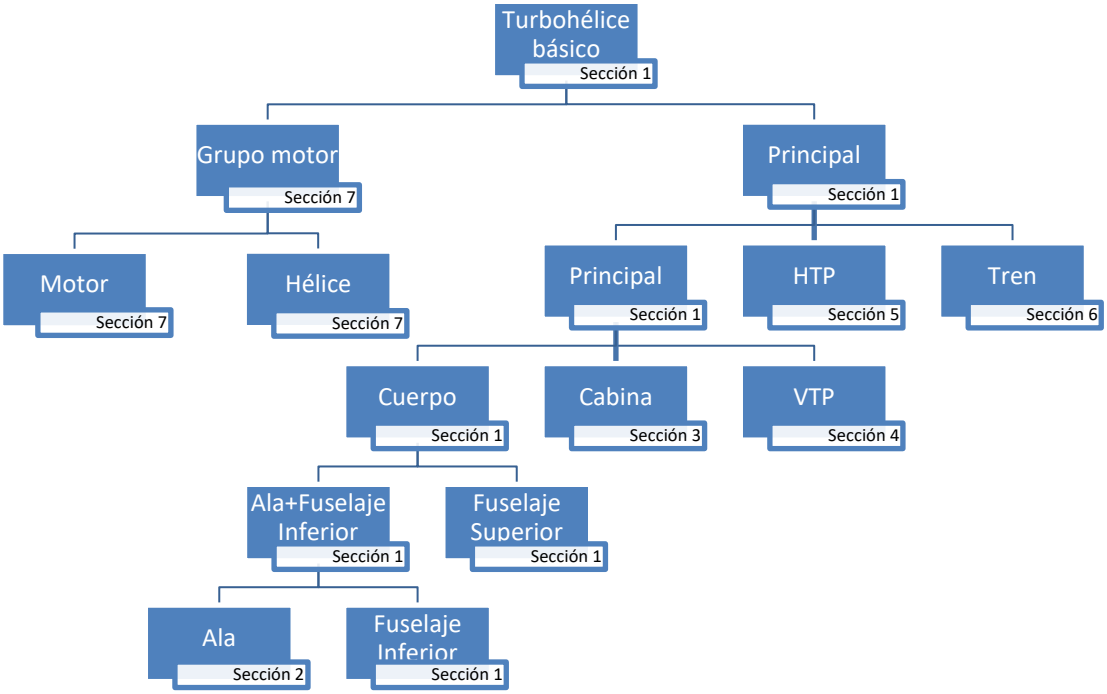


Figura 5–23. Estructura MBOM

Como se puede ver en la Figura 5–23, cada uno de los cuadros azules representa un proceso. Por un lado, cada una de las partes o componentes en las que se divide el avión son fabricadas individualmente en su estación. Posteriormente, estas son trasladadas a la sección principal (Sección 1) donde se van ensamblando. Los procesos son los siguientes:

1. Fabricación Alas. Unión de las piezas lego que forman las alas.
2. Fabricación Fuselaje Inferior. Unión de las piezas lego que forman el fuselaje inferior.
3. Transporte Alas. Transporte del grupo alar a la estación ensamblaje principal.
4. Ensamblaje Fus Inf-Ala. Ensamblaje del ala al fuselaje inferior.
5. Fabricación Fuselaje Superior. Unión de las piezas lego que forman el fuselaje superior.
6. Ensamblaje Fus Inf-Ala-Fus Sup. Ensamblaje del fuselaje superior a la unión ala - fuselaje inferior.
7. Fabricación Cabina. Unión de las piezas lego que forman la cabina.
8. Fabricación VTP. Unión de las piezas lego que forman el VTP.
9. Transporte Cabina. Transporte cabina a la estación ensamblaje principal.
10. Transporte VTP. Transporte VTP a la estación ensamblaje principal.
11. Ensamblaje Cuerpo-Cab-VTP. Ensamblaje de la cabina y el VTP a la estructura principal.
12. Fabricación HTP. Unión de las piezas lego que forman el HTP.
13. Fabricación Tren. Unión de las piezas lego que forman el Tren.
14. Transporte HTP. Transporte HTP a la estación ensamblaje principal.
15. Tranporte Tren. Transporte tren a la estación ensamblaje principal.
16. Ensamblaje Principal-HTP-Tren. Ensamblaje de la cabina y el VTP al cuerpo.
17. Fabricación Motor. Unión de las piezas lego que forman el motor.

