UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

INTEGRACIÓN DE TÉCNICAS AVANZADAS DE WEB SEMÁNTICA PARA DISEÑO Y VISUALIZACIÓN DE ESTRUCTURAS DE CONOCIMIENTO

Integration of Advanced Semantic Web Technologies for Design and Visualization of Knowledge Structures

Autor: Luis Miguel Álvarez Jorquera

Director: Rodrigo Martínez Béjar

Febrero 2010

ÍNDICE

Abstract

Actualmente, uno de los principales retos de la investigación relacionada con la Web Semántica es el de la organización de la información de modo que pueda ser fácilmente comprendida tanto por humanos como por las aplicaciones software. Para ello, actualmente se utilizan ontologías, que han sido consideradas una tecnología esencial para el logro de la Web Semántica [1], puesto que permiten que los profesionales reutilicen y compartan el conocimiento contenido en ellas. De este modo el mismo contenido ontológico puede ser usado para diferentes tareas y aplicaciones [2]. Por el contrario, la construcción de ontologías de calidad es una tarea que no es sencilla.

Los problemas principales de la *ingeniería ontológica* son el diseño, la especificación, la formalización y el mantenimiento de ontologías de buena calidad [3]. El gran vacío en este campo es la ausencia de una metodología estándar. Entre los diferentes tipos de ontologías existentes, las taxonomías han recibido la mayor atención debido a su proximidad al modo de razonamiento de los seres humanos cuando clasifican. Por tanto, la construcción de buenas taxonomías ha sido un tema importante y recurrente de investigación en los últimos años [4,5]. De ahí la importancia que hemos dado en este trabajo a las taxonomías en el análisis ontológico de este proyecto.

Otro aspecto importante para el progreso científico en la investigación ontológica es la comprensión, y en último término, la evaluación, aspectos ambos de la calidad ontológica. Las ontologías necesitan ofrecer una alta calidad para que los algoritmos de razonamiento puedan producir resultados provechosos. Su empleo extendido conduce a una necesidad creciente de metodologías dependientes del dominio y directrices para la ingeniería ontológica y la evaluación. Un campo importante respecto a la evaluación de ontologías es la formalización de éstas, debido a que gracias a dicho proceso pueden evidenciarse ambigüedades implícitas en la ontología. Sin embargo, no hay muchas metodologías centradas en especificar formalmente la estructura y el contenido ontológico [6]. Es posible concluir, por tanto, que hay una necesidad de proporcionar más formalización al contenido y la estructura ontológica y éste es otro objetivo del trabajo [5].

La decisión de si un término debería subsumir a otro es una de las decisiones ontológicas más importantes que un modelador debe tener en cuenta en la construcción de una ontología, y el suministro de una base formal para evaluar estas decisiones ha demostrado un paso importante en la práctica de modelado conceptual.

En este trabajo, se presenta una metodología para la **creación de ontologías taxonómicas haciendo uso de una serie de métricas y la posibilidad de realizar consultas, de varios tipos, sobre dichas ontologías, y sobre otras ontologías remotas accesibles vía http**. Además el sistema incorpora una API para la visualización de cualquiera de las ontologías descritas anteriormente en Protégé que es un producto para editar ontologías que se explica más adelante.

Una de las aplicaciones reales que ha tenido este proyecto final de carrera es el diseño de una base de datos, mediante el diseño de una estructura de conocimiento, para una aplicación del sector agrario desarrollada en la asociación mediterránea de organizaciones de productores agrarios llamada “Sistema de Información Económica y Contable de Explotaciones Agrarias” (SIECAGRI). Para el diseño de esta estructura se partió del concepto principal (el concepto raíz) que es “Parcela Agrícola” a partir de la cual se le van asociando especies de cultivo, variedades, maquinaria, riegos, productos fertilizantes, fitosanitarios, etc.

Esta aplicación software puede aplicarse a cualquier dominio de conocimiento, a cualquier estructura de conocimiento, de cualquier tipo de conocimiento que pueda existir. Sin embargo uno de los dominios en que se basará este trabajo será la biología molecular y más concretamente se ha trabajado con la estructura “The Gene Ontology”, ya que constituye el dominio de aplicación del proyecto de investigación donde se enmarca este proyecto fin de carrera y uno de los sectores donde se están aplicando más estas tecnologías.

1. Introducción y Referencias Históricas

1.1 Bioinformática

Bioinformática es una disciplina científica emergente que utiliza tecnología de la información para organizar, analizar y distribuir información biológica con la finalidad de responder preguntas complejas en biología [7]. Bioinformática es un área de investigación multidisciplinaria, la cual puede ser ampliamente definida como la interfase entre dos ciencias: Biología y Computación y esta impulsada por la incógnita del genoma humano y la promesa de una nueva era en la cual la investigación genómica puede ayudar dramáticamente a mejorar la condición y calidad de vida humana.

Avances en la detección y tratamiento de enfermedades y la producción de alimentos genéticamente modificados son entre otros ejemplos de los beneficios mencionados más frecuentemente. Involucra la solución de problemas complejos usando herramientas de sistemas y computación. También incluye la colección, organización, almacenamiento y recuperación de la información biológica que se encuentra en base de datos.

Según la definición del Centro Nacional para la Información Biotecnológica "National Center for Biotechnology Information" [8]:

"Bioinformática es un campo de la ciencia en el cual confluyen varias disciplinas tales como: biología, computación y tecnología de la información. El fin último de este campo es facilitar el descubrimiento de nuevas ideas biológicas así como crear perspectivas globales a partir de las cuales se puedan discernir principios unificadores en biología. Al comienzo de la "revolución genómica", el concepto de bioinformática se refería sólo a la creación y mantenimiento de base de datos donde se almacena información biológica, tales como secuencias de nucleótidos y aminoácidos. El desarrollo de este tipo de base de datos no solamente significaba el diseño de la misma sino también el desarrollo de interfaces complejas donde los investigadores pudieran acceder los datos existentes y suministrar o revisar datos.

Luego toda esa información debía ser combinada para formar una idea lógica de las actividades celulares normales, de tal manera que los investigadores pudieran estudiar cómo estas actividades se veían alteradas en estados de una enfermedad. De allí viene el surgimiento del campo de la bioinformática y ahora el campo más popular es el análisis e interpretación de varios tipos de datos, incluyendo secuencias de nucleótidos y aminoácidos, dominios de proteínas y estructura de proteínas. El proceso de analizar e interpretar los datos es conocido como biocomputación. Dentro de la bioinformática y la biocomputación existen otras subdisciplinas importantes: El desarrollo e implementación de herramientas que permitan el acceso, uso y manejo de varios tipos de información El desarrollo de nuevos algoritmos (fórmulas matemáticas) y estadísticos con los cuales se pueda relacionar partes de un conjunto enorme de datos, como por ejemplo métodos para localizar un gen dentro de una secuencia, predecir estructura o función de proteínas y poder agrupar secuencias de proteínas en familias relacionadas."

Las tecnologías de la información jugarán un papel fundamental en la aplicación de los desarrollos tecnológicos en el campo de la genética a la práctica médica como refleja la presencia de la Bioinformática médica y la Telemedicina dentro de las principales líneas en patología molecular. La aplicación de los conocimientos en genética molecular y las nuevas tecnologías son necesarias para el mantenimiento de la competitividad del sistema sanitario no sólo paliativo sino preventivo. La identificación de las causas moleculares de las enfermedades junto con el desarrollo de la industria biotecnológica en general y de la farmacéutica en particular permitirán el desarrollo de mejores métodos de diagnóstico, la identificación de dianas terapéuticas y desarrollo de fármacos personalizados y una mejor medicina preventiva.

Se debe distinguir entre tres acepciones en las que se unen la biología y la informática, pero con objetivos y metodologías bien diferenciadas:

- Bioinformática o Biología Molecular Computacional: investigación y desarrollo de la infraestructura y sistemas de información y comunicaciones que requiere la biología molecular y la genética (Redes y bases de datos para el genoma, microarrays). En una parte de esta rama es en la que se basará este trabajo.

- Biología Computacional: computación que se aplica al entendimiento de cuestiones biológicas básicas, no necesariamente en el nivel molecular, mediante la modelización y simulación. (Ecosistemas, modelos fisiológicos).

- Biocomputación: desarrollo y utilización de sistemas computacionales basados en modelos y materiales biológicos. (Biochips, biosensores, computación basada en ADN, redes de neuronas, algoritmos genéticos).

1.2 Tecnologías a aplicar

1.2.1 La Web Semántica

Actualmente, la Web es un espacio de intercambio de información diseñado para los humanos que, a pesar de las indudables ventajas que ofrece a la hora de buscar información, carece de una forma precisa y eficiente de encontrarla y de poder realizar deducciones. Se pueden identificar las mayores carencias de la Web actual como la sobrecarga de información y la heterogeneidad de las fuentes de información que dificultan una mejor utilización. La idea que ha surgido en los últimos años es que los datos puedan ser “comprendidos” por los ordenadores sin necesidad de la intervención humana. Se trata de convertir la información en conocimiento a través de la representación de los datos en algún lenguaje formal que haga posible que las máquinas razonen sobre el contenido de los recursos Web.

Esto es lo que se conoce como Web Semántica. En palabras de su creador, Tim Bernes Lee [1]: "*El primer paso es colocar los datos en la Web de un modo en que las máquinas puedan entenderlos naturalmente o convertirlos a esa forma. Esto crea lo que yo llamo una Web semántica: una red de datos que pueden ser procesados directa o indirectamente por máquinas*."

Todo ello contribuirá a que la Web tenga la capacidad de responder con mayor éxito a las demandas de los usuarios. Debemos preguntarnos cómo será posible pasar de un medio pasivo, como es la Web tradicional, a una Web más inteligente que tenga incorporado conocimiento, en la que se consiga una comunicación efectiva entre ordenadores.

Las cuatro características básicas necesarias para que la Web Semántica sea factible según Tim Bernes Lee son [1]:

- Expresar el significado. La Web Semántica debe dotar de semántica la información de las páginas Web, haciendo posible que agentes software viajen de una página a otra realizando tareas sofisticadas para los usuarios.

- Acceso a representaciones del conocimiento. La Web Semántica deber resolver las limitaciones de la representación de conocimiento tradicionales creando lenguajes tan expresivos como para que pueda llevarse a cabo un razonamiento.

- Ontologías. La Web Semántica debe añadir conocimiento formalizado y datos que sean procesados por ordenadores. Las ontologías representan el conocimiento y pueden ser compartidas, por lo que incrementan en eficiencia e interoperabilidad. Además, debe ser capaz de realizar el mapeado (mapping) para asociar términos de ontologías diferentes.

- Agentes. "El poder real de la Web Semántica se conocerá cuando haya gente que cree numerosos programas capaces de capturar el contenido de la Web desde diversas fuentes, procesar la información e intercambiar los resultados con otros programas. La eficacia de tales agentes software crecerá exponencialmente a medida que estén disponibles más contenidos en la Web, legibles por ordenador, y más servicios automatizados (incluyendo otros agentes)". Los "agentes inteligentes" artificiales permitirían ofrecer servicios individuales muy personalizados, que ayuden a los usuarios en una amplia gama de actividades, y servicios orientados a grupos que faciliten las interacciones sociales.

La Web Actual y la Web Semántica.

La web actual consiste esencialmente en un conjunto enorme de páginas que contienen texto no estructurado, es decir, texto cuyo contenido no nos hemos preocupado por caracterizar. Básicamente nos hemos limitado a reseñar la manera en que debe visualizarse dicho contenido, como lo demuestra la naturaleza de las etiquetas HTML. Esta simplicidad ha favorecido, sin duda, el éxito de la web actual y justifica su enorme crecimiento en número de páginas y usuarios, pero al tiempo acarrea problemas y dificultades a la hora de manejar y recuperar tal cantidad ingente de información.

Los seres humanos somos incapaces de controlar la información que en un momento dado puede sernos de utilidad en relación a una necesidad informativa entre los millones de páginas existentes en la web, máxime cuando los cambios en la misma se suceden a un ritmo vertiginoso. De hecho, se estima que un 40% de la red se modifica mensualmente. En tales circunstancias, hemos ideado buscadores que nos ayudan a decidir qué páginas pueden incluir información relevante ante un problema cualquiera. Pero dado que la información textual de los sitios web no está estructurada, en cuanto que no está descrita ni caracterizada de alguna forma, los algoritmos de los motores de búsqueda únicamente pueden basarse en la aparición de las palabras consideradas aisladamente.

Ello provoca, sin duda, falta de precisión y exhaustividad en los resultados obtenidos. Falta de precisión por cuanto en los resultados que nos presenta un buscador se hallan páginas que no tienen relación alguna con nuestra necesidad informativa. Eso sucede, por ejemplo, cuando las palabras poseen varios significados. Si consultamos por la palabra banco obtendremos páginas relativas a entidades bancarias, pero también a un tipo de asiento. De igual forma, la falta de exhaustividad puede venir provocada, entre otros motivos, por la utilización de un sinónimo en una página en lugar de la palabra empleada en la consulta. En tal caso, la página no será recuperada pues no contiene estrictamente la palabra introducida en la búsqueda.

Además, los buscadores proporcionan enlaces a documentos que pueden ser útiles para el usuario, pero no son capaces de proporcionar la respuesta concreta que busca en muchas ocasiones. Si una persona busca los coches más baratos entre los concesionarios de una zona geográfica concreta, hoy día el usuario debe ocupar muchas horas comparando la información de los distintos concesionarios que un buscador le ha facilitado previamente.

Otro problema de la web actual consiste en la falta de fiabilidad de las fuentes. El usuario no tiene elementos de juicio sobre la veracidad y confiabilidad de los datos presentes en los sitios web recuperados.

La evolución de la web diseñada por Tim Berners-Lee trata de solucionar los problemas planteados en los párrafos anteriores. Imaginemos por un momento una web donde el contenido de las páginas está caracterizado y descrito de tal manera que sea capaz de discernir los distintos significados de las palabras, pueda deducir la existencia de relaciones de sinonimia entre palabras en cierto contexto temático, de manera que sea capaz de recuperar páginas útiles en relación a la necesidad informativa del usuario aunque en ellas no aparezcan las palabras introducidas expresamente en la consulta, o que fuese capaz de comparar datos e información procedentes de varias fuentes,

efectuar inferencias o deducciones lógicas a partir de ellos para mostrarnos directamente la información que buscábamos (el concesionario cercano con los coches más baratos, por ejemplo). Incluso que fuese capaz de emitir juicios sobre la fiabilidad de los datos presentes en las diversas páginas y considerar en la respuesta exclusivamente los más veraces, desechando los menos confiables.

Tim Berners-Lee ha denominado web semántica a la web donde las aplicaciones serán capaces de efectuar un procesamiento de la información mucho más profundo. Esta web estará caracterizada por programas capaces de "comprender" el contenido de las páginas web, y por tanto, de relacionar la información contenida en páginas hoy aisladas, de procesarla, de discriminar la más fiable en un momento dado, e incluso de deducir o inferir información no registrada previamente, tomando decisiones con un cierto grado de autonomía.

Para que estas aplicaciones y servicios más "inteligentes" sean posibles es necesario que la información de las páginas web esté estructurada, esto es, perfectamente descrita y clasificada de manera que su significado exacto esté al alcance de las máquinas. De esta manera los ordenadores podrán manipular y procesar la información adecuadamente. De ahí la denominación de Web Semántica.

La manera que se ha ideado para codificar los significados de la información contenida en las páginas web consiste en el empleo de etiquetas que especifiquen el valor semántico o la interpretación correcta de los contenidos. Así, un número puede indicar, según las circunstancias, un precio, un año o una longitud. Su significado preciso en cada caso se especificará mediante la presencia de una etiqueta.

El marcado y anotación de los contenidos de la web debe realizarse siguiendo unas reglas y formatos comunes, pues de lo contrario sería imposible la manipulación efectiva de la información por parte de los ordenadores. En primer lugar, un marcado consistente implica la estructuración previa del dominio que se representa, detallando las entidades principales que lo componen, su jerarquía y la naturaleza de las relaciones existentes entre ellas. En segundo lugar, debe cuidarse que todos los usuarios empleemos formatos compatibles, pues si coexisten varios conjuntos de etiquetas y no se procura un método para garantizar su utilización conjunta, todos los esfuerzos serían inútiles.

El cumplimiento de ciertas normas necesarias para desarrollar de manera coherente el etiquetado de los contenidos web supone la creación de ontologías sobre el dominio o área de conocimiento que deseamos representar semánticamente. En consecuencia, las ontologías son el medio principal para lograr el objetivo de la web semántica, al facilitar la definición formal de las entidades y conceptos presentes en los diferentes dominios, la jerarquía que les sustenta y las diferentes relaciones que los unen entre sí. De esta manera garantizamos una representación formal legible por las máquinas, basado en un lenguaje común -XML- que puede ser compartido y utilizado por cualquier sistema de manera automática.

No menos importante que los retos tecnológicos y de formalismos se plantea el reto de la explotación y uso de la web semántica. Haciendo un símil con la web actual, que presenció su auge en cuanto se perfilaron nuevos modelos de negocio, se esbozan aquí algunas posibilidades o visiones sobre los tipos de aplicaciones en la web semántica.

La tecnología de la web semántica ofrece la posibilidad de construir contenido de manera formal y completa de acuerdo a modelos semánticos consensuados. La existencia de estos modelos permite que las funcionalidades ofrecidas por estos sistemas abarquen, entre otras, las siguientes aplicaciones:

- **Recuperación de información mediante buscadores semánticos**: las búsquedas semánticas, al contrario que las tradicionales -basadas en palabras clave-, trabajan con el significado de las palabras de acuerdo al modelo subyacente asegurando la precisión del 100% en las búsquedas. El resultado presentado al usuario pasa a ser la información solicitada en forma de conceptos del modelo, en lugar de los documentos posiblemente relacionados, tal como hacen los buscadores actuales.

- **Publicación de la información de acuerdo al modelo**. La navegación y la presentación de la información se podrá hacer de acuerdo a su contenido, de manera que el usuario puede visualizar los conceptos del modelo y consultar los conceptos relacionados independientemente de los documentos presentes en el sistema.

- La presencia del modelo permite la incorporación de **Interfaces inteligentes** como son los basados en lenguaje natural. La posibilidad de formular consultas en un lenguaje cercano al natural asegura la usabilidad del sistema final.

- **Sistema de inferencia y compleción de información**. En base a los axiomas de los modelos de la web semántica es posible validar y aumentar la información mediante sistemas de inferencia automáticos.

- **Intercambio de información a formatos de aplicaciones específicas**. La posibilidad de traducir la información a formatos de otras aplicaciones, como pueden ser aplicaciones educativas, permite aumentar la rentabilidad de la codificación de la misma. Actualmente el gasto de las empresas en hacer compatibles a sistemas heterogéneos supone un 30% del gasto de toda la industria de tecnologías de la información.

1.2.2 Tecnologías de Representación de Conocimiento (Ontologías).

Según la Real Academia Española de la Lengua el término "Ontología" es "*Parte de la metafísica que trata del ser en general y de sus propiedades trascendentales*". La palabra "Ontología" proviene del griego οντος, genitivo del participio del verbo εἰμί, ser, estar y λόγος, (ciencia, estudio, teoría). Es una disciplina filosófica identificada con la Metafísica considerada como la más importante que estudia lo que es en tanto que es y existe como *sub-stantia* de los fenómenos. Por ello la metafísica es muchas veces, más erróneamente, estimada como ontología, *teoría del ser*, es decir, el estudio de todo lo que es: qué es, cómo es y cómo es posible. En definitiva ontología se puede definir como: "*Representación Formal de un Conjunto de Conceptos de un Dominio y de las Relaciones que existen entre ellos*".

En la actualidad, han sido muy utilizadas para representar conocimiento en distintos tipos de dominios [9]. Las ontologías permiten que el conocimiento que éstas contienen pueda reutilizarse y compartirse, por lo que su uso conlleva una reducción del esfuerzo necesario para implementar sistemas expertos. Las ontologías se desarrollaron en el área de Inteligencia Artificial para facilitar la compartición y la reusabilidad. Desde los años 90 del siglo XX, ha sido un tema puntero de investigación, y han sido objeto de investigación en varias comunidades científicas, como la de Ingeniería del Conocimiento, Procesamiento de Lenguaje Natural o Representación de Conocimiento. El motivo de la creciente popularidad de la investigación en ontologías es principalmente lo que permite su utilización: una comprensión compartida y común de un dominio que puede ser comunicado entre personas y aplicaciones informáticas. Como tales, el uso de las ontologías ofrece una oportunidad de mejorar las posibilidades de realizar cualquier tarea relacionada con la gestión de información y conocimiento.

Por otra parte, existe el problema de la representación de las ontologías en un lenguaje común, aunque hay una propuesta de estandarización de lenguaje ontológico llamado OWL, especificado por un grupo de trabajo (Web Ontology) del consorcio W3C [10], que ayudaría a superar los impedimentos actuales para la construcción cooperativa de ontologías entre diferentes plataformas de construcción ontológica que usen diferentes modelos expresados en OWL. Sin embargo, la mayor limitación actual de los trabajos en ingeniería ontológica es que están centrados en la taxonomía, dejando a un lado otras (no menos importantes) relaciones ontológicas formales para describir contenidos (y, por tanto, realizar consultas) en portales de conocimiento. Sin embargo, algunos trabajos recientes apuntan la necesidad de considerar, entre otras, estructuras temporales para modelar procesos de servicio.