18. Fabricación Hélice. Unión de las piezas lego que forman la hélice.
19. Ensamblaje Motor-Hélice. Ensamblaje grupo motor.
20. Transporte GM. Transporte GM a la estación ensamblaje principal.
21. Ensamblaje Principal-GM. Ensamblaje del GM a la estructura principal.

Todos estos procesos estarán recogidos en un plan de procesos, y llevarán asociados localizaciones (secciones), recursos que intervendrán en los mismos y parámetros indicadores de rendimiento (KPI).

- Localizaciones:
    1. Sección 1
    2. Sección 2
    3. Sección 3
    4. Sección 4
    5. Sección 5
    6. Sección 6
    7. Sección 7
- Recursos:
    1. Operario
    2. Grúa
    3. Remachadora
    4. Remaches

- KPIs:
    1. OEE. Overall equipment effectiveness. Indicador relacionado con la eficiencia de los equipos. Sirve para medir la eficiencia proudctiva de una determinada máquina.
    2. OLE. Overall labour effectiveness. Indicador para medir la utilización, trabajo y calidad de los trabajadores y su impacto en la productividad.

Las relaciones de los procesos con recursos, localizaciones y KPIs se muestran en el Anexo 7.1.

5.4 Instanciación

Tras la importación del modelo ontológico en la base de datos, se procede a la inclusión de un par de ejemplos o instancias para comprobar su correcto funcionamiento. A continuación se presentan las dos instancias correspondientes a dos clientes disntintos, en concreto, se muestra un ejemplo del archivo XSD, si bien en el anexo 7.2 se muestra el archivo XML al completo:

- Instancia 1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="PPR">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Producto" maxOccurs="unbounded" minOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="id" type="xs:string"></xs:element>
                            <xs:element name="nombre" type="xs:int"></xs:element>
                            <xs:element name="descripcion" type="xs:string"></xs:element>
                            <xs:element name="revision" type="xs:int"></xs:element>
                            <xs:element name="estado" type="xs:int"></xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="PlanProceso" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Producto_PlanProceso" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Parte" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Proceso" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Recurso" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="KPI" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Localizacion" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="PlanProceso_Proceso" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Proceso_Consume" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Proceso_Obtiene" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Proceso_Recurso" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Proceso_KPI" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Proceso_Localizacion" maxOccurs="unbounded" minOccurs="1">
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figura 5–24. Instancia 1 XSD

El archivo XSD detalla la información que recoge el XML, es decir, existirán las clases que en él aparecen (Producto, PlanProceso, etc), especificando si debe existir un máximo o mínimo de apariciones (en este caso, son ilimitados y uno, respectivamente) y los elementos de cada clase.

- Instancia 2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="PPR">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Product" maxOccurs="unbounded" minOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="instance_number" type="xs:string"></xs:element>
                            <xs:element name="name" type="xs:int"></xs:element>
                            <xs:element name="description" type="xs:string"></xs:element>
                            <xs:element name="revision" type="xs:int"></xs:element>
                            <xs:element name="state" type="xs:int"></xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="ProcessPlan" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Product_ProcessPlan" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Part" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Process" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Resource" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="KPI" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Location" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="ProcessPlan_Process" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Process_Consumes" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Process_Obtains" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Process_Resource" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Process_KPI" maxOccurs="unbounded" minOccurs="1">
                <xs:element name="Process_Location" maxOccurs="unbounded" minOccurs="1">
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figura 5–25. Instancia 2 XSD

Como puede observarse, la definición de las instancias es diferente puesto que cada cliente tiene una forma de proceder distinta.

Si bien el procedimiento de importación de cada instancia es similar, asociada a cada una deberá crearse una interfaz adecuada para dicho procesamiento, de tal forma que la semántica particular de cada instancia sea "traducida" a la establecida por la ontología.

5.4.1 Informe

Tras la instanciación de ambos ejemplos, se obtiene la siguiente lectura, que debería coincidir en ambos casos puesto que contienen idéntica información. Como se puede observar en el ejemplo mostrado a cotninuación, sendas interfaces han "traducido" el contenido para expresarlo en la semántica del modelo almacenado en ARAS:

```
<Result>

Product
id_product:1_TH
name:1
description:Turbohelice
revision:0
state:0

ProcessPlan
id_processplan:2

Product_ProcessPlan
domain(id_product):1
range(id_processplan):2

Part
id_part:307026_HTP
name:307026
description:FLAT TILE 1X1
revision:0
state:0

Part
id_part:302324_HTP
name:302324
description:PLATE 1X2
revision:0
state:0

Process
id_process:3
name:Fabricación Alas
description:Unión de las piezas lego que forman las alas
leadtime:12
revision:0
state:0

Process
id_process:4
name:Fabricación Fuselaje Inferior
description:Unión de las piezas lego que forman el fuselaje inferior
leadtime:20
revision:0
state:0

Resource
id_resource:100
name:Operario
quantityconsumed:93
timeused:288

Resource
id_resource:101
name:Grúa
quantityconsumed:0
timeused:12

KPI
id_kpi:201
name:OEE
value:75

KPI
id_kpi:202
name:OLE
value:80

Location
name:Stación 1

Location
name:Stación 2

ProcessPlan_Process
domain(id_processplan):2
range(id_process):3

ProcessPlan_Process
domain(id_processplan):2
range(id_process):4

Process_Consumes
domain(id_process):3
range(id_part):2412194_Alas

Process_Obtains
domain(id_process):3
range(id_part):50000_Alas

Process_Resource
domain(id_process):3
range(id_resource):100
quantityconsumed:5
timeused:12

Process_Resource
domain(id_process):3
range(id_resource):101
quantityconsumed:0
timeused:3

Process_KPI
domain(id_process):3
range(id_kpi):201

Process_KPI
domain(id_process):3
range(id_kpi):202

Process_Location
domain(id_process):3
range(name):Sección 2

Process_Location
domain(id_process):4
range(name):Sección 1


</Result>
```

Figura 5–26. Informe ARAS

Al ser una simple lectura, lo que se muestra es toda la información, es decir, el documento no ha "pasado" por ningún tipo de filtro, como consecuencia, es posible notar la presencia de atributos en determinados nodos, en concreto, en aquellos asociados a relaciones. Si ambas instancias hubiesen sido procesadas completamente e introducidas en la base de datos, ninguno de esos atributos (Figura 5–27) aparecerían en ARAS, ya que la ontología no los contempla. Esta situación podría resolverse añadiendo en las propiedades de objeto la capacidad para almacenar atributos, mediante su definición en la ontología y la correspondiente relación entre ItemTypes en ARAS.

```
Process_Resource
domain(id_process):3
range(id_resource):100
quantityconsumed:5
timeused:12

Process_Resource
domain(id_process):3
range(id_resource):101
quantityconsumed:0
timeused:3
```

Figura 5–27. Ejemplo atributos no filtrados

Asimismo, para tipos procesos y recursos, al estar definidos como subclases y no como atributos de "tipo", en la lectura todos los procesos se almacenan como tal, sin clasificarse según la subclase:

```
Process
id_process:3
name:Fabricación Alas
description:Unión de las piezas lego que forman las alas
leadtime:12
revision:0
state:0

Process
id_process:4
name:Fabricación Fuselaje Inferior
description:Unión de las piezas lego que forman el fuselaje inferior
leadtime:20
revision:0
state:0
```

Figura 5–28. Ejemplo definición procesos

6 CONCLUSIONES Y EXTENSIONES

Aunque el desarrollo de proyectos mediante MBSE es relativamente reciente, el crecimiento experimentado y el interés que ha suscitado, han creado nuevos problemas en relación a la interoperabilidad y colaboración entre modelos. El presente proyecto ha abordado esta problemática siempre usando como base el concepto de ontología e incorporando las capacidades de un sistema PLM en la gestión colaborativa.