Las ontologías son una de las tecnologías que permiten compartir el conocimiento en el dominio en términos estáticos. Esta tecnología ha sido utilizada ya para la representación del conocimiento en diferentes dominios, como puede ser el caso de dominios clínicos [9,11], y las memorias organizacionales [12]. Las ontologías, al poseer las propiedades de reusabilidad y compartición permiten reducir el esfuerzo necesario para realizar el modelado ontológico de dominios complejos. Sin embargo, la construcción de ontologías es un proceso complejo, por lo que no existen en la actualidad muchos métodos disponibles. Uno de éstos se basa en un conjunto de funciones para cuantificar el conocimiento. Por medio de estas funciones, se pueden obtener vocabularios ontológicos así como restricciones que deben ser satisfechas por la estructura de la ontología [13]. La tecnología en cuestión se ha aplicado a diferentes

dominios médicos [14] y ya ha sido utilizada para gestionar conocimiento en dichos dominios por el grupo al que pertenece el director de la solicitante [15].

Cada ontología es un sistema de conceptos y las relaciones entre ellos, en el cual todos los conceptos están definidos e interpretados de forma declarativa. El sistema define el vocabulario de un dominio de problema y un conjunto de restricciones respecto a cómo pueden ser combinados para modelar el dominio. En entornos distribuidos, agentes utilizan ontologías para establecer comunicación a nivel de conocimiento utilizando lenguajes específicos y protocolos [16].

El Termino "Ontología".

Aunque existen muy diversas definiciones del concepto de ontología, una de las más ampliamente aceptadas es la siguiente de Thomas Gruber:

"Una Ontología es una especificación formal y explícita de una conceptualización compartida"

El término "conceptualización" implica que toda ontología desarrolla un modelo abstracto del dominio o fenómeno del mundo que representa. Dicho modelo abstracto se basa esencialmente en el empleo de conceptos, atributos, valores y relaciones.

Con "especificación explícita" se quiere expresar que una ontología supone la descripción y representación de un dominio concreto mediante conceptos, atributos, valores, relaciones, funciones, etc., definidas explícitamente. Las máquinas no pueden dar nada por supuesto o por obvio, y todo conocimiento -por básico que parezca, mientras sea necesario- debe ser explícitamente representado.

El término "formal" alude al hecho de que cualquier representación (concepto, atributo, valor, etc.) ha de ser expresada en una ontología mediante un formalismo siempre idéntico, de manera que pueda ser reutilizada y leída por cualquier máquina independientemente del lugar o de la plataforma o idioma del sistema que lo emplee.

Quizá el término más restrictivo e importante de los que figuran en la definición sea el de "compartida". En efecto, una ontología lo es cuando dicha conceptualización y su representación formal y explícita ha sido favorablemente acogida por todos los usuarios de la misma. Ello permite por una parte distinguir claramente las ontologías de las bases de datos (en las que también puede hablarse de conceptualización y especificación formal y explícita mediante conceptos, atributos y valores, pero en la que el creador no tiene que lograr el consenso de nadie). Sin embargo, al mismo tiempo plantea una gran dificultad, pues en la práctica es imposible o casi imposible conseguir el consenso de todos los involucrados en un dominio específico (piénsese, por ejemplo, en una ontología sobre sanidad: ¿quién pone de acuerdo a médicos, pacientes, profesores, alumnos, gestores, etc., en la terminología, valores y relaciones en el dominio de la medicina?). De ahí que, en la práctica, se considere imposible desarrollar una ontología de carácter genérico o global, y sin embargo, se desarrollen ontologías en ámbitos mucho más restringidos, porque alcanzar aquí el consenso es factible (un banco para desarrollar servicios online para sus clientes, una empresa que desea una ontología sobre el conocimiento generado internamente por ella, etc.). Cuanto más genérico es el

ámbito, en mayor medida el proceso hasta alcanzar el consenso se produce mediante una guerra de estándares inicial (varios organismos lanzan sus ontologías, esperando que cada una de ellas alcance el consenso de los demás), de las que surgen con el tiempo 2 ó 3 estándares de facto por área o sector, normalizándose finalmente hasta alcanzar una variante con el consenso de todos.

La parte formal del contenido está sujeta por el uso de ontologías, que otorgan la capacidad de comprensión a las aplicaciones o a los agentes inteligentes. La disponibilidad, facilidad de gestión y divulgación de éstas es otro de los retos que se plantea. Las ontologías, consideradas como repositorios formales de conocimiento, siguen un ciclo de vida que modela desde su construcción, refinamiento, modificaciones, uso o explotación hasta su retiro. Alrededor de esta figura se sitúa el reto de la disponibilidad de las ontologías, que incluye la necesidad de metodologías de construcción, herramientas que las soporten, métodos de evaluación, comprobación y metodologías de evolución como gestión de cambios y de versiones, entre otros. Las ontologías no son formalismos cerrados y están sujetos a procesos evolutivos. Es por eso que metodologías y herramientas que soporten estos procesos se hacen esenciales. El proceso de desarrollo de ontologías debe tener el apoyo necesario tanto desde el punto de vista metodológico como de herramientas que lo faciliten.

Hoy día la principal ventaja que aportan las ontologías, desde el punto de vista de las bases de datos y de la recuperación de información, tiene que ver con el empleo simultáneo de bases de datos de muy distinta naturaleza, características y formato a las que poder interrogar simultáneamente para recuperar la información buscada. Estas bases de datos pueden ser de contenido tan dispar como documentación textual poco estructurada (la clásica que forma parte de los buscadores en Internet), documentación fotográfica, documentación geográfica, museográfica, urbanística, espacios naturales, etc.

Otra de las grandes ventajas de las ontologías es la posibilidad de utilizar la información preexistente en dichas bases de datos a través de la propia ontología sin tener que renunciar a la base de datos original de partida, de manera que pueden convivir y seguir empleándose simultáneamente bases de datos iniciales y ontología. Tan solo debemos tener presente la necesidad de actualizar la ontología con los nuevos datos que vayan añadiéndose a las bases de datos. Tampoco es necesario introducir de nuevo manualmente -aunque podría hacerse- las bases de datos en la nueva estructura que hemos desarrollado con la ontología. Se puede automatizar el proceso de volcado de la información mediante el desarrollo informático de programas que transformen el formato de la base de datos inicial en el formato y lugar adecuado en la nueva estructura ontológica. Lógicamente, cuanto más estructurada se halle la información inicial, más sencilla es la transformación correspondiente.

Una de las mayores desventajas, en cambio, radica en la imposibilidad de utilización directa de las ontologías previamente desarrolladas. Es preciso crear programas de interrogación de la ontología, pues las herramientas existentes apenas permiten la visualización de los datos previamente introducidos. Eso exige la participación de especialistas en programación dentro de los equipos de desarrollo de ontologías si deseamos utilizarla posteriormente.

Otra de sus grandes desventajas surge al tratar de sacar el máximo partido de las amplísimas posibilidades de recuperación que proporcionan las ontologías, lo que se consigue mediante el desarrollo de buscadores denominados semánticos. Estos buscadores semánticos involucran la programación de complejos algoritmos que echan mano de herramientas de procesamiento de lenguaje natural poco desarrollados aún. En consecuencia, las ontologías posibilitarían una mejora sustancial en la precisión de la recuperación gracias a la posibilidad real de "entender" los términos y conceptos presentes en las preguntas, pero para ello dependen de complejos algoritmos de comprensión del lenguaje natural que no han alcanzado todavía un desarrollo suficiente. En resumen, suponen una herramienta con grandes posibilidades que, sin embargo, dependen para desarrollar todo su potencial de avances en otras áreas ajenas como la Inteligencia Artificial o el Procesamiento del Lenguaje Natural.

1.2.3 Taxonomías

Consideremos los elementos que constituyen una ontología. Las ontologías proporcionan un vocabulario común de un área y definen el significado de los términos y las relaciones entre ellos. El conocimiento en ontologías es, a menudo, especificado a través de cinco tipos de componentes: clases, relaciones, funciones, axiomas e instancias. Estos elementos deben ser formalizados para obtener un modelo de ontología formal y orientado al mundo real. Actualmente, la mayoría de las herramientas de construcción de ontologías sólo trabajan con la estructura taxonómica ontológica, esto es, con taxonomías. Desde un punto de vista ontológico, una taxonomía es una organización ontológica basada en una relación parcial ordenada llamada IS-A, a través de la cual las entidades son agrupadas en / o incluidas en una clase de nivel más alto. En general, las taxonomías han sido importantes para modelar los esquemas de bases de datos, los sistemas basados en el conocimiento y los vocabularios semánticos. Debido a la importancia y el grado de uso de esta relación semántica se ha prestado una especial atención a esta formalización. En particular, autores tales como Guarino y Welty han hecho un análisis en profundidad de las taxonomías y sus propiedades, persiguiendo definir una metodología a través de la definición de buenas taxonomías [4].

Según Kremer [17], una buena taxonomía debe poseer las siguientes propiedades:

- Adecuación descriptiva: la taxonomía debe clasificar la totalidad de los objetos del dominio que se intenta modelar (este criterio asume una metodología de construcción *bottom-up*, en términos de nivel de abstracción, p.e., desde objetos a categorías de dominio).

- Simplicidad: la selección de nombres para cada elemento del dominio, así como su clasificación debería ser una tarea sencilla.

- Capacidad predictiva: ésta es la propiedad más deseable para buenas taxonomías. Los objetos con clasificaciones similares deberías poseer propiedades similares.

En [5] las propiedades y el tipo de clases-conceptos que pueden aparecer en las taxonomías han sido definidos dentro de un corpus metodológico. En esta aproximación, las taxonomías deben tener las siguientes propiedades básicas: asimetría, transitividad e irreflexibilidad.

También se introducen otras propiedades taxonómicas, relativas a los atributos de los conceptos pertenecientes a una taxonomía, es decir, redefinición y múltiple herencia. Además, los autores (Guarino y Welty) definen otras condiciones basadas en términos filosóficos, relativos a la taxonomía, tales como identidad, unidad, esencia, dependencia y rigidez [4]. Sin embargo, se pueden señalar algunas limitaciones a esta metodología.

Primero, sólo permite definir el papel taxonómico que juega un concepto particular, sin tratar la corrección estructural de la ontología en su conjunto. Segundo, la metodología no proporciona ningún método para formalizar un modelo ontológico con todos sus elementos.

Las taxonomías jerárquicas se han identificado como la tecnología clave para apoyar las preferencias del usuario en la Web Semántica [18]. En tales sistemas, se puede referir cada concepto en la jerarquía (no sólo una hoja) no sólo como un concepto abstracto (probablemente a través de un conjunto de propiedades que ese concepto contiene) sino también como un data-concept un concepto que puede ser asociado con las instancias afirmadas (por ejemplo, objetos reales). Chaimiel se centra en las propiedades estructurales de la ontología y aseguran las preferencias del usuario. Kiefer, Bernstein y Stocker aplicaron semantic data mapping, ontology mapping y semantic web service matchmaking utilizando técnicas de análisis lingüístico [18,19].

1.2.4 Metodologías de construcción y evaluación de taxonomías.

1.2.4.1. Metodologías Existentes.

La ingeniería ontológica engloba un conjunto de actividades llevadas a cabo durante la conceptualización, diseño, implementación y despliegue de ontologías. Además, cubre temas incluyendo filosofía, metafísica, formalismos de representación del conocimiento, metodología de desarrollo, compartición y reutilización de conocimiento, gestión de conocimiento, modelado de procesos de negocio, sistematización de dominio de conocimiento, información relevante para Internet, estandarización y evaluación [20].

Al igual que han sido desarrolladas numerosas ontologías, han sido numerosas las metodologías de desarrollo de ontologías propuestas. Ya en 1990, Lenat y Guha [21] publicaron algunas consideraciones metodológicas relacionadas con el desarrollo de la ontología CYC. Esta metodología presenta diferentes fases. La primera consiste en la codificación manual de artículos y elementos de conocimiento en los que el sentido común se extrae a mano. La segunda y tercera fases consisten en la adquisición de un nuevo conocimiento de sentido común usando lenguaje natural o herramientas de aprendizaje. La diferencia entre ellas es que en la segunda esta adquisición de conocimiento es realizada por herramientas pero ejecutada fundamentalmente por

humanos, mientras que en la tercera fase la adquisición está principalmente llevada a cabo por herramientas.

Algunos años más tarde, en 1995, Uschold y King publicaron los principales pasos que siguieron en el desarrollo de la ontología Enterprise [22]. Este método propone algunos pasos generales para desarrollar ontologías: primero hay que identificar el propósito, después capturar los conceptos y las relaciones entre los conceptos y los términos usados, para indicar ambos, y finalmente codificar la ontología.

En el mismo año, Grüninger y Fox mostraron la metodología utilizada en el desarrollo de la ontología TOVE (Toronto Virtual Enterprise) [23]. El desarrollo de esta propuesta es como sigue: primero identificar intuitivamente los principales escenarios (las posibles aplicaciones en las que se utilizará la ontología). Después, se realizarán un conjunto de preguntas en lenguaje natural para determinar el alcance de la ontología. Estas preguntas se utilizan para extraer los principales conceptos, sus propiedades, relaciones y axiomas que están formalmente definidos en Prolog. Por tanto, ésta es una metodología muy formal que tiene la ventaja de la robustez de la lógica clásica y puede ser utilizada como guía para transformar escenarios informales en modelos computables.

Un año más tarde, Uschold realiza una propuesta de unificación de ambas metodologías [24]. En la 12th *European Conference for Artificial Intelligence* se presenta la metodología utilizada para construir ontologías dentro del proyecto "*Esprit KACTUS Project*". En ella proponen empezar el proceso construyendo una base de conocimiento (KB) para una aplicación específica y, cuando se necesite una nueva KB en un dominio similar, generalizar el primer KB en una ontología y adaptarla para ambas aplicaciones. La aplicación de este método de forma recurrente permite capturar dentro de la ontología el conocimiento consensual necesario en todas las aplicaciones.

METHONTOLOGY apareció al mismo tiempo y se amplió pocos años más tarde [24]. Es una metodología para construir ontologías tanto desde cero como por el proceso de re-ingeniería. El framework de la Methontology permite la construcción de ontologías a nivel del conocimiento. Incluye: la identificación de las actividades del proceso principal de desarrollo de la ontología (por ejemplo la evaluación, gestión de la configuración, integración, implementación, etc.); un ciclo vital basado en prototipos involucrados; y la metodología misma, que especifica los pasos a seguir para ejecutar cada actividad, las técnicas usadas, los productos del *output* y el modo de evaluarlos.

También en 1997, Swartout propuso la metodología empleada para construir ontologías de dominios a partir de la ontología SENSUS [25]. Este método es una propuesta topdown para obtener ontologías de dominio específico a partir de otras más grandes. Los autores proponen identificar un conjunto de términos que son relevantes en un dominio particular. Tales términos están ligados manualmente a una ontología de amplia cobertura (en este caso a la ontología SENSUS que contiene más de 50.000 conceptos). Seleccionan automáticamente los términos relevantes para describir el dominio y ajustar la ontología SENSUS. Como consecuencia, el algoritmo entrega el

conjunto de términos estructurados jerárquicamente para describir el dominio que se puede utilizar como una base mínima para una KB.

Las metodologías establecidas para la ingeniería de ontologías en [3,27,28] se centran en el desarrollo centralizado de ontologías estáticas, por ejemplo, no consideran la interacción entre construcción-modificación y uso.

Todas estas metodologías no consideran el desarrollo colaborativo de ontologías. Una ontología es un entendimiento común y compartido de un dominio, hasta ahora, se ha puesto el énfasis en el consenso del contenido de la ontología, es decir, en el acuerdo colegiado de la especificación formal de los conceptos, relaciones, atributos y axiomas que la ontología proporciona. Sin embargo, el problema de construir conjuntamente una ontología en un entorno distribuido está todavía por resolver.

Holsapple hace una primera aproximación centrando su metodología en los aspectos colaborativos de la ingeniería ontológica, aunque todavía apunta a una ontología estática [29]. Un ingeniero de conocimiento define una ontología inicial que se amplía y modifica basada en el feedback de un panel de expertos de dominio.

Euzenat, en [30, 31] identificó los siguientes problemas:

1. Relativo a la construcción colaborativa de ontologías: gestión de la interacción y comunicación entre los participantes.

2. Control del acceso de datos.

3. Reconocimiento de un derecho moral sobre el conocimiento.

4. Detección del error y gestión.

5. Gestión concurrente.

Dos son las principales propuestas detalladas sobre cómo construir ontologías, CO4 [31] que propuso una construcción colaborativa de KBs en INRIA y KA [30] usada en construcción de ontologías en la *Knowledge Annotation Initiative* de la *Knowledge Acquisition Community*. CO4 es un protocolo para alcanzar consenso entre varias KBs y está basada en la idea principal de que la gente puede discutir y comprometerse con el conocimiento basado en las KBs. Estas KBs se construyen para ser compartidas y tienen conocimiento consensual, por lo tanto, pueden ser consideradas ontologías. La experimentación se ha hecho, sobre todo, en el dominio genético molecular. Según la propuesta de Euzenat, las KBs están organizadas en un árbol. La hojas se llaman usuario de KBs, y los nodos intermedios, grupos KBs. Por otra parte, no es obligatorio que el usuario de las KBs comparta conocimiento consensuado. Sobre el otro, cada grupo KB representa el conocimiento consensuado entre sus hijos (suscriptor de KBs). El objetivo de la *Knowledge Annotation Initiative*, también conocida como iniciativa KA [32], es modelar la construcción de adquisición de conocimientos utilizando ontologías desarrolladas en un esfuerzo conjunto por un grupo de gente de diferentes lugares usando las mismas plantillas y lenguaje. Para simplificar el roceso de

construcción de la ontología Research-Topic, el agente que coordina la ontología distribuye una plantilla entre los agentes Ontopic, que usan e-mail en su intra-comunicación y también para enviar sus resultados a los agentes que coordinan (expertos en diferentes materias). La ontología se genera a partir del conocimiento presentado por la plantilla. Una vez que los agentes coordinadores de la ontología obtienen todas las porciones de las ontologías de los agentes Ontopic, las integran, actividad que se beneficia de la presencia de un modelo común.

Las herramientas más relevantes, hoy en día, de construcción colaborativa de ontologías son Bibsonomy [33], Collaborative Protégé [34], Hozo [35], DBin [36], OntoWiki [37] y SOBOLEO [38]. Algunas de ellas pueden ser consideradas como extensiones de herramientas de ontologías más tradicionales, mientras que otras se acercan más a lo que se conoce como wikis o escritorios semánticos. La principal diferencia de todas ellas con las propuestas de ingeniería ontológica clásica es que pretenden que los no expertos puedan hacer peticiones con la mayor extensibilidad posible sin que eso interfiera con el modelado formal arriba mencionado. Integrar técnicas inteligentes podría ser una buena decisión, pero deberían estar ocultas para el usuario.

Se han propuesto diferentes formas de clasificar las metodologías para construir ontologías. Una posible clasificación es como sigue:

- Metodologías para construir ontologías desde cero, por ejemplo, [21] y [39].

- Metodologías para la construcción colaborativa de ontologías, por ejemplo, [15, 39,40].

- Metodologías para construir ontologías a través de procesos de reingeniería, por ejemplo, [41].

Las metodologías más famosas para construir ontologías tratan del modo de crear la ontología, con los procesos psicológicos/mentales/reales que tienen lugar, cómo proceder para crear ontologías o controlar los procesos colaborativos. Sin embargo, no hay muchas metodologías centradas en especificar formalmente la estructura y el contenido ontológico. Este problema también ha sido señalado en [2], donde se señala una pequeña discusión en lo relativo al nivel real de conocimiento que una ontología puede representar con éxito. Creemos que hay una necesidad de proporcionar más formalización al contenido y la estructura ontológica y éste es el objetivo del trabajo.

En este ámbito, Ganter y Wille [42] presentan un modelo formal para definir conceptos con las siguientes características:

- Un concepto es un subconjunto de una gran estructura llamado contexto.

- Un concepto formal es un par de conjuntos que representan el concepto y sus atributos respectivamente.

- El orden del concepto está definido por las operaciones de inclusión de conjuntos.

- Se puede usar la teoría Lattice.

En esta propuesta, los llamados conceptos siguen a conceptos previos una vez que esos han ido completamente definidos por el usuario. En nuestra propuesta, la elección de términos es guiada por el usuario, de modo a que esté más cercana a los deseos del usuario.

En la propuesta de Ganter y Wille los términos de los conceptos van detrás de los conceptos previos, una vez que éstos han sido completamente definidos por el usuario. Sin embargo, en nuestra propuesta, la elección de los términos está totalmente guiada por los usuarios, de ese modo se está más cerca de los deseos del usuario, lo cual, mejoría tanto esta propuesta como la de Kunal Punera [42,43].

[43] es otro ejemplo de formalización en el campo de la metodología de construcción que propone un framework que caracteriza una "buena" taxonomía. También proporciona un algoritmo que trabaja de forma completamente automática sin los *inputs* del usuario.

Cabe destacar que todos estos métodos y metodologías descritos, fueron propuestos para la construcción de ontologías, pero hay muchos métodos y metodologías que no solo se encargan de esta parte del diseño sino abarcan temas como: la combinación o fusión de ontologías [44], el aprendizaje [45] y la evaluación [46].

1.2.4.2. Metodología Utilizada

A continuación en esta sección se explicará con detalle la construcción de ontologías taxonómicas (construcción de conceptos y de atributos de dichos conceptos). Esta metodología utilizada marcará la métrica que servirá para evaluar la calidad de la ontología taxonómica resultante. La ontología que se representa a continuación pertenece al dominio de la biología molecular.

Para explicar esta metodología presentaremos en un proceso paso a paso, empezando con la definición formal de los atributos, conceptos y categorías, para dar paso después, a la obtención de una serie de métricas que nos permitirán evaluar la calidad de las ontologías taxonómicas, a la vez que definir unas pautas para la buena construcción de éstas.