El prototipo desarrollado en este proyecto ha permitido establecer un método de definición de ontologías mediante la herramienta Enterprise Architect y su gestión, que se ha realizado haciendo uso del software PLM ARAS Innovator. Así como del estudio del impacto de dicha definción de la ontología sobre el desarrollo de un proyecto.

Tras el desarrollo y aplicación de la metodología se han alcanzado las siguientes conclusiones:

En primer lugar, en cuanto a las herramientas usadas, es posible afimar que Enterprise Architect es una herramienta sencilla, intuitiva y cómoda de usar para la definición de ontologías, si bien es cierto que es algo rígida pues no permite exportar los atributos definidos en las clases por lo que es necesario recurrir a herramientas secundarias para esa labor, en este caso Protégé. El hecho de no usar Protégé desde el principio se debe a su complejidad, ya que en su caso la definición de ontología no es gráfica, a diferencia de Enterprise Architect, es decir, es necesario conocer con más detalle su funcionamiento.

En segundo lugar, se ha comprobado la utilidad de un PLM como gestor de modelos de información, así como su carácter interoperable a la hora de realizar lecturas de instancias proporcionadas por herramientas distintas y con diferente definición. Además de la posibilidad de gestionar el ciclo de vida del modelo ontológico, realizando revisiones del mismo, como de los componentes del mismo (producto, procesos, partes, recursos).

En tercer lugar, se ha podido observar tras las instanciaciones realizadas que la definición de la ontología juega un papel mayor en el desarrollo de un proyecto. Como se vio en el informe, el hecho de definir los tipos de procesos y recursos como subclases de sus originales, limita su clasificación y el acceso a dichas clases por tipos; hecho que podría ser útil a la hora de manejar procesos y recursos. Asimismo, se destacó la ausencia de atributos en las propiedades de objeto, lo cual sería de utilidad para saber en el caso de recursos cuánto consume un determinado proceso o en el caso de los KPI, evitaría tener que duplicar el mismo indicador para todos los procesos que lo usen, pues cada uno tomará un valor distinto. Estas limitaciones pueden tomarse como parte de posibles revisiones futuras que mejoren la ontología establecida en este trabajo para, poco a poco, alcanzar una definción más eficiente.

Otro aspecto ampliable se trata de la inclusión de más conceptos a la ontología, como puede ser la inclusión de otro tipo de anotaciones a las clases como restricciones en las propiedades. Relacionado con el tema del atributo "tipo" de procesos y recursos, sería interesante definir en ARAS una lista para cada uno de ellos e introducir en la interfaz la capacidad procesar esa información.

7 ANEXOS

7.1 Relaciones de Procesos

7.1.1 Procesos – Recursos

Proceso	Recurso	Cantidad	Tiempo
Fabricación Alas	Operario	5	12
	Grúa	-	3
	Remachadora	-	4
	Remaches	100	-
Fabricación Fuselaje Inferior	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Transporte Alas	Grúa	-	1
	Operario	3	1
Ensamblaje Fus Inf-Ala	Operario	5	10
	Grúa	-	4
Fabricación Fuselaje Superior	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Ensamblaje Fus Inf-Ala-Fus Sup	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Fabricación Cabina	Operario	5	20
	Remachadora	-	10
	Remaches	100	-

Fabricación VTP	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Transporte Cabina	Grúa	-	1
	Operario	3	1
Transporte VTP	Grúa	-	1
	Operario	3	1
Ensamblaje Cuerpo-Cab-VTP	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Fabricación HTP	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Fabricación Tren	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Transporte HTP	Grúa	-	1
	Operario	3	1
Tranporte Tren	Grúa	-	1
	Operario	3	1
Ensamblaje Principal-HTP-Tren	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Fabricación Motor	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Fabricación Hélice	Operario	5	20
	Remachadora	-	10