Para que esta metodología sea más fácil de entender se irán poniendo ejemplos en cada paso, se utilizarán los conceptos siguientes: Célula, Archaea, Bacteria, Eukaryota, Crenarchaeota, Korarchaeota, Actinobacteria, Cianobacteria.

Formalización

Definiciones sobre atributos

En primer lugar presentamos una serie de definiciones para que los atributos sean manejables en términos de un posterior proceso del modelo propuesto.

Definición 1: Atributo bien definido.

Sea a un atributo de algún concepto c y V permanezca al alfabeto usado para formar términos de un determinado dominio. n representa la máxima longitud permitida para un término y M es un conjunto definible por el usuario cuyo valor puede estar entre N, Z, Q y R. a se dice que está bien definido, escrito well-defined(c,a), si existe una función, llamada:

POSSIBLE_VALUES(c,a), definida como $Vn \times Vn \rightarrow F$

$$donde\ F \subseteq \begin{cases} P(Vn)\ si\ los\ valores\ para\ a\ son\ no\ numéricos \\ P(M)\ en\ otro\ caso \end{cases}$$

Y tal que las siguientes condiciones se cumplen:

i) $c \neq a$

ii) $Card\ (F) \geq 1$ donde *Card* (Cardinalidad) se refiere al número de elementos de un conjunto.

iii) $\{c\} \cap POSSIBLE\_VALUES\ (c, a) = \varphi$

iv) $\{a\} \cap POSSIBLE\_VALUES\ (c, a) = \varphi$

Ejemplo 1: Suponemos que el concepto “Archaea” tiene como atributos “Size”, “Optimun_PH” y “Respiration_type”. También suponemos que el atributo “Size” está fijado en 0.4 micrómetros, por lo tanto Card (Size)=1. El conjunto de posibles valores de “Optimun_PH” es R+ (Reales positivos). Así, podemos decir que “Optimun_PH” está bien definido. Con respecto al atributo “Respiration_type”, si V está definido como un conjunto de caracteres alfanuméricos, incluyendo espacios en blanco y otros caracteres de puntuación, también él se podrá decir que está bien definido.

Definición 2: Abstracción de atributo.

Ahora, consideraremos el rango de valores que un atributo puede tomar para modelar una medida del nivel de abstracción asociado a la definición de los atributos como tal. Así, si pensamos en términos de “cuanto mayor sea el rango de valores que un atributo puede tomar mayor será su nivel de abstracción” se puede dar la siguiente definición:

Sea a un atributo bien definido de un concepto c. Entonces su nivel de abstracción, llamado ***a_abstraction (c, a)***, se define de la siguiente manera:

$$a\_abstraction\ (c,a) \begin{cases} d\ si\ F \in \{N,Z\} \\ D\ si\ F \in \{Q,R\} \\ Card(F) en\ otro\ caso \end{cases}$$

Donde *F* está definida de acuerdo con la Definición 1 y cumple *Card (F) < d < D*

- *F Є {N,Z},* significa que *F* tiene un conjunto infinito contable de elementos.
- *F Є {Q,R},* significa que *F* tiene un conjunto infinito incontable de elementos.

Ejemplo 2: Siguiendo con el ejemplo 1, el nivel de abstracción para "Optiumun_PH" es D, para "Respiration_type" d y 1 para "number_of_wheels".

Propiedad 1: Sea x un atributo bien definido e y un concepto, entonces:

$$\forall x \forall y\ \text{well-defined}(x,y) \rightarrow a\_abstraction\ (x,y) \geq 1$$

Definición 3: Atributo constante-abstracto.

Definiremos ahora un atributo cuando es modelado de modo que su posible conjunto devalores está restringido a uno sólo.

Sea a un atributo bien definido de un concepto c. a se dice ser un atributo constante, escrito ***constant_att(c, a), si a_abstraction (c, a) = 1***. En otro caso, se dice que es un atributo abstracto, escrito ***abstract_att(c, a)***.

Ejemplo 3: En nuestro ejemplo, dado que los niveles de abstracción para Optimun_PH" y "Respiration_type" son D y d respectivamente, ambos son atributos abstractos, mientras que "Size" es constante.

Definiciones sobre conceptos

En este apartado se formalizarán una serie de definiciones para que los conceptos sean manejables en términos de un posterior proceso del modelo propuesto.

Definición 4: Concepto bien definido/estructura interna conceptual.

Sea c un concepto; a_i un atributo bien definido de c; V pertenece al alfabeto usado para formar ambos términos a y c; y n la longitud máxima permitida para los términos. La estructura interna de c, escrita ***IN(c)***, está definida como $Vn \rightarrow P(Vn)$ con . Entonces, c se dice que es $IN(c) = \{ a_i \mid well\_defined(c, a_i)\}$ un concepto bien definido.

Ejemplo 4: En nuestro caso, el concepto "Archaea" estaría bien definido si asumimos que su estructura interna está formada por "Size", "Optimun_PH" y "Respiration_type", por ejemplo.

Definición 5: Abstracción de concepto.

Sea c un concepto bien definido e ***IN(c)*** su estructura interna. Entonces, el nivel de abstracción de c, escrito ***c_abstraction(c)***, viene definido como el vector $[a\_abstraction\ (c, a_i), \ldots, a\_abstraction\ (c, a_m)]$ donde a_i es i-ésimo elemento de ***IN(c)*** en ***Card (IN(c))***.

Ejemplo 5: El nivel de abstracción de "Archaea" es el vector [1, D, d].

Definición 6: Concepto abstracto/ instancia.

Para adaptar la Definición 3 al ámbito de conceptos se propone la siguiente definición:

Sea c un concepto y ***c_abstraction(c)*** su nivel de abstracción. Entonces:

a) c se dice ser una instancia, escrito ***instance(c)***, si
$\forall x \in IN(c),\ constant\_att(c,x)$

b) c se dice abstracto, ***abstract(c)***, si
$\exists\ x \in IN(c)$, tal que $abstract\_att(c,x)$

Ejemplo 6: En nuestro ejemplo, ***c_abstraction(Archaea)= [1, D, d]***. Esto significa que dos atributos son abstractos y, por lo tanto, el concepto "*Archaea*" es abstracto. Si definimos un nuevo concepto "*Crenarchaeota*", con los atributos: *Size=0.4 micrómetros*, *Optimun_PH=3.2* y *trademark=Anaeróbico*. Este último concepto es una instancia debido a que todos sus atributos están definidos, son constantes.

Propiedad 2: Sea x un concepto e y un elemento de ***IN(x)***. Se cumple la siguiente condición:

$$\forall x\ (instance(x) \leftrightarrow \forall x\ \forall y\ \ a\_abstraction(x, y) = 1$$

Definición 7: Instancia definida completamente/incompletamente.

En este momento, tendremos en cuenta que algún atributo de una instancia, aunque deba tener un valor preciso, puede permanecer incierto en el momento de darle los valores al atributo. Sin embargo, parece no tener sentido que todos los atributos de una instancia pudieran estar indefinidos en un instante concreto. Por ello vamos a imponer la restricción de que una instancia debe tener al menos un valor preciso no desconocido de atributo.

Sea c una instancia con ***IN(c)*** su estructura interna. c se dice definida incompletamente, ***incomplete_instance(c)***, si cumple las siguientes condiciones:

i. ***Existe x Є IN(c)*** tal que su valor es preciso aunque desconocido (en el momento del modelado). Podemos decir ***POSSIBLE_VALUES(c,x) = {unknown}.***

ii. Card(x tal que POSSIBLE_VALUES(c,x) = {unknown}) < Card(IN(c))

Definiciones sobre categorías y contextos

En una ontología taxonómica podemos distinguir distintas categorías de conceptos como los subconceptos de otros, los que están por encima llamado super-conceptos, la raíz de la jerarquía, etc. El modo de explicar el significado de un concepto es muy subjetivo, pero en un alto nivel de abstracción podemos intentar clarificarlo a través de sus características intrínsecas, en concreto, las propiedades relacionadas con el concepto. También, utilizamos el nivel de abstracción categórico para definir la clasificación del mismo dentro de la ontología taxonómica. Se hace evidente que es bastante natural usar (sub)categorías en el entendimiento o definición de un concepto. Asumiendo esto, presentaremos en un proceso paso a paso nuestra definición de conceptos a través de una visión centrada en categorías. Para ello, se irán utilizando el conjunto de definiciones que presentamos anteriormente.

Definición 8: Categoría de nivel más alto/ más bajo

Sean c y c' dos conceptos diferentes bien definidos, con IN(c) e IN (c') sus respectivas estructuras internas. c' se dice ser una categoría de nivel superior de c, upper (c',c), si se cumplen las siguientes condiciones:

i) $IN(c') \subseteq IN(c)$

ii) $\forall x \in IN(c'),\ a\_abstraction(c,x) \leq a\_abstraction(c',x)$

iii) $\exists y \in IN(c')$ *tal que* $a\_abstraction(c,y) < a\_abstraction(c',y)$

Ejemplo 7: Consideremos ahora el concepto "*Célula*" que tiene el atributo "*Size*", tal que ***POSSIBLE_VALUES (vehicle, Size) =R+.*** Comprobaremos ahora las condiciones con c siendo "*Archaea*":

i. ***IN (Célula) = {Size} {Size, Optimun_PH, Respiration_type} = IN (Archaea).***
Se satisface.

ii. ***a_abstraction(Célula, Size)= D***
a_abstraction (Célula, Size) = 1

Como d > 1, la segunda condición también se satisface. Además, la tercera condición también se satisface y entonces ***upper (Célula, Archaea)*** se cumple.

Asimismo, c' es una categoría de nivel inferior de c, lower (c', c), si:

i) $IN(c') \supseteq IN(c)$

ii) $\forall x \in IN(c),\ a\_abstraction(c,x) \geq a\_abstraction(c',x)$

iii) $\exists y \in IN(c)$ *tal que* $a\_abstraction(c,y) > a\_abstraction(c',y)$

Ejemplo 8: Si procedemos de la misma forma que en ejemplo anterior, se comprobaría fácilmente que se cumple ***lower (Archaea, Célula).***

Propiedad 3:

Sea C un conjunto no vacío de conceptos bien definidos y dados $x \neq y \neq z \in C$. Entonces, se cumple lo siguiente (lo mismo para lower):

i) $\forall x \neg upper(x,x)$ *(no reflexiva)*

ii) $\forall x \forall y\ upper(x,y) \rightarrow \neg upper(y,x)$ (asimétrica)

iii) $\forall x \forall y \forall z,\ upper(x,y) \wedge upper(y,z) \rightarrow upper(x,z)$ (transitiva)

Definición 9: Contexto conceptual más bajo/más alto.

Sea C un conjunto no vacío de conceptos bien definidos. El contexto conceptual más alto de un concepto c ∈ C, ***upper_context(c, C),*** viene definido como ***{c' ∈ C tal que upper(c',c)}***. De forma similar, el contexto conceptual más bajo de un concepto c ∈ C, ***lower_context(c,C)***, se define como ***{c' ∈ C s.t. lower(c',c)}***.

Ejemplo 9: Ahora, podemos considerar lo conceptos bien definidos de nuestro ejemplo C={Célula, Archaea, Bacteria, Eukaryota }. El concepto "Bacteria" tiene los siguientes atributos {Size, Optimun_PH, Respiration_type, Gram_type}. Los atributos de "Eukaryota" son {Size, Optimun_PH, Respiration_type}.

POSSIBLE_VALUES(Celula, Size)=R+
POSSIBLE_VALUES(Archaea, Size)=0.4 micrometros
POSSIBLE_VALUES(Archaea, Optimun_PH)=3.2
POSSIBLE_VALUES(Archaea, Respiration_type)={any string}
POSSIBLE_VALUES(Bacteria, Size)=0.7 micrometros
POSSIBLE VALUES(Bacteria, Optimun_PH)=5
POSSIBLE_VALUES(Bacteria, Respiration_type)= {any string}
POSSIBLE_VALUES(Bacteria, Gram_type)={any string}
POSSIBLE_VALUES(Eukaryota, Size)=0.6 micrometros
POSSIBLE VALUES(Eukaryota, Optimun_PH)=7
POSSIBLE_VALUES(Eukaryota, Respiration_type)= {any string}

Dadas las definiciones de los conceptos es fácil comprobar que están bien definidos y que satisfacen las siguientes relaciones: ***upper(Célula, Eukaryota)***, ***upper(Célula, Archaea)***, ***upper(Célula, Bacteria)***, ***lower(Eukaryota, Célula)***, ***lower(Archaea, Célula)*** y ***lower(Bacteria, Célula)***.

Además,

upper_context(Archaea, C) = ***upper_context(Bacteria, C)*** = ***upper_context(Eukaryota, C) = {Célula}***
lower_context (Célula, C)={Archaea, Bacteria, Eukaryota) y
lower_context(Archaea, C) = lower_context (Bacteria, C) =
upper_context(Eukaryota, C)= φ.

Propiedad 4

Sea C un conjunto no vacío de conceptos bien definidos y x Є C. Entonces se cumple:

i) $\forall x,\ \phi \subseteq upper\_context(x,C) \subseteq C\backslash\{x\}$

ii) $\forall x,\ \phi \subseteq lower\_context(x,C) \subseteq C\backslash\{x\}$

Corolario 1. Sea C un conjunto no vacío de conceptos bien definidos y x Є C. Entonces:

$$\forall x\ upper\_context(x,C) \cap lower\_context(x,C) = \phi$$

Corolario 2. Sea C un conjunto no vacío de conceptos bien definidos y x Є C. Entonces:

$$\forall x\ instance(x) \rightarrow lower\_context(x,C) = \phi.$$

Definición 10: Categorías de máximo nivel.

Sea C un conjunto no vacío de conceptos bien definidos y c Є C. c se dice ser una categoría de máximo nivel de C, ***top_level(c,C)***, si:

i) $upper\_context(c,C) = \phi$

ii) $\forall x \in C \backslash \{c\},\ c \in upper\_context(x,C)$

Ejemplo 10: En nuestro ejemplo, C={ Célula, Archaea, Bacteria, Eukaryota}, la categoría de máximo nivel es "Célula".

Propiedad 5.

Sea *C* un conjunto no vacio de conceptos bien definidos. Entonces,
Card ({x Є C tal que top_level(x,C)}) = 1

Definición 11: Categoría de igual nivel.

Sean c y c' dos conceptos diferentes bien definidos, con ***IN(c)*** e ***IN (c')*** sus respectivas estructuras internas. Los conceptos c y c' se dicen que pertenecen al mismo nivel conceptual, ***equal (c',c)***, si se cumplen las siguientes condiciones:

i) ***IN(c) = IN(c')***

ii) ***¬upper(c',c) Λ ¬lower(c',c)***

iii) ***Para todo x Є upper_context(c, C), ¬upper(c', x) Λ ¬lower(c', x)***

iv) ***Para todo x Є upper_context(c', C), ¬upper(x, c) Λ ¬lower(x, c)***

Ejemplo 11: Podemos considerar los conceptos "Archaea" y "Eukaryota", ambos tienen los mismos atributos {number_of_wheels, load_capacity, trademark}, tal que:

POSSIBLE_VALUES(Archaea, Size)=0.4 micrómetros
POSSIBLE_VALUES(Archaea, Optimun_PH)=4
POSSIBLE_VALUES(Archaea, Respiration_type)={any string}
POSSIBLE_VALUES(Eukaryota, Size)=0.6 micrómetros
POSSIBLE VALUES(Eukaryota, Optimun_PH)=7
POSSIBLE_VALUES(Eukaryota, Respiration_type)= {any string}

Para los dos conceptos es evidente que se cumple ***upper(Archaea, Eukaryota)*** y ***upper(Eukaryota, Archaea)***. Veamos ahora si ambos están situados en categorías de igual nivel conceptual.

IN(Archaea)= {Size, Optimun_PH, Respiration_type}= IN(Eukaryota)

upper(Archaea, Eukaryota)? FALSE, because ¬ (existe x Є IN(Eukaryota) tal que a_abstraction(Archaea, x) < a_abstraction(Eukaryota, x))

upper(Eukaryota, Archaea)? FALSE, because ¬ (exist x Є IN(Archaea) tal que a_abstraction(Eukaryota, x) < a_abstraction(Archaea, x))

lower(Archaea, Eukaryota)? FALSE, because ¬ (existe x Є IN(Eukaryota) tal que a_abstraction(Eukaryota, x) > a_abstraction(Archaea, x))

lower(Eukaryota, Archaea)? FALSE, because ¬ (exist x Є IN(Eukaryota) tal que a_abstraction(Archaea, x) > a_abstraction(Eukaryota, x))

Por lo tanto, pertenecen al mismo nivel, ***equal(Eukaryota, Archaea)***.

Propiedad 6: Sea C un conjunto no vacío de conceptos bien definidos y x ≠ y ≠ z Є C. Entonces:

i) $\forall x\ equal(x,x)$ (reflexiva)

ii) $\forall x \forall y\ equal(x,y) \leftrightarrow equal(y,x)$ (simétrica)

iii) $\forall x \forall y \forall z\ equal(x,y) \wedge equal(y,z) \rightarrow equal(x,z)$ (transitiva)

Definición 12: Contexto conceptual de igual nivel.

Sea C un conjunto no vacío de conceptos bien definidos. El contexto conceptual de igual nivel de un concepto c Є C, ***equal _context (c,C)***, se define como *{c' Є C tal que equal (c,c')}*.

Ejemplo 12: Por ejemplo, para C={Célula, Archaea, Eukaryota }, equal_context(Archaea, C)= {Eukaryota}

Propiedad 7: Sea C un conjunto no vacío de conceptos bien definidos y x Є C. Entonces se satisface:

$$\forall x,\ 0 \leq Card(equal\_context(x,C)) \leq Card(C)$$

Corolario 3. Sea C un conjunto no vacío de conceptos bien definidos y c Є C. Entonces:

$$\forall x \in C \backslash \{c\}\ (x \in equal\_context(c,C)) \leftrightarrow (x \notin upper\_context(c,C) \cup lower\_context(c,C))$$

Corolario 4. Si C tiene un concepto raíz y uno hoja, entonces:

$$\forall x \in C,\ 0 \leq Card(equal\_context(x,C)) \leq Card(C) - 2$$

Corolario 5. Sea C un conjunto no vacío de conceptos bien definidos y c ∈ C. Entonces:

$$top\_level(c,C) \rightarrow equal\_context(c,C) = \phi$$

Definición 13: Contexto categórico

Sea C un conjunto no vacío de conceptos bien definidos. El contexto categórico de un concepto dado c ∈ C, ***categorical_context(c,C)***, se define como:

$$upper\_context(c,C) \cup lower\_context(c,C) \cup equal\_context(c,C)$$

Ejemplo 13: Así, para nuestro ejemplo, el context categórico del concepto "Archaea" en C={Célula, Archaea, Eukaryota,Bacteria, Crenarchaeota} se puede calcular de la siguiente forma:

upper_context(Archaea, C)= {Célula}
lower_context(Archaea, C)={Crenarchaeota}
equal_context(Archaea, C)={Eukaryota, Bacteria}

Por lo tanto,
categorical_context(Archaea, C)={Célula, Crenarchaeota, Eukaryota, Bacteria}

Corolario 6.
Sea C un conjunto no vacío de conceptos bien definidos. Entonces:

$$\forall x \in C, instance(x) \leftrightarrow categorical\_context(x,C) = upper\_context(x,C) \cup equal\_context(x,C)$$

Corolario 7.
Sea C un conjunto no vacío de conceptos bien definidos y c ∈ C. Entonces:

$$top\_level(c,C) \leftrightarrow categorical\_context(c,C) = lower\_context(c,C)$$

Un modelo ontológico basado en el nivel de abstracción.

En este trabajo una ontología taxonómica es observada como un conjunto de conceptos bien definidos dispuestos de un modo jerárquico según el nivel de abstracción que tienen, junto con un conjunto de axiomas en relación a algún subconjunto de tales conceptos.

Definición 14: Buena conectividad.

Sea C un conjunto no vacío de conceptos bien definidos. Se dice que éstos están bien conectados ***well-connected(C)***, si se satisfacen las siguientes condiciones:

(i) $\forall x \in C, categorical\_context(x,C) \neq \phi$

(ii) $\exists y \in C \text{ tal que } top\_level(y,C)$

Que ningún concepto se quede sin estar conectado a otro.

Ejemplo 14: En nuestro ejemplo, C={Célula, Archaea, Bacteria, Eukaryota}, podemos comprobar si está bien conectado, ***well-connected(C)***.

categorical_context(Célula, C)={Archaea, Bacteria, Eukaryota}≠ φ
categorical_context(Archaea, C)={Célula, Archaea, Bacteria, Eukaryota}≠ φ
categorical_context(Eukaryota, C)={Célula, Archaea, Bacteria}≠ φ
categorical_context(Bacteria, C)={Célula, Archaea, Eukaryota}≠ φ
top_level(Célula,C) se cumple, por lo que well-connected(C).

Definición 15: Conjunto de nivel más alto o más bajo.

Sea C un conjunto no vacío de conceptos bien definidos y S un subconjunto no vacío de C, S C. El conjunto de nivel más alto de S in C, ***upper_set(S,C)*** se define como:

$$\bigcup_{c \in S} upper\_context(c, C)$$

De igual modo, el conjunto de nivel más bajo de conceptos de S en C, ***lower_set(S, C)***, se define como:

$$\bigcup_{c \in S} lower\_context(c, C)$$

Ejemplo 15:

Para el conjunto

C = {Célula, Archaea, Eukaryota, Bacteria, Crenarchaeota, Actinobacteria }, y su subconjunto S = {Actinobacteria, Archaea} podemos decir:

upper_set(S,C)=upper_context(Actinobacteria, C) U
upper_context(Archaea, C)={Célula, Bacteria} U {Célula}={Célula, Bacteria};

lower_set(S,C)=lower_context(Actinobacteria, C) U lower_context(Archaea, C) = ∅ U {Crenarchaeota}={Crenarchaeota}.

Propiedad 8:

0 ≤ Card(upper_set(S,C)) ≤ Card(C) – Card(S)
0 ≤ Card(lower_set(S,C)) ≤ Card(C) – Card(S)

Corolario 8. Sea C un conjunto no vacío de conceptos bien definidos. Entonces:

$$\forall X \subseteq C \; upper\_set(X,C) \cap lower\_set(X,C) = \phi$$

Corolario 9. Sea C un conjunto no vacío de conceptos bien definidos y S Є C no vacío. Entonces:

$$S = \bigcup_{x \in C} instance(x) \rightarrow lower\_set(S,C) = \phi$$

Corolario 10. Sea C un conjunto no vacío de conceptos bien definidos y c Є C. Entonces:

$$top\_level(c,C) \leftrightarrow upper\_set(\{c\},C) = \phi$$

Definición 16: Axioma bien definido

Sea C un conjunto no vacío de conceptos bien definidos y AX una regla/ley definida sobre un conjunto no vacío de conceptos S Є C para las relaciones entre los atributos de los elementos de S. AX se dice bien definido para S en C, ***well-defined(AX,S,C)*** si:

i) $\forall x \in S, abstract(x);$

ii) $scope\ (lower\_set(S,C), AX)$

Donde scope(y,z) permanece para la sentencia lógica ‘el conjunto de conceptos y cumple el axioma z’.

Ejemplo 16: Dado C= {Célula, Archaea, Bacteria, Eukayota, Crenarchaeota, Actinobacteria } y siendo S= {Célula}. Entonces, el axioma A=“El tamaño máximo para una Célula es 10 micrómetros” está bien definido.

Definición 17: Estructura de ontología taxonómica.

Ahora, podemos definir una estructura taxonómica como sigue:

Sea C un conjunto no vacío de conceptos bien definidos y bien conectados, y sea AX un conjunto de axiomas bien definidos sobre el subconjunto S Є C. Una estructura de ontología taxonómica, **TAX**, está definido por <C, AX>.

Ejemplo 17:

En nuestro ejemplo, podríamos definir la siguiente taxonomía: <C,AX>= <{ Célula, Archaea, Bacteria, Eukaryota, Crenarchaeota, Actinobacteria }, {"the maximum size is 10 micrometres"}>.

Definición 18: Concepto padre/hijo.

En las siguientes líneas, definiremos parámetros formales para poder evaluar ontologías taxonómicas. Después pasaremos a tratar de definir un conjunto de conceptos, agrupados en diferentes categorías, teniendo en cuenta su respectivo grado de abstracción. Para ello, utilizaremos el concepto "closest category" siguiendo la terminología aristotélica para clasificar de modo preciso entidades abstractas y concretas. Sea C un conjunto no vacío de conceptos bien definidos y bien conectados con c, c' Є C. c' se dice que es categoría de nivel padre de c, **parent(c',c)**, si sigue las siguientes condiciones:

i) $upper(c',c)$

ii) $not\ (\exists x \in C\ t.q.\ upper(c',x) \wedge upper(x, c))$

Se dice que c es una subcategoría hija de c', ***child(c, c')*** si ***parent(c',c)***. Denotaremos como ***children(c)*** al conjunto de todos los c' tal que ***child(c',c), children(c)={c' | child(c',c)}***.

Ejemplo 18: Considerando los conceptos "Crenarchaeota" y "Archaea"; upper(Archaea, Crenarchaeota) se satisface y no hay otro concepto c tal que upper(c, Crenarchaeota) y upper(Archaea, c). Por lo tanto, se cumplen parent(Archaea, Crenarchaeota) y child(Crenarchaeota, Archaea).

Definición 19: Bien categorizada.

Ahora, podemos establecer que una ontología taxonómica está bien categorizada, ***well_categorised***, cuando cada atributo, de cada concepto en la ontología, tiene un rango de posibles valores formado por la unión de los rangos de valores correspondientes a cada concepto hijo. O bien, si dado un conjunto no vacío de conceptos bien categorizados C, si C es una estructura de ontología taxonómica con las siguientes propiedades:

$$\forall c \in C,\ \forall a \in IN(c)\ POSSIBLE\_VALUES(c,a) = \bigcup_{c' \in children(c)} POSSIBLE\_VALUES(c',a)$$

Ejemplo 19: En nuestro ejemplo, dado C= {Archaea, Crenarchaeota, Euryarchaeota}. Comprobaremos que C está bien categorizado.

POSSIBLE_VALUES(Archaea, Size)=0.4 micrómetros
POSSIBLE_VALUES(Archaea, Optimun_PH)= 3.2
POSSIBLE_VALUES(Archaea, Respiration_type)={Aeróbica, Anaeróbica}
POSSIBLE_VALUES(Crenarchaeota, Size)= 0.4 micrómetros
POSSIBLE_VALUES(Crenarchaeota , Optimun_PH)= 3.2
POSSIBLE_VALUES(Crenarchaeota , Respiration_type)={Aeróbica}
POSSIBLE_VALUES(Euryarchaeota, Size)= 0.4 micrómetros
POSSIBLE_VALUES(Euryarchaeota, Optimun_PH)= 3.2
POSSIBLE_VALUES(Euryarchaeota, Respiration_type)={Anaeróbica}

Dado que "Archaea" tiene dos hijos llamados "Crenarchaeota" y "Euryarchaeota", tenemos que comprobar que el conjunto de valores de cada atributo de "Archaea" es la unión del conjunto de valores del atributo para sus hijos.

Definición 20: Contexto más cercano al más alto/bajo.

Sean c y c' dos conceptos bien definidos. c' es el contexto más cercano por arriba de c, Closest_upper(c',c) si se cumple:

i) $IN(c) = IN(c')$

ii) $\forall x \in IN(c),\ a\_abstraction(c,x) \leq a\_abstraction(c',x)$

iii) $Card(\{x \in IN(c)\ t.q.\ a\_abstraction(c,x) < a\_abstraction(c',x)\})=1$

De igual manera podemos definir ***Closest_lower(c',c)*** donde se deben cumplir las siguientes condiciones

i) $IN(c) = IN(c')$

ii) $\forall x \in IN(c),\ a\_abstraction(c,x) \geq a\_abstraction(c',x)$

iii) $Card(\{x \in IN(c)\ s.t.\ a\_abstraction(c,x) > a\_abstraction(c',x)\})=1$

Criterios de calidad

Trataremos de definir criterios de calidad para la evaluación de ontologías taxonómicas. En primer lugar, nos centraremos en la utilidad. Un indicador de calidad podría ser encontrar el número de instancias que existen en la ontología. Una que hemos calculado este valor, podemos ver en qué medida son útiles algunas categorías

(abstractas) para clasificar otras. Para ello, se considerarán ambos tipos de conceptos, abstractos e instancias.

Además, podemos asumir que un indicador de una buena ontología taxonómica será el número de diferentes niveles de abstracción que contiene, incluyendo el de las instancias, de tal manera que cuantas más instancias haya entre los nodos hoja mejor será la taxonomía.

Definición 21: Estructura de la ontología taxonómica instanciada/abstracta.

Dada una estructura de ontología taxonómica T bien categorizada, con un conjunto de conceptos C bien definidos y bien conectados. T se dice que está instanciada, ***instantiated(T)***, si:

$$\exists x \in C \text{ t.q. } instance(x)$$

T se dice abstracta, ***abstract_t(T)***, si

$$\forall x \in T, abstract(x)$$

Ejemplo 20: Siendo T una estructura de ontología taxonómica bien categorizada cuyos conceptos son {Célula, Archaea, Crenarchaeota, Euryarchaeota}. Definiremos "Crenarchaeota" con Size=0.4 micrómetros, Optimun_PH=3.2, Respiration_type=Aeróbica y definimos "Euryarchaeota" con Size=0.4 micrómetros, Optimun_PH=3.2, Respiration_type=Anaeróbica. T está instanciada porque instance(Crenarchaeota) y instance(Euryarchaeota) se cumplen.

Definición 22: Estructura de la ontología taxonómica instanciada completamente/parcialmente.

Data la estructura de ontología taxonómica T con su conjunto de conceptos bien definidos y bien conectados C. T se dice que está completamente instanciada, ***c_instantiated(T)***, si se cumple lo siguiente:

$$\forall x \in C \text{ t.q. } lower\_context(x,C) = \phi \rightarrow instance(x)$$

En otro caso, se dice que T está parcialmente instanciada, ***p_instantiated(T)***, y cumplirá lo siguiente:

$$\exists x \in C \text{ t.q. } lower\_context(x,C) \neq \phi \rightarrow abstract(x)$$

Definición 23: Grado de instanciación.

Ahora podemos definir una medida del grado de instanciación asociada a la ontología taxonómica. Para ello consideraremos el conjunto de instancias con respecto al conjunto de conceptos abstractos situados en el nivel de abstracción más bajo.

Sea T una estructura de ontología taxonómica bien categorizada con C su conjunto de conceptos bien definidos y bien conectados; Sea ***LOWER(C) = {x ∈ C t.q. lower_context(x,C) = φ} y sea INSTANCES(C) = {x ∈ C t.q. instance(x)}***. Entonces, el grado de instanciación de T, ***degree_inst(T)***, será:

$$Card(INSTANCES(C))/\ Card(LOWER(C))$$

Propiedad 9	$\forall T,\ 0 \leq degree\_inst(T) \leq 1$
Proposition 1.	$\forall T,\ instantiated(T) \rightarrow degree\_inst(T) > 0$
Proposition 2.	$\forall T,\ c\_instantiated(T) \leftrightarrow degree\_inst(T) = \infty$
Proposition 3.	$\forall T,\ p\_instantiated(T) \leftrightarrow (degree\_inst(T) < 1)$
Proposition 4.	$\forall T,\ abstract\_t(T) \leftrightarrow degree\_inst(T) = 0$

Definición 24: Estructura taxonómica bien construida.

Otro parámetro que proponemos para medir la calidad de una ontología taxonómica es la completitud. Podemos asegurar que es completa cuando **para cada atributo de cada concepto abstracto existen dos o más hijos**.

Dada una estructura de ontología taxonómica T con su conjunto de conceptos bien definidos y bien conectados C. T se dice que es una estructura taxonómica bien construida, ***well-built(T)***, si:

$$\forall x \in T\ abstract(x) \rightarrow Card(Children(x)) \geq 2$$

Ejemplo 21:

Sea C={Célula, Archaea, Bacteria, Actinobacteria, Cianobacteria, Crenarchaeota, Euryarchaeota}, entonces C está bien construida porque:

ABSTRACT(C)= {x ∈ C t.q. abstract(x)}={Célula, Archaea, Bacteria}

Children(Célula)={Archaea, Bacteria} entonces card(children(Célula))=2

Children(Archaea)={Crenarchaeota, Euryarchaeota} entonces card(children(Archaea))=2

Children(Bacteria)={Cianobacteria, Actinobacteria} entonces card(children(Bacteria))=2

Por todo ello C es una TAX bien construida.

Definición 25: Grado de completitud

Podemos definir una medida, similar a la de utilidad, para medir el grado de completitud basado en los conceptos que contiene la ontología taxonómica.

Dada la estructura de ontología taxonómica T y su conjunto de conceptos bien definidos y bien conectados C. Sea ***ABSTRACT(C)={x Є C t.q. abstract(x)}*** y ***CHILDREN(C)={x Є C t.q. children(x)}***. El grado de completitud de T, ***degree_completeness(T)***, está definido como:

$$Card(ABSTRACT\ (C))/\ Card(CHILDREN(C))$$

Ejemplo 22:
ABSTRACT(C)={x Є C t.q. abstract(x)}={Célula, Archaea, Bacteria}

CHILDREN(C))={Archaea, Bacteria, Actinobacteria, Cianobacteria, Crenarchaeota, Euryarchaeota }

Entonces degree_completeness:
(T)=Card(ABSTRACT (C))/Card(CHILDREN(C))=3/6=1/2.

Proposition 5. $\forall T,\ degree\_completeness(T) \geq 0$

Proposition 6. $Instantiated(T) \leftrightarrow degree\_completeness(T)=0.$

Proposition 7. $Degree\_completeness(T) \geq 1/2 \leftrightarrow well\text{-}built(T).$

2. Análisis de Objetivos y Metodología

2.1 Objetivos Del Proyecto

Este proyecto consiste en la **creación de una aplicación web para que se puedan crear manualmente ontologías taxonómicas y se puedan almacenar en una base de datos, y también se podrán realizar consultas de varios tipos a las ontologías taxonómicas almacenadas y de otras remotas vía http, todo ello enlazado con una API que permite visualizar las ontologías en la aplicación Protégé**.

Este software ha sido puesto en funcionamiento con el siguiente nombre en inglés:

AVIS
(Integration of Advanced Semantic Web Techniques for Design and Visualization of Knowledge Structures)

A la hora de desarrollar este proyecto se ha tomado como base para el desarrollo una metodología que fue desarrollada teóricamente con anterioridad al desarrollo de este proyecto.

Por lo tanto el proyecto se centra tanto en implementar dicha metodología como en el desarrollo de un prototipo Web para mostrar la funcionalidad de la aplicación. El proyecto se divide en varias partes a realizar:

1. Estudiar y elegir las herramientas necesarias para el desarrollo del proyecto.

2. Implementar la metodología para poder crear y evaluar Ontologías Taxonómicas.

3. Implementar la metodología para realizar consultas de varios tipos a ontologías creadas en dicha aplicación y ontología remotas vía http.

4. Crear el prototipo Web para poder mostrar la funcionalidad de la aplicación.

2.2 Metodología.

El desarrollo del presente proyecto ha estado guiado por la metodología en espiral de Boehm [47]. (Ver figura 1)

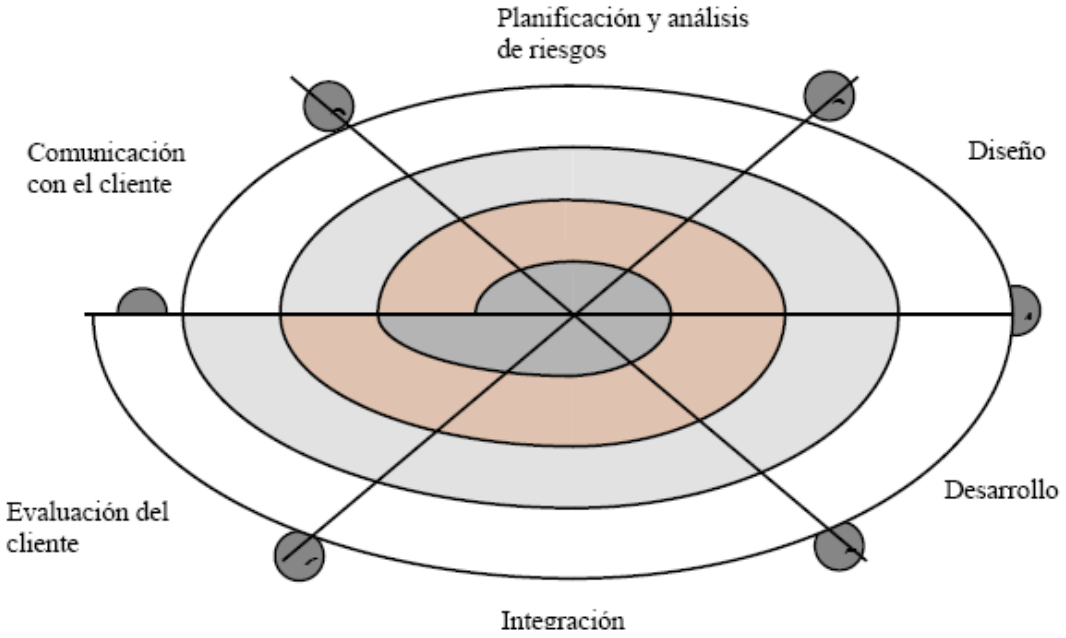


Figura 1

Esta metodología plantea un enfoque “top-down” para el desarrollo de sistemas, ya que, a partir del diseño arquitectónico, cada iteración va agregando detalles, hasta llegar a la implementación. El enfoque “top-down” y evolutivo hace que el riesgo de apartarse de los objetivos especificados se reduzca notablemente, debido a la reducción del tamaño de los problemas a atacar (y por lo tanto la complejidad) durante una iteración.

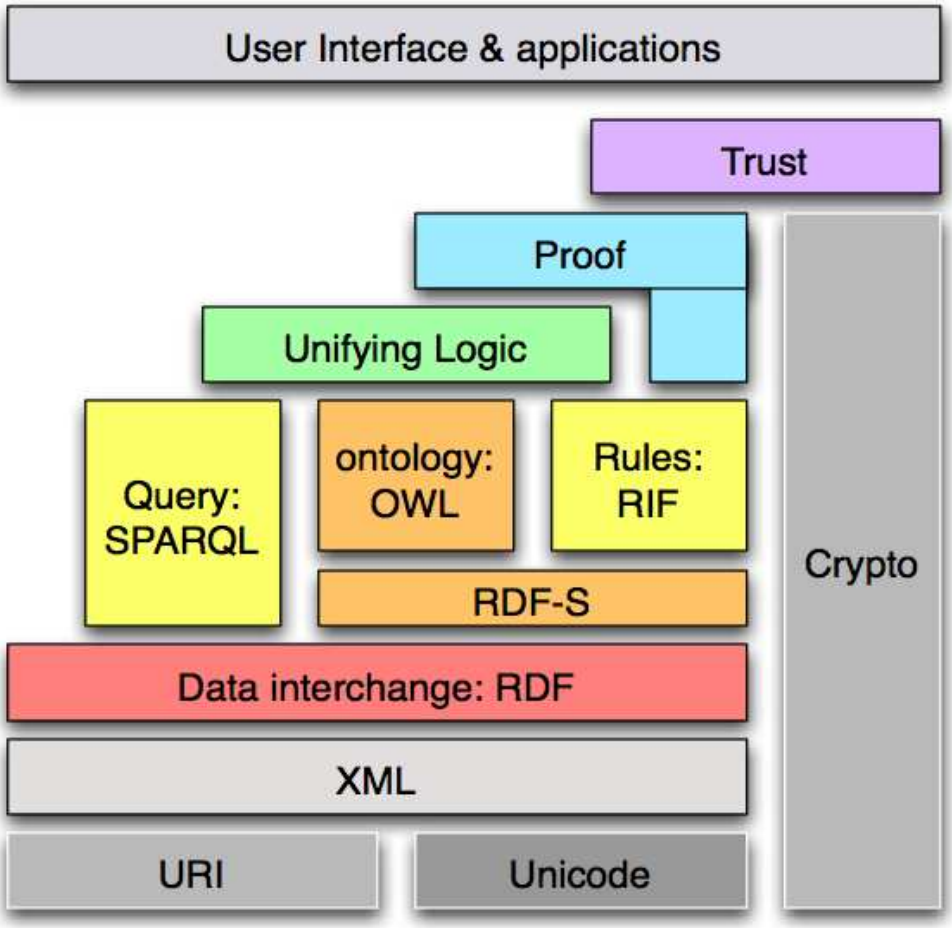
En este documento, se recoge todo el ciclo de vida del desarrollo del proyecto mostrado anteriormente, pasando por las fases de obtención de requisitos, planificación, análisis, diseño, desarrollo, integración y evaluación, y por una fase de pruebas que garantiza que la aplicación cumple correctamente con las especificaciones a partir de las cuales fue diseñada.

2.3 Herramientas y Tecnologías Utilizadas.

2.3.1 Tecnologías de Web Semántica y Ontologías

2.3.1.1 Lenguajes Ontológicos.

Para tener las ideas más claras respecto a los lenguajes ontológicos, véase el siguiente esquema:

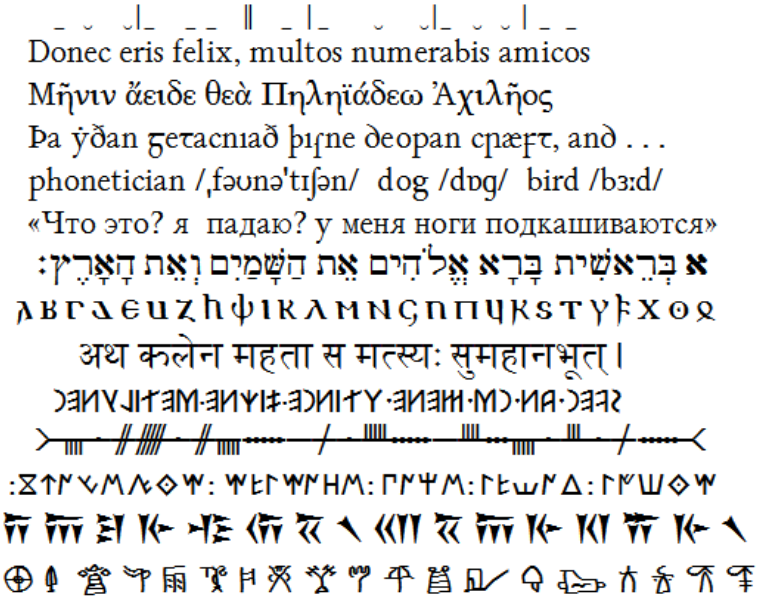


UNICODE

Unicode es estándar industrial cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático. Unicode proporciona un **número único** para cada **carácter**:

sin importar la plataforma,
sin importar el programa,
sin importar el idioma.