	Remaches	100	-
Ensamblaje Motor-Hélice	Operario	5	20
	Remachadora	-	10
	Remaches	100	-
Transporte GM	Grúa	-	1
	Operario	3	1
Ensamblaje Principal-GM	Operario	5	20
	Remachadora	-	10
	Remaches	100	-

Tabla 7-1. Relación Procesos - Recursos

7.1.2 Procesos – Localización

Proceso	Localización
Fabricación Alas	Sección 2
Fabricación Fuselaje Inferior	Sección 1
Transporte Alas	-
Ensamblaje Fus Inf-Ala	Sección 1
Fabricación Fuselaje Superior	Sección 1
Ensamblaje Fus Inf-Ala-Fus Sup	Sección 1
Fabricación Cabina	Sección 3
Fabricación VTP	Sección 4
Transporte Cabina	-
Transporte VTP	-
Ensamblaje Cuerpo-Cab-VTP	Sección 1
Fabricación HTP	Sección 5
Fabricación Tren	Sección 6
Transporte HTP	-
Tranporte Tren	-
Ensamblaje Principal-HTP-Tren	Sección 1

Fabricación Motor	Sección 7
Fabricación Hélice	Sección 7
Ensamblaje Motor-Hélice	Sección 7
Transporte GM	-
Ensamblaje Principal-GM	Sección 1

Tabla 7-2. Relación Procesos - Localizaciones

7.1.3 Procesos – KPI

Proceso	KPI	Valor
Fabricación Alas	OEE	75
	OLE	80
Fabricación Fuselaje Inferior	OEE	81
	OLE	82
Transporte Alas	OEE	90
	OLE	85
Ensamblaje Fus Inf-Ala	OEE	89
	OLE	82
Fabricación Fuselaje Superior	OEE	83
	OLE	85
Ensamblaje Fus Inf-Ala-Fus Sup	OEE	84
	OLE	74
Fabricación Cabina	OEE	75
	OLE	80
Fabricación VTP	OEE	81
	OLE	82
Transporte Cabina	OEE	90
	OLE	85
Transporte VTP	OEE	89
	OLE	82

Ensamblaje Cuerpo-Cab-VTP	OEE	83
	OLE	85
Fabricación HTP	OEE	84
	OLE	74
Fabricación Tren	OEE	75
	OLE	80
Transporte HTP	OEE	81
	OLE	82
Tranporte Tren	OEE	90
	OLE	85
Ensamblaje Principal-HTP-Tren	OEE	89
	OLE	82
Fabricación Motor	OEE	83
	OLE	85
Fabricación Hélice	OEE	84
	OLE	74
Ensamblaje Motor-Hélice	OEE	75
	OLE	80
Transporte GM	OEE	81
	OLE	82
Ensamblaje Principal-GM	OEE	90
	OLE	85

Tabla 7-3. Relación Procesos - KPI

7.2 Instancias

Lo que se muestra a continuación es un ejemplo de cada isntancia en su formato XSD, no las instancias completas, pues sería demasiado extenso.