La realidad es que tenemos caracteres muy variados en todo el mundo como podemos ver en la siguiente figura:



Los objetivos de Unicode son:

- Crear un repertorio de todos los caracteres en todos los lenguajes del mundo.
- Asignar un **código numérico único a cada carácter** (abandonando la idea de usar 8 bits).
- Mantener en lo posible la **compatibilidad** con ASCII e ISO.

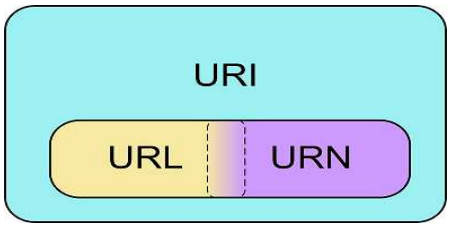
Unicode se organiza en planos, cada uno de ellos de 2^{16} caracteres y se limita el número de planos a 17.

De los 17 planos, el plano 0 está completamente definido. Se denomina “Basic Multilingual Plane” (BMP) y contiene todos los alfabetos “vivos”. El plano 1 está en proceso de ser definido, contiene alfabetos “muertos” o exóticos, y más símbolos matemáticos y musicales. Los restantes planos aun no han sido definidos.

Un código Unicode suele escribirse como **“U-” más 8 cifras hexadecimales** (realmente, bastaría con 6). Las **cuatro primeras** cifras dan el número de **plano**. Las **cuatro inferiores** la **localización dentro del plano** Si el plano es 0, suele omitirse y escribirse **“U+”** más 4 cifras hexadecimales.

URI

Uniform Resource Identifier es una cadena caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.) físico o abstracto.



URI que **describe el método de acceso** a un **recurso** (en cierta manera, nos indica su **localización**).

URI que identifica un recurso por el **nombre**, en un **espacio** determinado. Una URN **puede** servir para tratar sobre un recurso sin expresar su ubicación o como acceder a él. Está más basado en el recurso en sí (contenido, características) que en el lugar en el que se halla el recurso.

IRI es la versión internacionalizada de una URI, que permite utilizar caracteres **Unicode** ya que la sintaxis de las URIs sólo permite usar el subconjunto del US-ASCII y obliga a recodificar los caracteres no permitidos.

XML

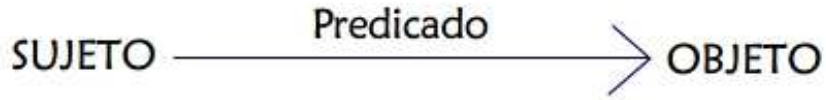
XML es un meta-lenguaje (lenguaje para hacer lenguajes) extensible de etiquetas, tiene un conjunto infinito de etiquetas permitidas. El marcado permitido para una aplicación se suele documentar en un **DTD** o **XML schema.**

Un documento bien formado (well formed) es aquel que cumple con todas las definiciones básicas de formato y puede, por lo tanto, analizarse correctamente usando un analizador sintáctico (parser). **XML no añade semántica** (desde un punto de vista computacional).

EL modelo de datos asociado a XML es un árbol (no un grafo), los hijos están ordenados. XML modela la estructura de documentos, y el **mundo real** no es un documento estructurado, sino una **red de relaciones. XML estandariza formato, no significado.**

RDF

RDF (Resource Description Framework, Marco de Descripción de Recursos) [57,60,61] es un lenguaje que permite utilizar etiquetas XML para expresar el significado del contenido de la página mediante afirmaciones acerca del mismo, codificadas mediante tripletas (sujeto, predicado, objeto), donde cada uno de los elementos de la tripleta queda identificado sin ambigüedad mediante un URI (Uniform Resource Identifier, Identificador Uniforme de Recurso). Constituye, por tanto, un modelo de datos muy simple que permite describir los recursos a que se refiere y las relaciones entre ellos. Está diseñado para ofrecer una formato común para describir información y, por lo tanto, para que pueda ser leída y entendida por una aplicación informática.



El sujeto de la tripleta es el recurso que se está describiendo, el predicado la propiedad o relación que se desea establecer acerca del recurso y, por último, el objeto es el valor de la propiedad o el otro recurso con que se establece la relación. Esta terminología procede de la lógica y de la lingüística, en las que las estructuras predicativas se utilizan también para dar significado a las representaciones sintácticas.

RDFS

RDF Schema es un lenguaje extensible que, al igual que RDF, puede utilizar etiquetas de XML en sus construcciones, y cuya finalidad es la de describir vocabularios de RDF, es decir, ontologías, estructurando así los recursos de RDF. Constituye, por tanto, una extensión semántica de RDF destinada a la representación del conocimiento, y es el primero de los lenguajes que nos permite describir directamente ontologías, si bien su potencia expresiva resulta bastante limitada.

Mediante sus construcciones podemos agrupar los recursos en clases, asociar un rango y un dominio a sus propiedades y establecer, mediante generalización, jerarquías de clases y propiedades. También es posible asociar etiquetas a clases y propiedades.

OWL

OWL (Web Ontology Language) o Lenguaje de Ontologías para la Web, se convirtió en recomendación del W3C el 10 de febrero de 2004 [http://www.w3.org/TR/owl-ref/]. La web: OWL Web Ontology Language Overview [http://www.w3.org/TR/owl-features/] ofrece una explicación detallada de en qué consiste este lenguaje, para qué se usa y cuáles son los conceptos fundamentales empleados por dicho lenguaje [57,60,61].

OWL, como su nombre indica, es el lenguaje definitivo para la creación de ontologías e intercambio de contenidos semánticos a través de la web, y será utilizado con este fin a lo largo de nuestro proyecto. Está diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos. OWL puede usarse para representar explícitamente el significado de términos en vocabularios y las relaciones entre aquellos términos. OWL constituye una ampliación del vocabulario de RDF con la diferencia de que su potencia expresiva es muy superior, lo que le hace totalmente apropiado para representar, con la complejidad necesaria, el conocimiento asociado a cualquier dominio.

Así pues, OWL nos permite estructurar los recursos en clases o conceptos y describir instancias o individuos de estas clases. También diferencia entre propiedades cuyo rango es un tipo de datos básico (DataTypeProperty) y propiedades cuyo rango es un objeto (ObjectProperty), siendo posible restringir la extensión de estos rangos. Incluso permite establecer la equivalencia o diferenciación entre clases, propiedades e individuos. Y, al extender a RDF Schema, puede utilizar también las construcciones de éste para definir relaciones jerárquicas entre clases y propiedades.

OWL se presenta en tres variantes, cada una de las cuales posee una expresividad superior a la anterior, si bien presenta mayores problemas que ésta a la hora de aplicar reglas de inferencia al conocimiento que representa. Cada una de estas variantes puede tener un interés específico para un grupo de usuarios o desarrolladores, por lo que se debe elegir la más apropiada teniendo en cuenta sus características:

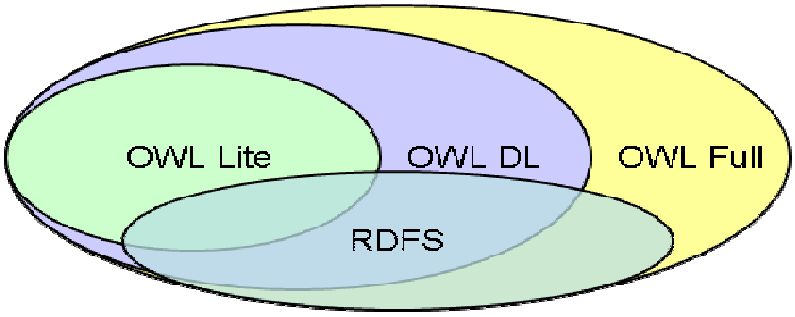
- **OWL Lite** está diseñado para aquellos usuarios que necesitan principalmente una clasificación jerárquica y restricciones simples. Por ejemplo, a la vez que admite restricciones de cardinalidad, sólo permite establecer valores cardinales de 0 ó 1. Debería ser más sencillo proporcionar herramientas de soporte a OWL Lite que a sus parientes con mayor nivel de expresividad, y OWL Lite proporciona una ruta rápida de migración para tesauros y otras taxonomías. OWL Lite tiene también una menor complejidad formal que OWL DL.

- **OWL DL** está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubilidad (todos los cálculos se resolverán en un tiempo finito). OWL DL incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usados bajo ciertas restricciones (por ejemplo, mientras una clase puede ser una subclase de otras muchas clases, una clase no puede ser una instancia de otra). OWL DL es denominado de esta forma debido a su correspondencia con la lógica de descripción (Description Logics, en inglés), un campo de investigación que estudia la lógica que compone la base formal de OWL.

- **OWL Full** está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser considerada simultáneamente como una colección de clases individuales y como una clase individual propiamente dicha. OWL Full permite una ontología para aumentar el significado del

vocabulario preestablecido (RDF o OWL). Es poco probable que cualquier software de razonamiento sea capaz de obtener un razonamiento completo para cada característica de OWL Full.

Cada uno de estos sublenguajes es una extensión de su predecesor más simple en los que ambos pueden ser expresados legalmente y en los que pueden ser válidamente incluidos. El conjunto siguiente de relaciones es correcto, sin embargo, no sus inversas:



Los desarrolladores de ontologías que adoptan OWL deberían considerar cuál es el sublenguaje que mejor se adapta a sus necesidades. La elección entre OWL Lite y OWL DL depende de las necesidades de los usuarios sobre la expresividad de las construcciones, proporcionando OWL DL las más expresivas. La elección entre OWL DL y OWL Full depende principalmente de las necesidades de los usuarios sobre los recursos de metamodelado del esquema RDF (por ejemplo, definir clases de clases, o definir propiedades de clases). Cuando se usa OWL Full en comparación con OWL DL, el soporte en el razonamiento es menos predecible, ya que no existen en este momento implementaciones completas de OWL Full.

A continuación se exponen los tipos de datos de OWL utilizados para la realización de este proyecto:

- ***Class***: Una clase define un grupo de individuos que pertenecen a la misma porque comparten algunas propiedades. Por ejemplo, Deborah y Frank son miembros de la clase Persona. Las clases pueden organizarse en una jerarquía de especialización usando subClassOf. Se puede encontrar una clase general llamada Thing que es una clase de todos los individuos y es una superclase de todas las clases de OWL. También se puede encontrar una clase general llamada Nothing que es la clase que no tiene instancias y es una subclase de todas las clases de OWL.

- ***rdfs:subClassOf:*** Las jerarquías de clase deben crearse haciendo una o más indicaciones de que una clase es subclase de otra. Por ejemplo, la clase Persona podría estar definida como subclase de la clase Mamífero. De esto podemos deducir que si un individuo es una Persona, entonces, también es un Mamífero.

- ***disjointWith***: Es posible establecer que las clases sean disjuntas unas de otras. Por ejemplo, Hombre y Mujer pueden definirse como clases disjuntas. A partir de esta declaración de disjointWith, un razonador puede deducir una inconsistencia cuando un individuo se indica como instancia de ambas clases y de igual manera un razonador puede deducir que si A es una instancia de Hombre, entonces A no es una instancia de Mujer.

- ***rdf:Property:*** las propiedades pueden utilizarse para establecer relaciones entre individuos o de individuos a valores de datos. Ejemplos de propiedades son tieneHijo, tieneFamiliar, tieneHermano, y tieneEdad. Los tres primeros pueden utilizarse para relacionar una instancia de la clase Persona con otra instancia de la clase Persona (siendo casos de ObjectProperty), y el último (tieneEdad) puede ser usado para relacionar una instancia de la clase Persona con una instancia del tipo de datos Entero (siendo un caso de DatatypeProperty). Ambas, owl:ObjectProperty y owl:DatatypeProperty, son subclases de la clase de RDF rdf:Property.

- ***rdfs:domain:*** Un dominio de propiedad reduce los individuos a los que puede aplicarse la propiedad. Si una propiedad relaciona un individuo con otro individuo, y la propiedad tiene una clase como uno de sus dominios, entonces el individuo debe pertenecer a esa clase. Por ejemplo, puede establecerse que la propiedad tieneHijo tenga como dominio la clase Mamífero. De esto, un razonador puede deducir que si "Frank tieneHijo Ana", entonces Frank debe ser un Mamífero. Obsérvese que rdfs:domain se denomina restricción global debido a que la restricción se refiere a la propiedad y no sólo a la propiedad cuando está asociada con una clase en concreto.

- ***rdfs:range:*** El rango de una propiedad reduce los individuos que una propiedad puede tener como su valor. Si una propiedad relaciona a un individuo con otro individuo, y ésta como rango a una clase, entonces el otro individuo debe pertenecer a dicha clase. Por ejemplo, la propiedad tieneHija debe establecerse que tiene como rango la clase Mamífero. A partir de aquí, un razonador puede deducir que si Louise está relacionada con Deborah mediante la propiedad tieneHija, (por ejemplo, Deborah es hija de Louise), entonces Deborah es un Mamífero. El rango es, al igual que el dominio, una restricción global.

- ***hasValue***: (Valores de la propiedad): Una propiedad puede ser necesaria que tenga un determinado individuo como un valor. Por ejemplo, instancias de la clase ciudadanosHolandeses pueden ser caracterizadas como las personas que tiene PaísesBajos como valor de su nacionalidad. (El valor de la nacionalidad, PaísesBajos, es una instancia de la clase Nacionalidades).

- ***disjointWith:*** Es posible establecer que las clases sean disjuntas unas de otras. Por ejemplo, Hombre y Mujer pueden definirse como clases disjuntas. A partir de esta declaración de disjointWith, un razonador puede deducir una inconsistencia cuando un individuo se indica como instancia de ambas clases y 33 de igual manera un razonador puede deducir que si A es una instancia de Hombre, entonces A no es una instancia de Mujer.

- ***unionOf***: OWL DL y OWL Full permiten combinaciones boleanas arbitrarias de clases y de restricciones: unionOf, complementOf, e intersectionOf. Por ejemplo, usando unionOf, podemos indicar que una clase contiene elementos que son ciudadanosdeEE.UU o ciudadanosHolandeses. Usando complementOf, podríamos indicar que los niños no son PersonasMayores (es decir, la clase Niños es una subclase de complemento de PersonasMayores. La ciudadanía de la Unión Europea se podía describir como la unión de las ciudadanías de todos los Estados miembros.

- ***minCardinality, maxCardinality*** (Cardinalidad completa): Mientras que en OWL Lite, las cardinalidades se ciñen a como mínimo, como máximo o exáctamente 1 ó 0, OWL Full permite realizar declaraciones de cardinalidad para números enteros no negativos arbitrarios. Por ejemplo la clase de MatrimonioconDobleSueldoSinHijos restringiría la cardinalidad de la propiedad tieneSueldo a una cardinalidad mínima de 2 (mientras que la propiedad tieneHijo tendría que ser restringida a cardinalidad 0).

Con todo esto, OWL nos proporciona un lenguaje que permite añadir respecto a RDF las siguientes capacidades a las ontologías:

- Capacidad de ser distribuidas a través de varios sistemas.
- Escalable a las necesidades de la Web.
- Compatible con los estándares Web de accesibilidad e internacionalización.

2.3.1.2 JENA.

JENA, (http://jena.sourceforge.net/), [49,50,51] es una plataforma de software libre desarrollada por HB Labs para la creación de aplicaciones de Web Semántica que ofrece un buen número de herramientas con ese objetivo. Entre ellas destaca, por su utilidad para nuestro proyecto, una API para Java que permite trabajar con ontologías en OWL y RDF Schema con una potencia que cubre todos los aspectos del OWL Full, haciendo posible incluso utilizar algoritmos de inferencia, sin perder robustez en ningún momento.

Es por ello que sobresale por encima de otras APIs que tienen el mismo objetivo y que consideramos como opciones a la hora de realizar la implementación, pero que descartamos por su clara inferioridad frente a JENA, tal como ocurre con el caso de OWL API. En este proyecto en concreto, utilizaremos esta API para acceder a ficheros OWL y obtener de ellos la información necesaria para el modelo de ontología que utilizamos en nuestro proyecto. Su corazón es la API RDF, la cual soporta la creación, manipulación y consulta de grafos RDF. La API también soporta diferentes tecnologías de almacenamiento y permite lectura y escritura automática para diferentes lenguajes que los desarrolladores pueden usar para representar grafos RDF.

Jena Incluye:

- Presentaciones múltiples y flexibles de grafos RDF para las aplicaciones de los programadores, permitiendo a los grafos de datos poder ser accedidos y manipulados a través de interfaces de mayor nivel.
- Soporte para la persistencia en Bases de datos.

- Incluye además una API para el manejo de Ontologías.

- Soporta el lenguaje OWL.

Esta API proporciona clases java para representar modelos y manipularlos: creación, escritura y lectura, carga en memoria, navegar un modelo a partir de la URI de un recurso, consultar el modelo (se podrá buscar información del modelo y realizar consultas avanzadas), operaciones sobre modelos (unión, intersección y diferencia). Permite también representar recursos, propiedades, literales y statements.

Debido a la gran potencia que presenta esta API, su estructura de clases e interfaces resulta bastante compleja, por lo que trataremos de resumir muy superficialmente esta compleja jerarquía. En ella, las ontologías vienen representadas por la interfaz OntModel, y cada uno de los elementos de la ontología, por la interfaz OntResource. De esta interfaz se derivan OntClass que representa a las clases, Individual que representa las instancias o individuos de estas clases y OntProperty que representa a las propiedades de las clases.

Cada una de estas interfaces contiene los métodos necesarios para acceder a los elementos de la jerarquía de clases, atributos y relaciones de la ontología, así como a la estructura de ésta, restricciones, etiquetas, comentarios y, en general, a todos los aspectos de la ontología que necesitamos recopilar.

Además, JENA incluye varios componentes, entre los que cabe destacar:

- ARP: Un Parser de RDF.

- API de Ontologías con soporte para OWL en los tres niveles: OWL Lite, OWL DL y OWL Full.

- API de ontologías con soporte para DAML y RDF Schema.

- Soporte para persistencia: permite crear modelos persistentes que son mantenidos de forma transparente al usuario en una base de datos relacional. Jena 2 soporta MySQL, Oracle y PostgreSQL.

- RDQL: lenguaje de consultas para RDF desde un enfoque totalmente declarativo, considera un modelo RDF como un conjunto de tripletas (Objeto, Propiedad, Valor). Permite especificar patrones que son mapeados contra las tripletas del modelo para retornar un resultado y permite realizar consultas en RDQL desde una aplicación Java.

- API para realizar consultas SPARQL desde una aplicación Java.

- Validador de OWL: Existe la posibilidad de realizar una validación básica de OWL. Esta validación sólo comprueba la sintaxis, no infiere ni razona. Para validaciones más complejas Jena 2 ofrece soporte para inferencias y detecta la violación de las restricciones definidas en el Schema por las instancias.

- Subsistema de Razonamiento: Inferir es deducir información adicional. Al código que realiza la tarea de inferir se le llama razonador (Reasoner).

2.3.1.3 PROTÉGÉ.

Protégé es una herramienta libre para el desarrollo de Ontologías y Sistemas basados en el conocimiento creada en la Universidad de Stanford. Protégé está desarrollada en Java con código abierto y puede funcionar perfectamente bajo el Sistema Operativo Windows. En Protégé, las ontologías pueden ser exportadas en diferentes formatos, entre ellos están RDF Schema y OWL. El código fuente de Protégé lo podemos encontrar en http://protege.stanford.edu.

Un proyecto en Protégé consiste en el desarrollo de una ontología o estructura de conocimiento. Los elementos que se pueden ir creando son fundamentalmente clases, slots, formularios, instancias y consultas, aunque la herramienta es modular y permite adicionar más componentes de una forma sencilla. Cada uno de estos elementos dispone de una etiqueta en la ventana principal de la herramienta, seleccionando cada una de ellas podemos elegir el tipo de elemento concreto sobre el que se va a trabajar.

Protégé está apoyado por una fuerte comunidad de desarrolladores y académicos, gubernamentales y usuarios corporativos, que están utilizando Protégé para soluciones de conocimiento en áreas tan diversas como la biomedicina, la inteligencia y el modelado empresarial.

Protégé, cómo se ha comentado antes, se centra en la construcción de modelos de dominio y conocimiento de aplicaciones basadas en ontologías. En su núcleo, Protégé implementa un amplio conjunto de conocimientos de modelado de estructuras y acciones que apoyan la creación, visualización y manipulación de ontologías en distintos formatos de representación. Protégé se pueden personalizar para facilitar la creación de modelos de conocimiento y entrada de datos. Además, Protégé se puede ampliar en Java a través de un Application Programming Interface (API) para la construcción de herramientas basadas en el conocimiento y aplicaciones.

La plataforma Protégé soporta dos formas principales de modelar ontologías:

- El editor **Protégé-Frames** permite a los usuarios construir y poblar las ontologías basadas en Frames, de acuerdo con el protocolo Open Knowledge Base Connectivity (OKBC). En este modelo, una ontología consta de un

conjunto de clases organizadas en una jerarquía de subsunción para representar conceptos importantes de un dominio, un conjunto de espacios asociados a las clases para describir sus propiedades y relaciones, y un conjunto de instancias de las clases - ejemplares individuales de los conceptos que sostienen los valores específicos de sus propiedades.

- El editor **Protégé-OWL** permite a los usuarios construir ontologías para la Web Semántica, en particular en la Web del W3C Ontology Language (OWL). Una ontología OWL pueden incluir descripciones de las clases, de las propiedades y de sus instancias. Una vez tenemos la ontología, la semántica formal de OWL especifica cómo derivar sus consecuencias lógicas, es decir, hechos no literales presentes en la ontología, pero implicados por la semántica. Estas implicaciones pueden estar basadas en un documento único o múltiples documentos distribuidos que se han sido combinados usando mecanismos OWL (para más información véase el OWL Web Ontology Language Guide disponible en la web de Protégé).

Una vez comentado el objetivo de Protégé decir el porqué ha sido seleccionada esta herramienta como manipuladora de Ontologías y no cualquier otra herramienta.

- Principales Ventajas o Facilidades de Protégé:

  - ✓ Herramienta libre y no de pago.
  - ✓ Disponibilidad de código fuente para extensibilidad para otras aplicaciones o cualquier uso.
  - ✓ Desarrollada en el conocido Lenguaje de Programación Java.
  - ✓ Interfaz de usuario muy intuitiva y manejo con Conceptos y Propiedades muy sencillo.
  - ✓ Posibilidad de diferentes formatos de exportación: CLIPS, HTML, N-TRIPLE, N3, OWL, RDF Schema y TURTLE.
  - ✓ Permite al usuario:
    - Construir un dominio Ontológico.
    - Personalizar formas de adquisición del dominio.
    - Penetrar en el dominio del conocimiento.

2.3.1.4 Pellet.

Es una herramienta creada en java para razonar ontologías OWL DL como es nuestro caso, su código es libre y podemos descargarlo aquí http://clarkparsia.com/pellet/ . Soporta la total expresividad de OWL DL y casi toda la totalidad de OWL 1.1. [65]

Pellet posee todos los servicios típico de inferencia (de razonamiento).

- Chequeo de consistencia: Comprueba que las ontología no poseen hecho contradictorios.

- Concept Satisfiability: Determinar si una clase puede tener una instancia.

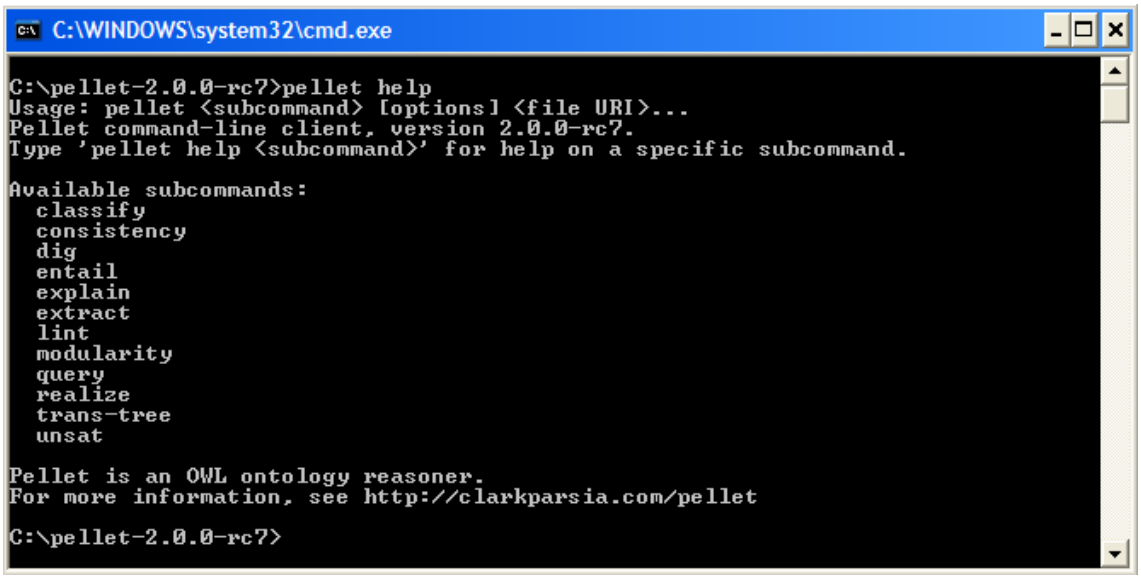
- Calssify: Crea la jerarquía completa de clases.

- Encuentra la clase más específica a la que pertenece un individuo.

Pellet se puede lanzar desde una línea de comandos o mediante una API de programación.

Pellet está desarrollado por ***Clark&Parsia*** y la última versión disponible que es la que se usa en este proyecto es la ***pellet-2.0.0-rc7*** con fecha 11 Junio de 2009.

La reciente versión de pellet se puede ejecutar en una línea de comandos, el comando ***pellet help*** nos muestra en pantalla todas las opciones que tiene esta versión.



Comando pellet help

Véase en la figura todas las opciones de pellet, de las cuales podemos destacar las opciones de *classify*, *consistency*, *explain*, y *query*. Pellet es bastante completo en lo a que opciones se refiere y destaca sobre todo porque los rultados los ofrece en un formato tabulado, por ejemplo, las jerarquías de conceptos las escribe en forma de árbol.

2.3.2 Tecnologías de Diseño.

2.3.2.1 Diseño orientado a objetos

Como metodología de diseño se ha elegido una metodología orientada a objetos por diversas razones:

Proximidad de los conceptos de modelado respecto de las entidades del mundo real.

- Mejora la captura y validación de requisitos.

- Acerca el “espacio del problema” y el “espacio de la solución”.

- Constituye un sistema de modelado integrado de propiedades estáticas y dinámicas del ámbito del problema.

- Presenta conceptos comunes de modelado durante el análisis, diseño e implementación.
    - Facilita la transición entre las distintas fases.
    - Favorece el desarrollo iterativo del sistema.
    - Disipa la barrera entre el “qué” y el “cómo”.

- Facilita la construcción, mantenimiento y reutilización.

2.3.2.2 UML.

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group) [59]. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa (Lengua de Modelación Unificada), no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la orientación a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas:

- *Diagramas de despliegue*, para modelar la distribución de los distintos componentes físicos utilizados en la implementación de un sistema, así como las relaciones entre ellos.

- *Diagramas de paquetes*, para mostrar los diferentes subsistemas logicos en que se divide el sistema y mostrar las dependencias existentes entre ellos.

- *Diagramas de clases*, para especificar los diferentes tipos de objetos relevantes al dominio de la aplicación, así como las relaciones entre los mismos.

- *Diagramas de casos de uso*, para especificar la funcionalidad del sistema, de una forma clara y concisa, en lenguaje natural, y por tanto fácilmente comprensible por todas las personas involucradas en el desarrollo del proyecto.

- *Diagramas de secuencia*, en los que se muestra la evolución en el tiempo de los mensajes que intercambian los diferentes actores y objetos del sistema en un escenario determinado. Es decir, detallan la funcionalidad de los casos de uso en el tiempo.

Además de haberse convertido en un estándar de facto, UML es un estándar industrial promovido por el grupo OMG. Para la revisión de UML se formaron dos "corrientes" que promovían la aparición de la nueva versión desde distintos puntos de vista. Finalmente se impuso la visión más industrial frente a la académica. Recientemente se ha publicado la versión 2.0 en la que aparecen muchas novedades y cambios que, fundamentalmente, se centran en resolver carencias prácticas. Además, esta versión recibe diversas mejoras que provienen del lenguaje SDL.

A pesar de su status de estándar ampliamente reconocido y utilizado, UML siempre ha sido muy criticado por su carencia de una semántica precisa, lo que ha dado lugar a que la interpretación de un modelo UML no pueda ser objetiva. Otro problema de UML es que no se presta con facilidad al diseño de sistemas distribuidos. En tales sistemas cobran importancia factores como transmisión, persistencia, etc. UML no cuenta con maneras de describir tales factores. No se puede, por ejemplo, usar UML para señalar que un objeto es persistente o remoto, o que existe en un servidor que corre continuamente y que es compartido entre varias instancias de ejecución del sistema analizado. Sin embargo, UML si acepta la creación de nuestros propios componentes para este tipo de modelado

2.3.2.3 Patrones de Diseño.

Los Patrones de Diseño (Design Patterns) [58] son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un Patrón de Diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

Para el desarrollo de este proyecto se ha seguido el patrón Modelo Vista Controlador (MVC). MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML o JSP y el código que provee de datos dinámicos a la página, el controlador es el Sistema Gestor de Base de Datos y el modelo es el modelo de datos.

El modelo es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o portes en un carrito de la compra. Evidentemente en este proyecto la ontología es el modelo, ya que es la representación específica de la información.

La vista presenta el modelo en un formato adecuado para interactuar, usualmente es la interfaz de usuario. Por último, el controlador responde a eventos. Usualmente acciones del usuario, e invoca cambios en el modelo y, probablemente, en la vista.

Muchos sistemas informáticos utilizan un Sistema Gestor de Base de Datos para gestionar los datos. En MVC corresponde al controlador. Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1) El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace).

2) El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3) El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4) El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por

ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice.

5) La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

2.3.3 Tecnologías de Desarrollo.

2.3.3.1. Java.

Uno de los condicionantes que nos han guiado a la hora de abordar el desarrollo del sistema informático resultado de este proyecto ha sido el de conseguir una herramienta multiplataforma capaz, por tanto, de funcionar en diferentes sistemas operativos. También se ha planteado el reto de seleccionar un lenguaje de programación conocido, ampliamente extendido, y que permita fácilmente la reutilización del código desarrollado en futuras revisiones o versiones de la herramienta. La solución ha estado clara desde el principio, y ha sido la de utilizar Java [48] (http://java.sun.com/) como lenguaje de programación de la herramienta.

Además de los dos requisitos mencionados, se han valorado también sus otras no menos importantes características, que han hecho de él el principal lenguaje de programación multiplataforma:

Orientación a objetos. El diseño de la aplicación se ha realizado en base a clases de objetos percibidos del dominio del sistema. Es por ello muy importante la elección de un lenguaje de programación que permita representar estos objetos. Además, Java es un lenguaje de herencia simple, aspecto que permite la reutilización de código sin complicar la jerarquía de clases, pero que soporta herencia múltiple de interfaces, lo que resulta de gran utilidad.

- ***Robusto***. Frente a otros lenguajes orientados a objetos, *Java* es tipado estáticamente, lo que permite la comprobación de tipos en tiempo de compilación. Además, el uso de excepciones y la comprobación de límites de tablas proporcionan una gran seguridad en tiempo de ejecución (ante un fallo en la programación, el programa no se cuelga, sino que produce una excepción).

- ***Seguro***. La ejecución de programas en *Java* pasa por una serie de comprobaciones, tanto de privilegios como de *bytecodes*, que hacen que se produzcan excepciones cuando un programa intente acceder a un fichero para el que no tiene permiso o cuando intente acceder a una zona del sistema.

- ***Multiplataforma***. Esta es una de las características más importantes y reconocidas de *Java*. El hecho de que los programas Java no se ejecuten directamente sobre el sistema operativo, sino sobre la maquina virtual *Java*, hace que puedan escribirse y ejecutarse programas *Java* independientemente de la maquina y el sistema operativo.

- ***Interpretado***. El hecho de que *Java* sea interpretado hace de el un lenguaje lento, pero es la clave de otras características como la multiplataforma o el chequeo de *bytecodes*.

- ***Distribuido***. *Java* ofrece mecanismos para interactuar con otras maquinas mediante *sockets* vía *TCP/IP*.

- ***JDBC***. Java ofrece la interfaz *JDBC* para acceder a una base de datos relacional, como la utilizada para almacenar toda la información del sistema desarrollado.

- ***En expansión***. *Java* es uno de los lenguajes que ha alcanzado mayor popularidad en los últimos anos, y parece claro que seguirá siendo así.

- ***Gratuito***. No es necesario pagar ninguna licencia para utilizar el lenguaje ni la maquina virtual sobre la que se ejecutara. Simplemente se descarga de *Internet* y se instala.

2.3.3.2. Java con JENA.

Ya se ha comentado que Jena es una plataforma de software libre que ofrece un marco de recursos Java para construir aplicaciones de Web Semántica. Ofrece un entorno para RDF, RDF Schema y OWL e incluye un motor basado en reglas de inferencia. Su API RDF, soporta la creación, manipulación y consulta de grafos RDF.

La API también soporta diferentes tecnologías de almacenamiento y permite lectura y escritura automática para diferentes lenguajes que los desarrolladores pueden usar para representar grafos RDF.

2.3.3.3. J2EE.

Java Enterprise Edition se puede decir que es la versión empresarial-comercial de las aplicaciones web java. Consiste en aplicar el patrón MVC (Modelo-Vista-Controlador) a una aplicación web.

J2EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son *conformes a Java EE.*

Java EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores.

2.3.3.4. BEA Workshop for JSP.

Esta es una herramienta desarrollada por ORCALE http://www.oracle.com/bea/index.html creada con un plugin para la plataforma de desarrollo en java Eclipse. El propósito de BEA Workshop es proporcionar un modelo de programación en java orientado totalmente al entorno web. Posee características y utilidades suficientes para la programación con JSP, JSTL, HTML, XML, etc. Es decir BEA Workshop es una herramienta Eclipse mejorada para trabajar con java en entornos web [64].

En cuanto a la herramienta Eclipse decir que es una poderosa plataforma de software libre que permite integrar diferentes aplicaciones y herramientas para construir un completo entorno integrado de desarrollo que presenta una gran potencia y flexibilidad debidas a una arquitectura de *plugins* que hace posible añadir funcionalidades a la plataforma de modo sencillo y transparente. Además de su potencia y flexibilidad es muy cómoda de utilizar por el gran numero de herramientas de apoyo a la programación que contiene.

El proyecto Eclipse se lanzó originalmente en Noviembre de 2001, cuando IBM donó 40 millones de dólares del código fuente de Websphere Studio Workbench y formó el Eclipse Consistorium para controlar el desarrollo continuado de la herramienta. El objetivo de Eclipse era “desarrollar una plataforma industrial robusta, con todas las características y de calidad industrial para el desarrollo de herramientas altamente integradas”.

Como ventajas que aporta Eclipse, cabe destacar explícitamente:

Es una herramienta de software libre.

- Soporta la construcción de una variedad de herramientas para el desarrollo de aplicaciones.

- Soporta el desarrollo de aplicaciones basadas en GUI y non-GUI.

- Soporta plugins que manipulan diferentes tipos de lenguajes y ficheros, como Java, C, C++, GIF, JPG, etc.

- Está disponible para una gran cantidad de sistemas operativos incluyendo los más extendidos Windows y Linux.

- Ofrece a los desarrolladores herramientas que facilitan la creación de plugins.

- Mediante JDT facilita la creación de aplicaciones programadas en Java.

2.3.3.5. AJAX.

AJAX, acrónimo de ***A**synchronous **J**avaScript **A**nd **X**ML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. [66]

"¿Por qué actualizar todos los datos del servidor de la página web cuándo sólo se necesita modificar una mínima parte de ella? SOLUCIÓN: AJAX" – AJAX permite actualizar una parte de la página específica, sin tener que actualizar el resto de la página.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

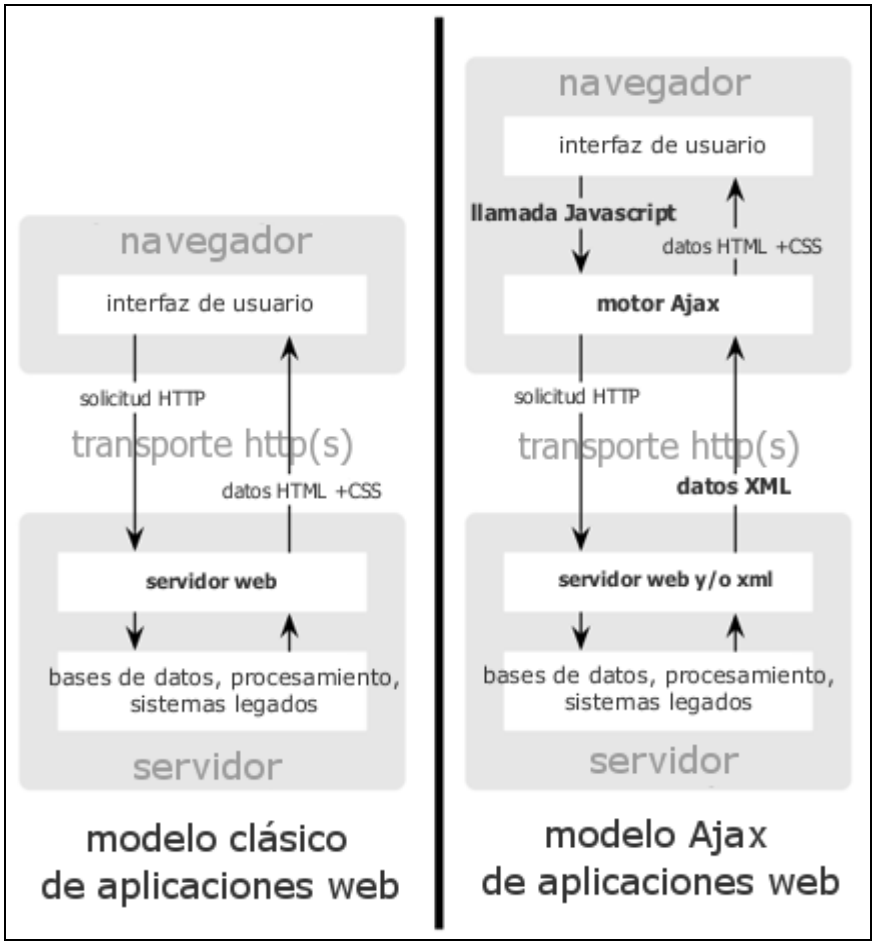
Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

AJAX es una combinación de cuatro tecnologías ya existentes:

- **XHTML** (o **HTML**) y hojas de estilos en cascada (**CSS**) para el diseño que acompaña a la información.
- Document Object Model(**DOM**) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto **XMLHttpRequest** para intercambiar datos de forma asíncrona con el servidor web. En algunos frameworks y en algunas situaciones concretas, se usa un objeto iframe en lugar del XMLHttpRequest para realizar dichos intercambios.
- **XML** es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

AJAX no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

Ahora vamos a comparar el modelo clásico de aplicaciones Web y el modelo AJAX de aplicaciones Web. El modelo clásico de aplicaciones Web funciona de esta forma: La mayoría de las acciones del usuario en la interfaz disparan un requerimiento HTTP al servidor web. El servidor efectúa un proceso (recopila información, procesa números, hablando con varios sistemas propietarios), y le devuelve una pagina HTML al cliente. Mientras el servidor esta haciendo lo suyo, ¿qué esta haciendo el usuario? Exacto, esperando. Y, en cada paso de la tarea, el usuario espera más.



Obviamente, si estuviéramos diseñando la Web desde cero para aplicaciones, no querríamos hacer esperar a los usuarios. Una vez que la interfaz esta cargada, ¿porqué la interacción del usuario debería detenerse cada vez que la aplicación necesita algo del servidor? De hecho, ¿porque debería el usuario ver la aplicación yendo al servidor?

Una aplicación AJAX elimina la naturaleza frenar la interacción en la Web introduciendo un intermediario -un motor AJAX- entre el usuario y el servidor. Parecería que sumar una capa a la aplicación la haría menos reactiva, pero la verdad es lo contrario.

En vez de cargar un pagina Web, al inicio de la sesión, el navegador carga el motor AJAX (escrito en JavaScript). Este motor es el responsable de modificar la interfaz que el usuario ve y de comunicarse con el servidor en nombre del usuario. El motor AJAX permite que la interacción del usuario con la aplicación suceda

asincrónicamente (independientemente de la comunicación con el servidor). Así el usuario nunca estará mirando una ventana en blanco del navegador y un icono de reloj de arena esperando a que el servidor haga algo.

2.3.3.6. PostgreSQL.

PostgreSQL [52,53,54] es un sistema de gestión de base de datos relacional orientada a objetos de software libre, publicado bajo la licencia BSD. Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Algunas de sus principales características son, entre otras:

Alta concurrencia: Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

Amplia variedad de tipos nativos: PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria.
- Texto de largo ilimitado.
- Figuras geométricas (con una variedad de funciones asociadas)
- Direcciones IP (IPv4 e IPv6).
- Bloques de direcciones estilo CIDR.
- Direcciones MAC.
- Arrays.

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

Otras características:

• Claves ajenas también denominadas Llaves ajenas o Claves Foráneas (foreign keys).

• Disparadores (triggers): Un disparador o trigger se define en una acción específica basada en algo ocurrente dentro de la base de datos. En PostgreSQL esto significa la ejecución de un procedimiento almacenado basado en una determinada

acción sobre una tabla específica. Ahora todos los disparadores se definen por seis características:

- El nombre del trigger o disparador
- El momento en que el disparador debe arrancar
- El evento del disparador deberá activarse sobre...
- La tabla donde el disparador se activara
- La frecuencia de la ejecución
- La función que podría ser llamada

Entonces combinando estas seis características, PostgreSQL le permitirá crear una amplia funcionalidad a través de su sistema de activación de disparadores (triggers).

- Vistas.
- Integridad transaccional.
- Herencia de tablas.
- Tipos de datos y operaciones geométricas.

2.3.3.7. Apache Tomcat.

Apache Tomcat [55] funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems.

Tomcat es un servidor Web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor Web Apache.

Tomcat puede funcionar como servidor Web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Tomcat es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software Licence. Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 6.x, que implementan las especificaciones de Servlet 2.5

y de JSP 2.1. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets Catalina.

La jerarquía de directorios de instalación de Tomcat incluye:

• bin - arranque, cierre, y otros scripts y ejecutables.

• common - clases comunes que pueden utilizar Catalina y las aplicaciones web.

• conf - ficheros XML y los correspondientes DTD para la configuración de Tomcat.

• logs - logs de Catalina y de las aplicaciones.

• server - clases utilizadas solamente por Catalina.

• shared - clases compartidas por todas las aplicaciones web.

• webapps - directorio que contiene las aplicaciones web.