- Instancia 1

```xml
<PPR>
<Producto><id>1_TH</id><nombre>1</nombre><descripcion>Turbohelice</descripcion><revision>0</revision><estado>0</estado></Producto>
<PlanProceso><id>2</id></PlanProceso>
<Producto_PlanProceso><idproducto>1</idproducto><idplanproceso>2</idplanproceso></Producto_PlanProceso>

<!-- Partes basicas -->
<Parte><id>307026_HTP</id><nombre>307026</nombre><descripcion>FLAT TILE 1X1</descripcion><revision>0</revision><estado>0</estado></Parte>
<Parte><id>302324_HTP</id><nombre>302324</nombre><descripcion>PLATE 1X2</descripcion><revision>0</revision><estado>0</estado></Parte>
<Parte><id>243126_HTP</id><nombre>243126</nombre><descripcion>FLAT TILE 1X4</descripcion><revision>0</revision><estado>0</estado></Parte>
<Parte><id>302026_HTP</id><nombre>302026</nombre><descripcion>PLATE 2X4</descripcion><revision>0</revision><estado>0</estado></Parte>
<!--    Partees basicas        -->

<!-- Partees resultantes de procesos -->
<Parte><id>50000_Alas</id><nombre>50000</nombre><descripcion>Alas</descripcion><revision>0</revision><estado>0</estado></Parte>
<Parte><id>60000_Fus_Inf</id><nombre>60000</nombre><descripcion>Fuselaje Inferior</descripcion><revision>0</revision><estado>0</estado></Parte>
<!--Partees resultantes de procesos -->

<!-- Procesos  -->
<Proceso><id>3</id><nombre>Fabricación Alas</nombre><descripcion>Unión de las piezas lego que forman las alas</descripcion>
<leadtiempouso>12</leadtiempouso><revision>0</revision><estado>0</estado></Proceso>
<Proceso><id>4</id><nombre>Fabricación Fuselaje Inferior</nombre><descripcion>Unión de las piezas lego que forman el fuselaje inferior</descripcion>
<leadtiempouso>20</leadtiempouso><revision>0</revision><estado>0</estado></Proceso>
<!--     Procesos       -->

<!-- Recursos  -->
<Recurso><id>100</id><tipo>Operario</tipo><cantidad>93</cantidad><tiempouso>288</tiempouso></Recurso>
<Recurso><id>101</id><tipo>Grúa</tipo><cantidad>0</cantidad><tiempouso>12</tiempouso></Recurso>
<!-- Recursos  -->

<!-- KPI  -->
<KPI><id>201</id><tipo>OEE</tipo><valor>75</valor></KPI>
<KPI><id>202</id><tipo>OLE</tipo><valor>80</valor></KPI>
<!-- KPI  -->

<!-- Localizacion  -->
<Localizacion><localizacion>Stación 1</localizacion></Localizacion>
<Localizacion><localizacion>Stación 2</localizacion></Localizacion>
<!-- Localizacion  -->

<!-- PlanProcesos_Procesos  -->
<PlanProceso_Proceso><idplanproceso>2</idplanproceso><idproceso>3</idproceso></PlanProceso_Proceso>
<PlanProceso_Proceso><idplanproceso>2</idplanproceso><idproceso>4</idproceso></PlanProceso_Proceso>
<!-- PlanProcesos_Procesos -->

<!-- Procesos_Parte  -->
<Proceso_Consume><idproceso>3</idproceso><idparte>2412194_Alas</idparte></Proceso_Consume>
<Proceso_Obtiene><idproceso>3</idproceso><idparte>50000_Alas</idparte></Proceso_Obtiene>
<!-- Procesos_Parte  -->

<!-- Procesos_Recursos  -->
<Proceso_Recurso><idproceso>3</idproceso><idrecurso>100</idrecurso><cantidad>5</cantidad><tiempouso>12</tiempouso></Proceso_Recurso>
<Proceso_Recurso><idproceso>3</idproceso><idrecurso>101</idrecurso><cantidad>0</cantidad><tiempouso>3</tiempouso></Proceso_Recurso>
<!-- Procesos_Recursos  -->

<!-- Proceso_KPI  -->
<Proceso_KPI><idproceso>3</idproceso><idkpi>201</idkpi><tipo>OEE</tipo><valor>75</valor></Proceso_KPI>
<Proceso_KPI><idproceso>3</idproceso><idkpi>202</idkpi><tipo>OLE</tipo><valor>80</valor></Proceso_KPI>
<!-- Proceso_KPI  -->

<!-- Procesos_Localizacion  -->
<Proceso_Localizacion><idproceso>3</idproceso><localizacion>Sección 2</localizacion></Proceso_Localizacion>
<!-- Procesos_Localizacion  -->
</PPR>
```

Figura 7–1. Instancia 1 XML

- Instancia 2

```xml
<PPR>
<Product><instance_number>1_TH</instance_number><name>1</name><description>Turbohelice</description><revision>0</revision><state>0</state></Product>
<ProcessPlan><instance_number>2</instance_number></ProcessPlan>
<Product_ProcessPlan><instance_number_product>1</instance_number_product><instance_number_processplan>2</instance_number_processplan></Product_ProcessPlan>

<!-- Basic Parts -->
<Part><instance_number>307026_HTP</instance_number><name>307026</name><description>FLAT TILE 1X1</description><revision>0</revision><state>0</state></Part>
<Part><instance_number>302324_HTP</instance_number><name>302324</name><description>PLATE 1X2</description><revision>0</revision><state>0</state></Part>
<!--    Basic Parts        -->

<!-- Obtained Parts -->
<Part><instance_number>50000_Alas</instance_number><name>50000</name><description>Alas</description><revision>0</revision><state>0</state></Part>
<Part><instance_number>60000_Fus_Inf</instance_number><name>60000</name><description>Fuselaje Inferior</description><revision>0</revision><state>0</state></Part>
<!--Obtained Parts -->

<!-- Process  -->
<Process><instance_number>3</instance_number><name>Fabricación Alas</name><description>Unión de las piezas lego que forman las alas</description>
<leadtime>12</leadtime><revision>0</revision><state>0</state></Process>
<Process><instance_number>4</instance_number><name>Fabricación Fuselaje Inferior</name><description>Unión de las piezas lego que forman el fuselaje inferior</description>
<leadtime>20</leadtime><revision>0</revision><state>0</state></Process>
<!--     Process       -->

<!-- Resources  -->
<Resource><instance_number>100</instance_number><type>Operario</type><quantity>93</quantity><time>288</time></Resource>
<Resource><instance_number>101</instance_number><type>Grúa</type><quantity>0</quantity><time>12</time></Resource>
<!-- Recsources  -->

<!-- KPI  -->
<KPI><instance_number>201</instance_number><type>OEE</type><value>75</value></KPI>
<KPI><instance_number>202</instance_number><type>OLE</type><value>80</value></KPI>
<!-- KPI  -->

<!-- Location  -->
<Location><location>Stación 1</location></Location>
<Location><location>Stación 2</location></Location>
<!-- Location  -->

<!-- ProcessPlan_Process  -->
<ProcessPlan_Process><instance_number_processplan>2</instance_number_processplan><instance_number_process>3</instance_number_process></ProcessPlan_Process>
<ProcessPlan_Process><instance_number_processplan>2</instance_number_processplan><instance_number_process>4</instance_number_process></ProcessPlan_Process>
<!-- ProcessPlan_Process -->

<!-- Process_Part  -->
<Process_Consumes><instance_number_process>3</instance_number_process><instance_number_part>2412194_Alas</instance_number_part></Process_Consumes>
<Process_Obtains><instance_number_process>3</instance_number_process><instance_number_part>50000_Alas</instance_number_part></Process_Obtains>
<!-- Process_Part  -->

<!-- Process_Resources  -->
<Process_Resource><instance_number_process>3</instance_number_process><instance_number_resource>100</instance_number_resource><quantity>5</quantity><time>12</time></Process_Resource>
<Process_Resource><instance_number_process>3</instance_number_process><instance_number_resource>101</instance_number_resource><quantity>0</quantity><time>3</time></Process_Resource>
<!-- Process_Resources  -->

<!-- Process_KPI  -->
<Process_KPI><instance_number_process>3</instance_number_process><instance_number_kpi>201</instance_number_kpi><type>OEE</type><value>75</value></Process_KPI>
<Process_KPI><instance_number_process>3</instance_number_process><instance_number_kpi>202</instance_number_kpi><type>OLE</type><value>80</value></Process_KPI>
<!-- Process_KPI  -->

<!-- Process_Location  -->
<Process_Location><instance_number_process>3</instance_number_process><location>Sección 2</location></Process_Location>
<!-- Process_Location  -->
</PPR>
```

Figura 7–2. Instancia 2 XML

REFERENCIAS

[1] I. U. Ltd. [En línea]. Available: https://www.nomagic.com/mbse/images/casestudies/Why_MBSE.pdf.