• work - almacenamiento temporal de ficheros y directorios

2.3.3.8. Servlets.

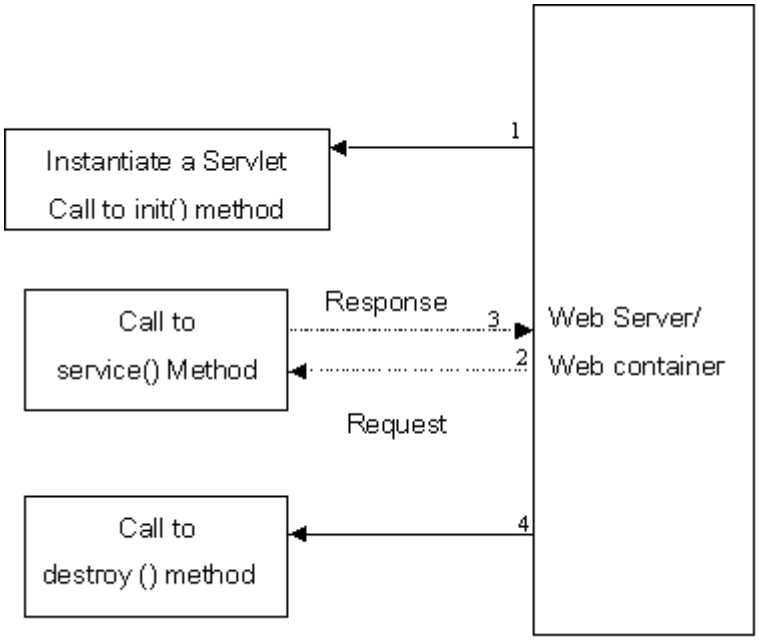
Un Servlet es un componente Java que puede ser instalado en un servidor para ampliar su funcionalidad: [64]

- Peticiones: HTTP
- Respuesta: HTML, XML o WML.

Se ejecutan dentro de un contenedor de servlets. (Tomcat es un contenedor de Servlets)

Tecnología Java que sustituye a la programación CGI. Situada en el nivel “módulos ejecutables”.

Especificación Servlet define un framework de programación Petición/Respuesta.



Funcionamiento de un Servlet

2.3.3.9. Java Server Pages (JSP).

JSP es un acrónimo de Java Server Pages, que en castellano vendría a decir algo como Páginas de Servidor Java. Es, pues, una tecnología orientada a crear páginas web con programación en Java. [64]

Con JSP podemos crear aplicaciones web que se ejecuten en variados servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java. Por tanto, las JSP podremos escribirlas con nuestro editor HTML/XML habitual.

El motor de las páginas JSP está basado en los servlets de Java -programas en Java destinados a ejecutarse en el servidor.

La especificación 2.0 de JSP introdujo una nueva librería estándar de etiquetas, denominada **JSTL**.

Estas etiquetas tratan de abstraer la complejidad de introducir código Java (scriptlet) dentro de JSP, del mismo modo que trata de evitar que cada equipo de desarrollo cree un juego de etiquetas no estándar para las mismas labores.

Dentro de estas etiquetas, se utiliza un lenguaje llamado EL, lenguaje de expresiones, que pretende ser un lenguaje más sencillo que Java, para realizar operaciones.

Uno de los primeros contenedores de JSP que soporta estas capacidades es Tomcat5.

Para poder usar las librerías de tags de JSTL, debemos descargarnos los ficheros binarios y registrarlos en el fichero web.xml.

2.3.3.10. Patrón Front Controller.

El patrón FrontController recomienda el uso de un Servlet controlador en la aplicación. Este servlet se encargará de realizar los controles comunes a todas las peticiones, identificar la acción asociada a cada petición, invocar la acción adecuada (modelo) y delegar en la vista que mostrará la respuesta.

Al igual que los servlets desarrollados en las prácticas anteriores, programamos el método init para que obtenga la factoría DAO de la aplicación. El método de procesamiento del servlet hace uso de una clase de apoyo (PeticionHelper) que será la encargada de determinar la acción necesaria para procesar la petición e instanciar el objeto Accion correspondiente. Seguidamente, el servlet invoca la acción que devuelve la URL de la vista que visualizará la respuesta. Finalmente llama a la vista para que visualice la respuesta. Este esquema es genérico y reutilizable para cualquier tipo de aplicación.

La clase PeticionHelper se encarga de analizar la petición para instanciar el objeto acción correspondiente. La estrategia más común para implementar esta funcionalidad es analizar la URI de la petición para determinar la acción e instanciar el objeto acción asociado. La asociación accion-base suele mantenerse en un fichero de propiedades y la instanciación del objeto accion se realiza mediante reflexión Java.

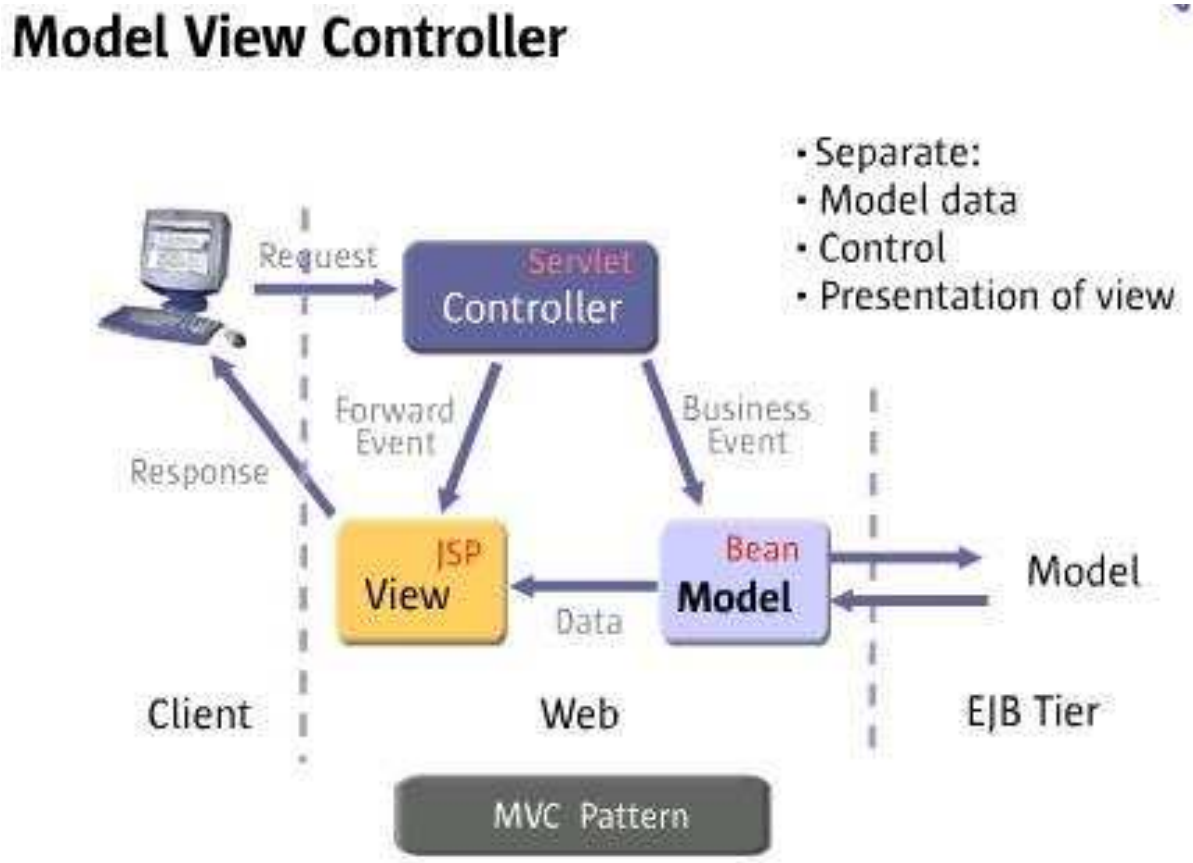
El controlador maneja objetos Accion que tiene un método ejecutar que toma como parámetros la petición y la respuesta del servlet, y que devuelve la URI (String) del recurso que visualizará el resultado (habitualmente una página JSP). Por tanto, las clases accion implementan la siguiente interface Java:

```
import javax.servlet.*;
import javax.servlet.http.*;

public interface Accion{

     public String ejecutar(HttpServletRequest peticion,
     HttpServletResponse respuesta, ServletContext aplicacion);

}
```



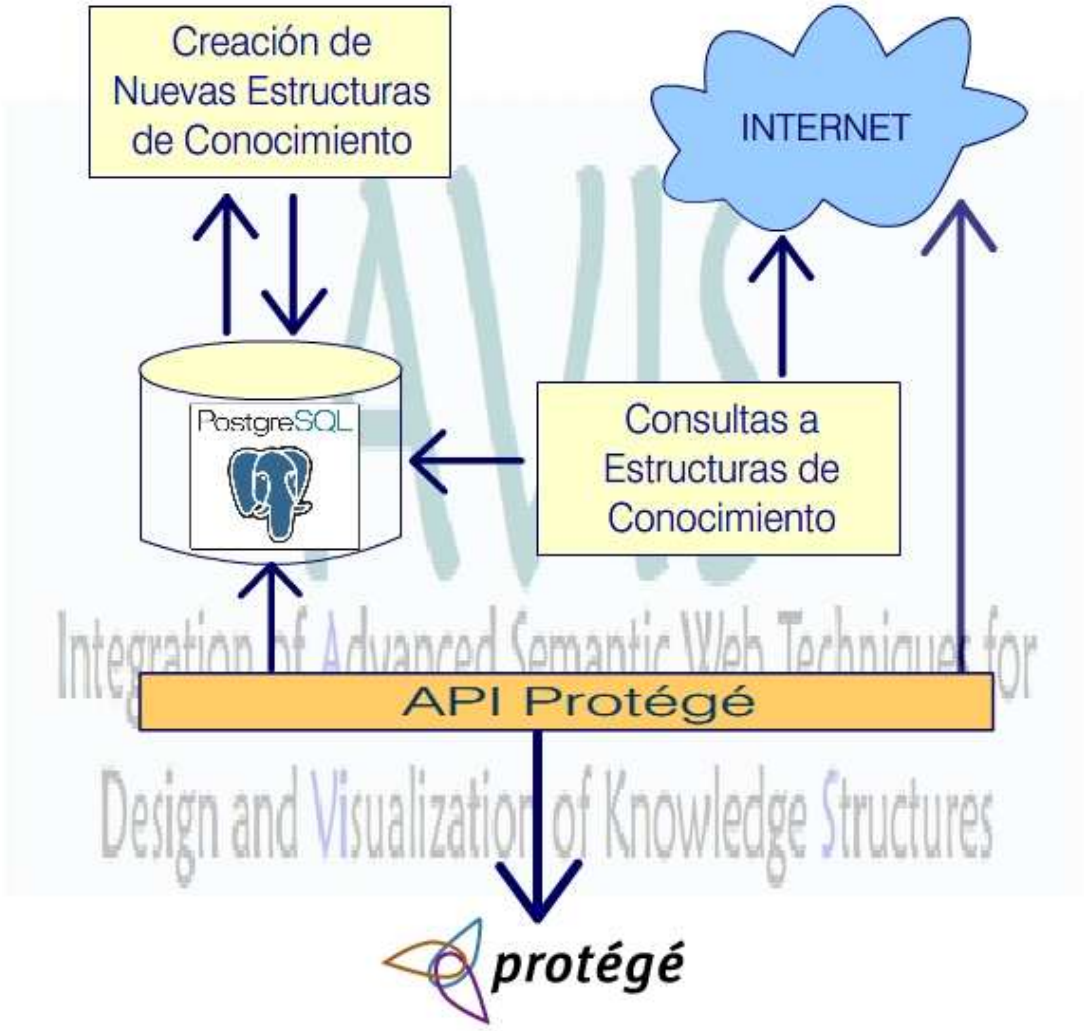
Esquema MVC en aplicación web (J2EE)
Esquema de funcionamiento de este proyecto

3. Diseño y Resolución del Trabajo Realizado

3.1. Arquitectura del sistema.

La arquitectura de este proyecto llamado AVIS (Integration of Advanced Semantic Web Techniques for Design and Visualization of Knowledge Structures) y en adelante tan sólo lo llamaré AVIS, lo conforman varias partes, por un lado tenemos un módulo de creación de nuevas ontologías o nuevas estructuras de conocimiento siguiendo la métrica descrita en el apartado 1.2.4.2, por otro lado tenemos el módulo de consultas a ontologías creadas en AVIS que se encontrarán en la base de datos del mismo, y también se podrán consultar ontologías remotas a partir de su URL correspondiente.

Aparte de estos módulos AVIS cuenta con una pequeña API para abrir las ontologías con Protégé, de este modo se puede comprobar visualmente el resultado de las consultas realizadas y la creación de nuevas estructuras de conocimiento. El poder establecer una conexión con ontologías remotas a través de su URL y poder abrirlas con Protegé es en AVIS uno de los puntos más importantes.



Arquitectura de AVIS

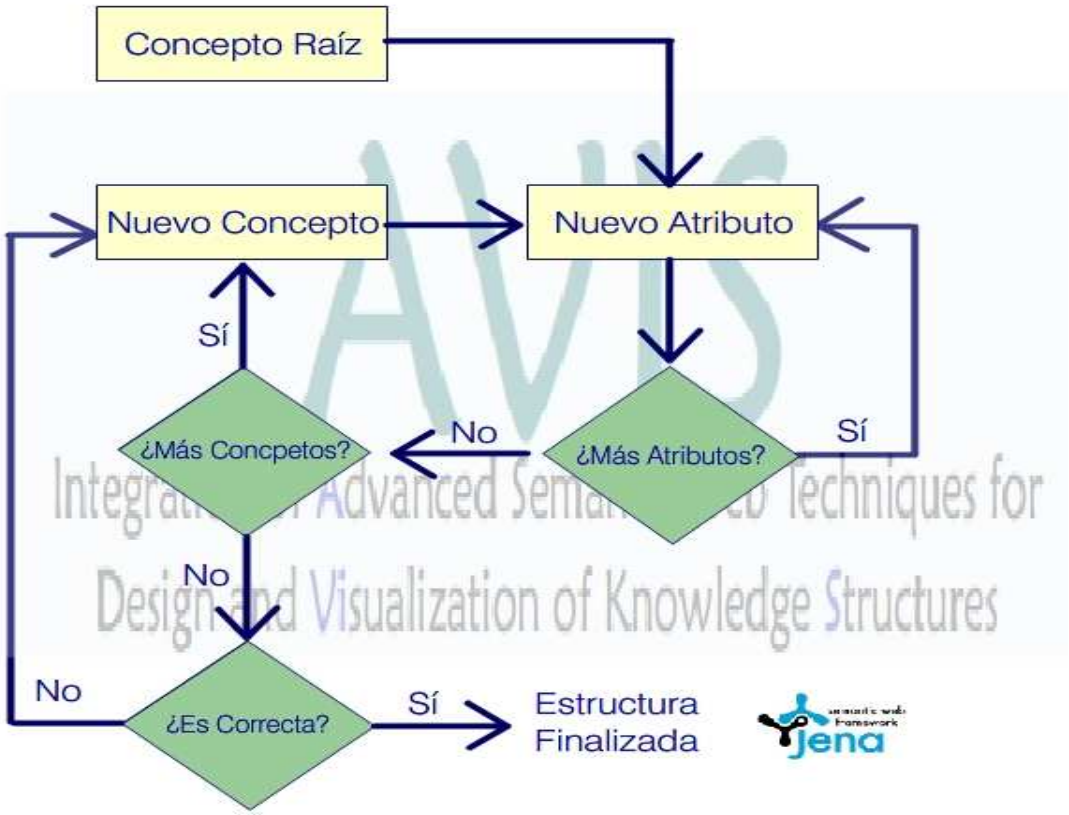
A continuación se explican cada una de las partes del diagrama anterior:

CREACIÓN DE NUEVAS ESTRUCTURAS DE CONOCIMIENTO

El usuario de la aplicación podrá crear desde cero tantas estructuras de conocimiento como desee. Para ello, comenzará dando un nombre a la nueva estructura, el cual no puede coincidir con los ya almacenados previamente en la base de datos. Una vez puesto este nombre el usuario creará el concepto raíz (Root Concept) con todos los atributos que necesite, y una vez finaliza la creación del Root Concept entonces deberá de crear los subconceptos y así sucesivamente.

Cuando se termine de crear un concepto, el sistema mostrará la estructura creada hasta el momento, a lo que se le llamará la ontología actual, y realiza una comprobación de si cumple con las métricas propuestas en este trabajo. Si las cumple se pude finalizar la ontología lo que la almacenaría en la base de datos y también se podrá ver directamente en el Protégé con la opción que hay disponible, si cumple con esta métrica pues no se puede finalizar hasta que no esté correcta.

Todo este proceso se puede conseguir con la herramienta JENA donde se creará el nuevo modelo y si irán añadiendo conceptos y atributos conforme el usuario lo decida y también se encargará JENA de almacenar esta estructura en la base de datos.



Creación de Nuevas Estructuras de Conocimiento

Como se puede apreciar el módulo de creación de nuevas estructuras de conocimiento consta de varias partes y cada una de ellas la explicamos a continuación. Antes de comenzar con el concepto raíz hay que decir que hay que ponerle un nombre a la ontología que estemos creando y éste no puede ser repetido, es decir, tras introducir el nombre se comprueba que no exista otra ontología almacenada en la base de datos que tenga el mismo nombre, en el caso de que coincidan se le muestra un mensaje al usuario para que introduzca un nombre distinto.

Concepto Raíz: Es el nombre de la clase raíz de la jerarquía y por tanto es el primer concepto que vamos a crear, le pondremos un nombre y a continuación le pondremos todos los atributos que tenga. Tras crear el concepto raíz ya solamente se podrán crear conceptos que sean hijos del concepto raíz porque toda ontología tiene un raíz única.

Nuevo Atributo: Es dónde crearemos los atributos del concepto que estemos formando. Debemos ponerle un nombre al atributo y elegir su tipo que puede ser: desconocido, cadena de caracteres, booleano, real, entero, entero positivo, entero negativo, y entero positivo más cero. Dependiendo del tipo elegido deberemos introducir una cadena de caracteres, verdadero o falso, un número real, o números enteros, para los número además podemos seleccionar una restricción como mayor o igual, menor o igual, entre 2 valores o simplemente podemos no introducir ninguna restricción. Para los tipos desconocidos no se introduce ningún dato adicional.

Una vez finalizado un atributo podemos seguir creando más atributos para este mismo concepto, o podemos pasar de crear nuevos atributos y crear un concepto nuevo.

Nuevo Concepto: Es dónde creamos todos los demás conceptos que no sean el raíz, el primero que creamos después del raíz siempre será hijo del raíz y después ya se pueden ir creando hijos de los que hayamos creado. Su funcionamiento es similar al del concepto raíz, se comprueba que el nombre no esté repetido y una vez creado se heredan todos los atributos del padre a los que debemos de ponerle un valor válido y si se desea se pueden añadir nuevos atributos en concreto a parte de los heredados. Todos los conceptos que se crean en este módulo heredan los atributos del concepto padre cualquiera que sea y los valores nuevos introducidos en este atributo tienen que cumplir con las restricciones impuestas en su creación anterior.

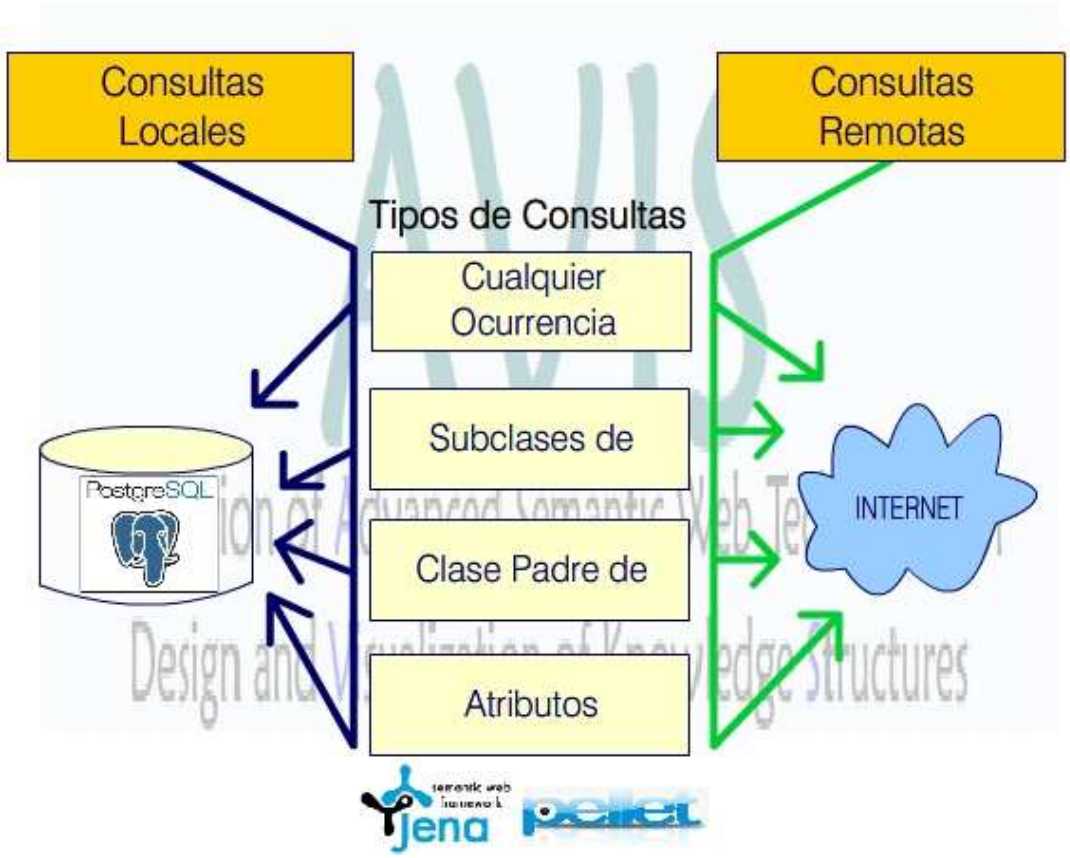
Comprobación: Cuando terminemos de crear toda la jerarquía de conceptos el sistema comprobará si la estructura creada es correcta o no, es decir, si cumple o no con las métricas explicadas en este trabajo. Para ello comprueba que todos los conceptos tengan al menos 2 subconceptos y que todos los atributos cumplen con las restricciones creadas en niveles anteriores al que están además de que tengamos una única raíz. Si todo es correcto el sistema nos deja finalizar y almacenar la estructura, y a parte también nos permite verla en Protégé con tan sólo pulsar un botón.