[2] «Ingeniería Industrial Online,» [En línea]. Available: https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/procesos-industriales/ingenieria-concurrente/.

[3] R. Delgado, «Revista Digital Inesem,» 2017. [En línea]. Available: https://revistadigital.inesem.es/gestion-integrada/ingenieria-concurrente/.

[4] I. Insituto de Análisis de la Defensa, «Informe R-338,» EE.UU, 1996.

[5] «MBSE.WORKS,» [En línea]. Available: http://mbse.works/.

[6] A. Tarski, Contributions to theory of models I, II, III, vol. 57, 1954, pp. 572-581, 582-588.

[7] J. A. Estefan, «Survey of Model-Based Systems Engineering (MBSE),» 2008. [En línea]. Available: http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf.

[8] D. M. Charles E. DIckerson, «A brief History of Models and Model-Based Systems Engineering and the Case for Relational Orientation,» *IEEE Systems Journal,* vol. 7, nº 4, 2013.

[9] [En línea]. Available: http://www.omgwiki.org/MBSE/doku.php.

[10] «OMG SysML,» [En línea]. Available: http://www.omgsysml.org/what-is-sysml.htm.

[11] R. Studer, V. Benjamis y D. Fensel, «Knowledge Engineering: Principles and Methods,» *Data & Knowledge Engineering 25(1-2),* pp. 161-197, 1998.

[12] T. Gruber, «A translation approach to portable ontology specification,» *Knowledge Acquisition 5(2),* pp. 199-220, 1993.

[13] D. L. McGuinness y F. v. Harmelen, «OWL Web Ontology Language Overview,» 2004. [En línea]. Available: https://www.w3.org/TR/owl-features/. [Último acceso: 05 2018].

[14] A. Farquhar, R. Fikes y J. Rice, *International Journal of Human-Computer Studies,* nº 46, pp. 707-727, 1997.

[15] O. F.-L. M. &. G.-P. A. Corcho, «Methodologies, tools and languages for building ontologies. Where is their meeting point,» *Data & Knowlede Engineerinf,* nº 49, pp. 41-64, 2002.

[16] R. Khondoker y P. Mueller, «Comparing Ontology Development Tools Based on an Online Survey,» de *WCE*, Londres, 2010.

[17] R. Mizoguchi y k. Kozaki, «Ontology Engineering Environments,» de *Handbook on Ontologies*, Berlin,

Springer-Verlag, 2009, pp. 315-335.

[18] D. Gasevic, D. Djuric y V. Devedzic, «Model Driven Architecture and Ontology Development,» de *Springle-Verlag*, Berlin, 2006.

[19] «Sparx Systems,» [En línea]. Available: https://sparxsystems.com/products/ea/.

[20] I. O. f. S. (ISO), «Industrial automation systems and integration - Product data representation exchange - Part 1: Overview and fundamental principles,» de *ISO 1030-1*, 1994.

[21] R. Laguionie, M. Rauch y J. Hascoet, «Toolpaths programming in an intelligent STEP-NC manufacturing context,» J. Mach. Eng, 2008, pp. 33-43, vol. 8.

[22] I. O. f. S. (ISO), «Industrial automation systems and integration - Product data representation exchange - Part 11: Description methods: The EXPRESS language reference manual,» de *ISO 10303-11*, 1994.

[23] r. Barbau, S. Krima, A. Narayanan, X. Fiorentini, S. Foufou y R. Sriram, «OntoSTEP: enriching product model data using ontologies.,» *Comput-Aided Design,* pp. 44(6):575-90, 2012.

[24] T. Hanis y D. Noller, «IBM Developer,» IBM, 2012 3 30. [En línea]. Available: https://www.ibm.com/developerworks/library/x-ind-semanticmodels/index.html.