De esta manera podemos ver como AVIS por si sólo es capaz de crear una ontología de una forma sencilla, manualmente, sin necesidad de usar Protégé que es un software complejo, y además permite ver cómo queda la ontología en Protégé que es un punto que resulta muy atractivo.

CONSULTAS A ESTRUCTURAS DE CONOCIMIENTO

A parte del módulo anterior de creación de estructuras de conocimiento AVIS también nos permite realizar consultas de las ontologías que hayamos creado y almacenado correctamente. Estas consultas pueden ser de 4 tipos como vemos en la siguiente figura y explicaremos a continuación. Además de estas consultas se puede ver todas las estructuras de conocimiento que tenemos almacenada y se pueden eliminar las que se quieran si así se desea, y como no, se pueden ver con Protégé con tan sólo pulsar en la opción disponible para ello.

Por otro lado AVIS también es capaz de realizar este tipo de consultas a estructuras de conocimiento remotas, es decir, ficheros .owl que se encuentren alojados en otros servidores de internet mediante su dirección o URL, para ello, AVIS dispone de un módulo de entrada de URL's, modificación y eliminación que se puede tener actualizado y acepta las URL's que se quiera. Cuando se quiera realizar una consulta a una ontología remota el sistema se conecta a la URL indicada (pueden ser varias) y muestra los resultados en pantalla al igual que las consultas locales. También se pueden visualizar las ontologías remotas con Protégé.



Consultas a Estructuras de Conocimiento

Veamos a continuación en qué consisten cada una de las consultas:

Cualquier Ocurrencia: Muestra en Pantalla cualquier concepto o atributo que contenga completa o parcialmente la palabra introducida de búsqueda. En pantalla se mostrará la estructura de conocimiento a la que pertenece, y el concepto y atributo de cualquier en cuestión o simplemente el concepto.

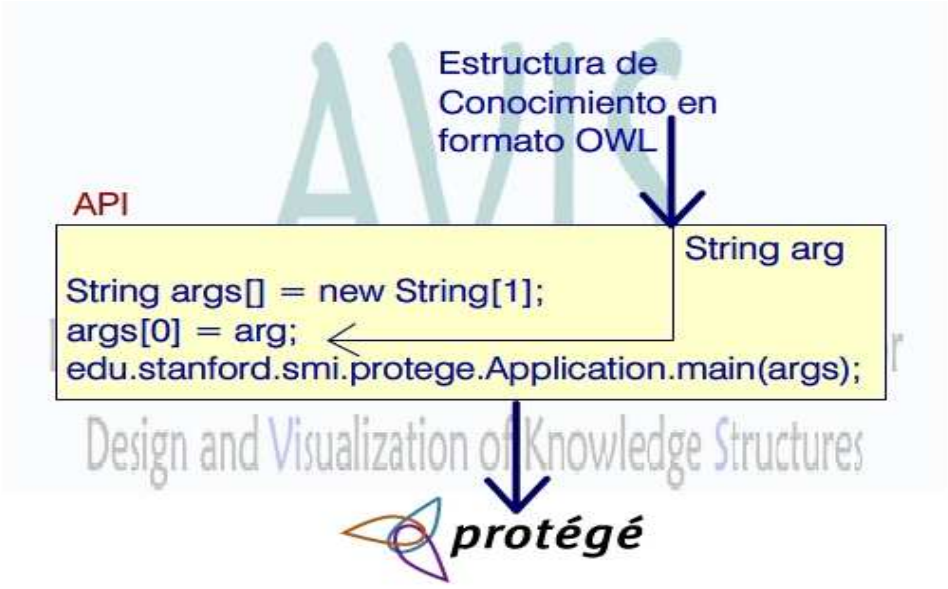
Subclases de: Muestra en pantalla todos los conceptos hijos del concepto introducido en la búsqueda. Este concepto no se teclea en las consultas locales, sino que simplemente se selecciona de un desplegable que se le ofrece al usuario, mientras que en la consultas remotas hay que escribir todas las palabras de búsqueda. En pantalla se muestra la estructura de conocimiento en cuestión y cada uno de los conceptos hijos.

Clase Padre de: Muestra en pantalla la clase que es padre de una en concreto, en el caso de que se introduzca una clase raíz no se muestra ninguna otra clase. En esta consulta pasa lo mismo que en la anterior, se selecciona en las locales y se escribe en las remotas.

Atributos: Muestra en pantalla los conceptos que contienen un determinado atributo que se introduce para buscar. En el caso de que se encuentre algún atributo llamado igual se muestra por pantalla la estructura de conocimiento en cuestión, y el concepto o varios conceptos que contengan dicho atributo. En esta consulta hay que teclear la palabra a buscar tanto en las consultas locales como remotas.

Cuando se realicen las consultas se puede comprobar con Protégé los resultados que obtenemos tanto en las locales como en las remotas con las opciones que hay disponibles.

API PROTÉGÉ



API Protégé

Gracias a este pequeño módulo es posible comprobar con Protégé las ontologías locales (las creadas en AVIS) y las ontologías remotas (las que tengamos URL dada de alta). Tal y como se ve en la figura anterior tan solo le introducimos la ruta del fichero .owl y listo.

Debido a que AVIS es una aplicación web y por tanto tiene que estar en un servidor de aplicaciones, esto hace que Protégé tenga que estar también en este servidor, de hecho Protégé va en el mismo código fuente que AVIS, es simplemente un librería de java importada. Esto provoca que si utilizamos AVIS desde un servidor remoto no podamos ver en la máquina cliente la ejecución de Protégé en cualquiera de sus ejecuciones y que para verlo es necesario tener activado un acceso por Tunneling sobre SSH (paquetes IP dentro de otros paquetes IP). Si estamos en un servidor local podremos ver Protégé sin problema alguno.

3.2 Ontología Basada en Genética.

Respecto a ontologías basadas en biología molecular hay que destacar que la ontología genética se ha convertido en una de las herramientas estándar de la bioinformática. El origen de la ontología genética se remonta a 1988 por un consorcio de investigadores que estudiaban el genoma de tres organismos modelo: *Drosophila melanogaster* (mosca del vinagre, o mosca de la fruta), el *Mus musculus* (ratón casero, o ratón de laboratorio), y el *Saccharomyces cerevisiae* (levadura).

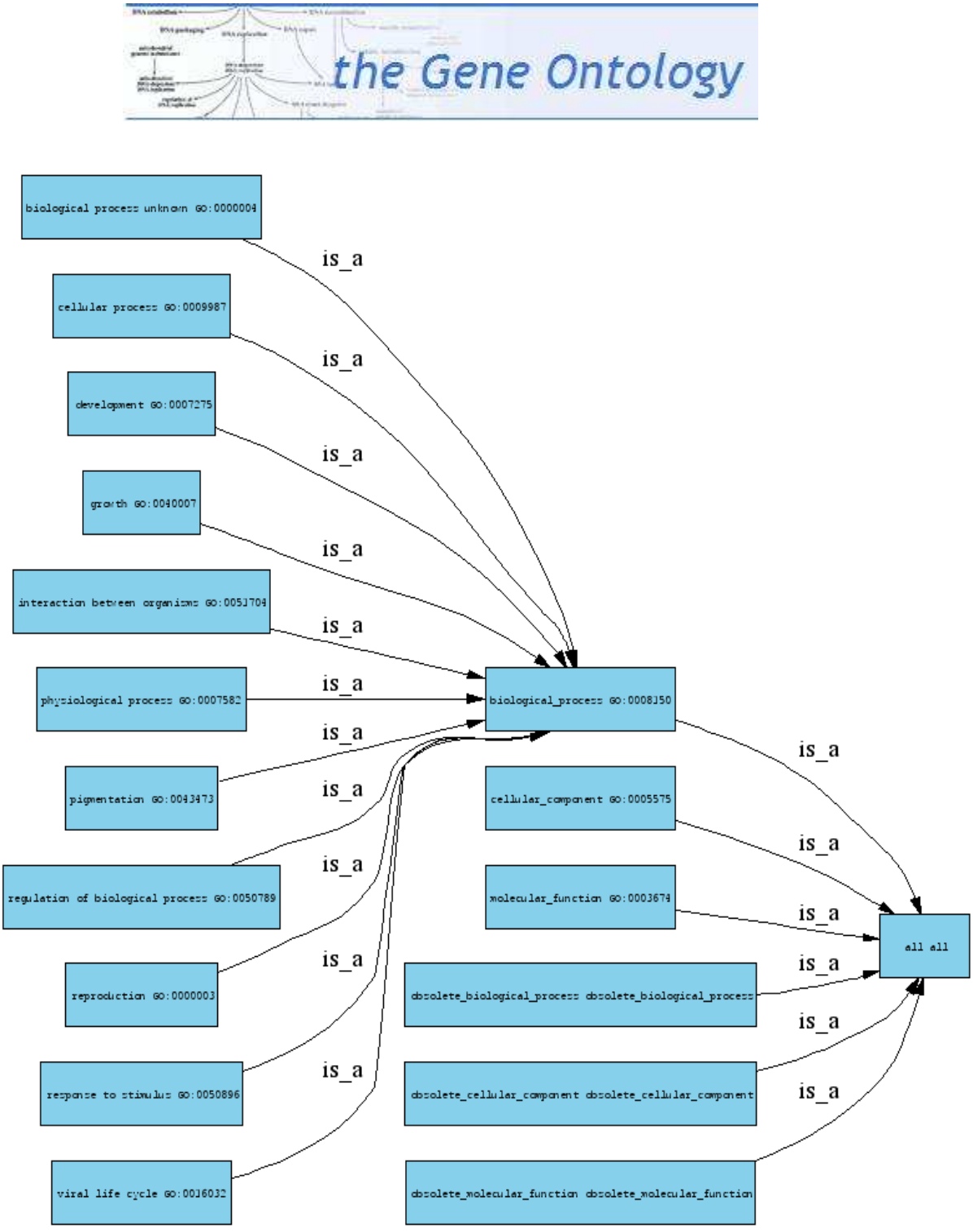
Nació entonces el proyecto **THE GENE ONTOLOGY [**63] que se ha convertido en la mayor ontología hay en la actualidad sobre biología molecular de cualquier organismo. El proyecto **Ontología Genética** (**Gene Ontology**, **GO**), provee un vocabulario controlado que describe el gen y los atributos del producto génico en cualquier organismo. Puede ser, en general, dividido en dos partes. La primera es la ontología por sí misma; en realidad son tres ontologías, cada cual representando un concepto clave en biología molecular: la **función molecular** de los productos génicos; su rol en los **procesos biológicos** de múltiples direcciones; y su localización en **componentes celulares**. Las ontologías son continuamente actualizadas, y se dispone de nuevas versiones mensualmente. La segunda parte es la anotación, la caracterización de los productos génicos usando términos de la ontología. Los miembros del Consorcio GO entregan su información y se hace disponible públicamente a través del sitio web GO www.genontology.org.

Cada término GO consiste en un único identificador alfanumérico, un nombre común, sinónimos (si son de aplicación), y una definición. Cuando un término tiene múltiples significados dependiendo de las especies, el GO usa una etiqueta "sensu" para diferenciarlos entre ellos. Los términos son clasificados en sólo una de las tres ontologías, las cuales están estructuradas, cada una de ellas, como un grafo dirigido acíclico.

Nuevos términos y anotaciones son sugeridas por miembros de las comunidades de investigación y anotación. Una vez publicados, son revisados por miembros del consorcio GO para determinar su aplicabilidad.

Si se decide que un término de la ontología no es apropiado, es desaprobado, o marcado como "obsoleto". Esto puede pasar por diferentes razones, como estar fuera del objetivo de la ontología o estar engañosamente nombrado o definido.

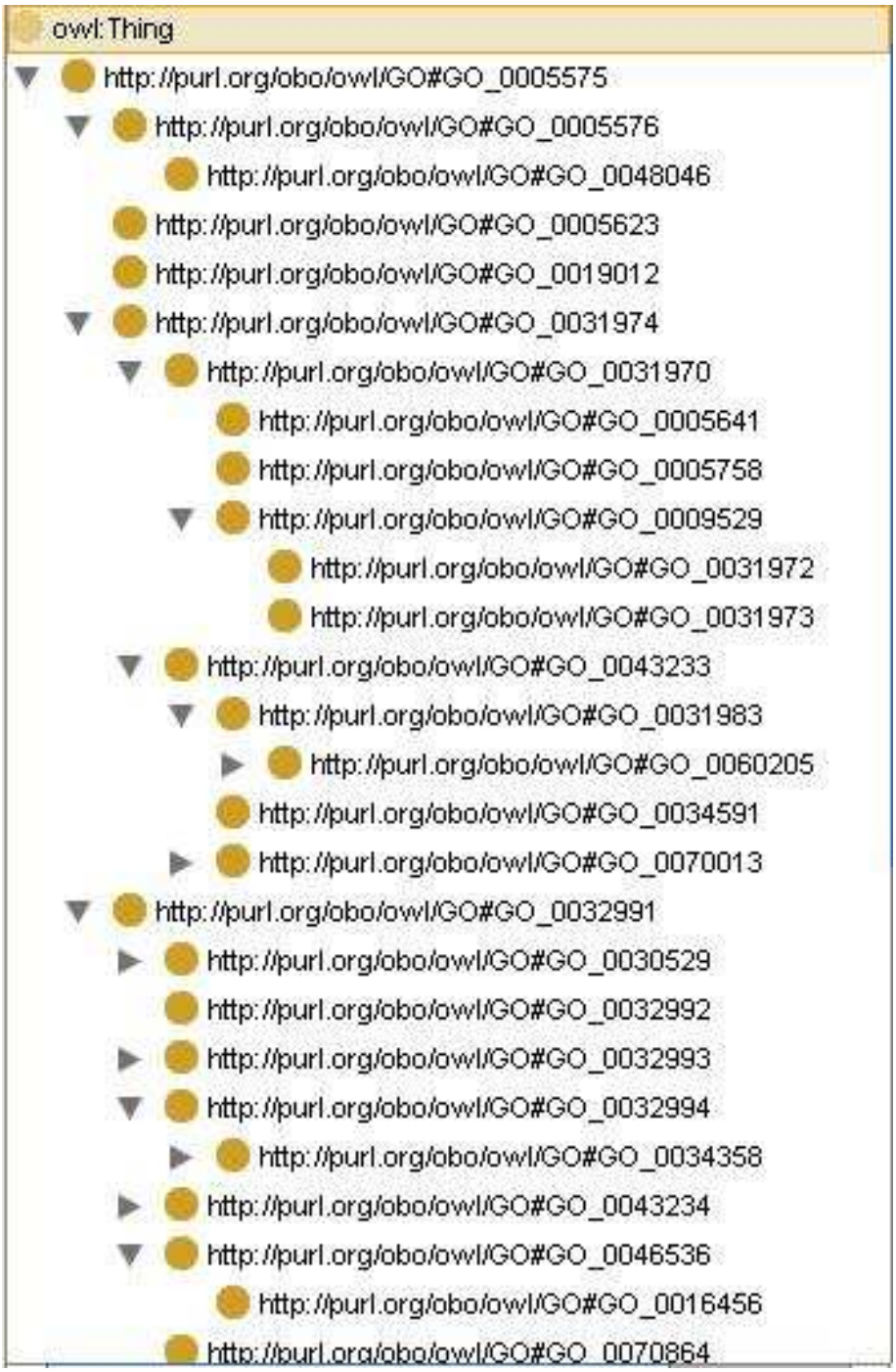
En este proyecto se ha trabajado con una de las ontologías más importantes que hay como es GO. Más concretamente en la parte de consultas remotas se puede introducir la URL de GO por ejemplo la de componente celular “celular_component” y se pueden realizar consultas sobre sus conceptos y atributos. De esta manera AVIS resulta una herramienta muy útil para biólogos expertos en el tema para buscar posibles modificaciones de GO por parte del consorcio y realizar sus investigaciones en este campo con una mayor facilidad.



El nivel más alto de Gene Ontology

En la figura anterior podemos ver el nivel más alto de la gene onotolgy y podemos comprobar como los conceptos son identificadores del tipo "GO_CifraAlfanumérica" y cada uno de estos lleva asociado una definición, y un sinónimo si son de aplicación.

Dentro de la gigantesca GO podemos encontrar estructuras más sencillas que forman parte de ella, que en su día fueron estructuras independientes de organismos que se han unido al consorcio GO aportando una mayor información. A continuación podemos ver una parte de "Gene Ontlogy" vista desde Protégé, más concretamente es la porción de "Cellular_Component" que corresponde al GO_0005575.

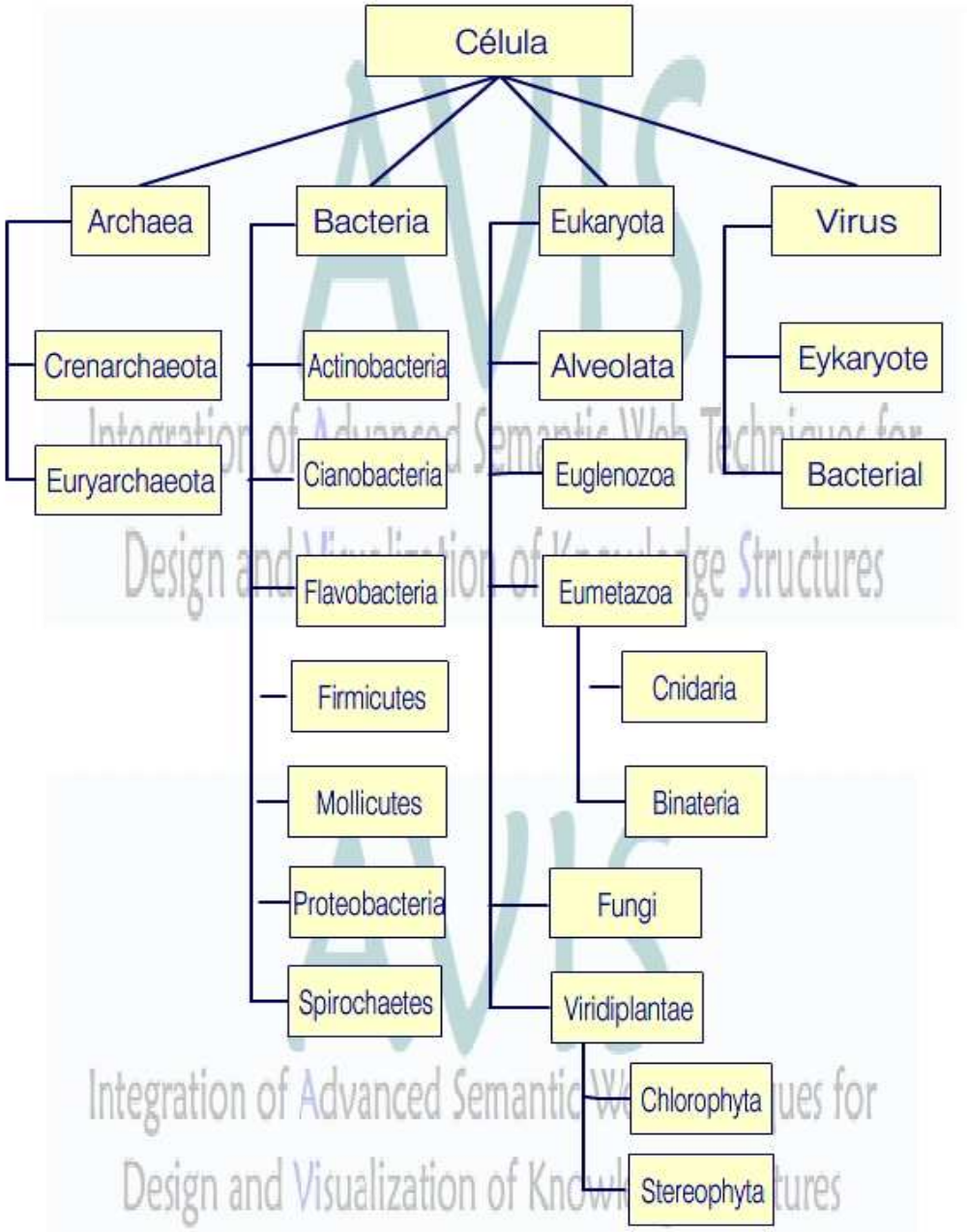


Comienzo de Cellular Component GO_0005575 desde Protégé

3.3 Ontología Basada en Biología Molecular.

Ligado a la gran ontología “the gene ontology” se encuentra una ontología más pequeña llamada FBsp.owl recogida de la web www.obofoundry.org/cgi-bin/detail.cgi?id=**fly_taxonomy** [62] que se ajusta bastante bien para mostrar un ejemplo del modulo de creación de ontologías de AVIS.

Este estructura trata sobre una clasificación de tipos de células como pueden ser virus y bacterias y los atributos que hacen posibles estas distinciones. A continuación vamos a ver con más detalle esta estructura.



Ontología basada en biología molecular

Los atributos son los siguientes:

- *Has_Nucleus*: Existen dos tipos de Células, las que tienen núcleo y las que no tienen núcleo. Por tanto el rango para este atributo solo podrá tomar los valores True o False.

- *Needs_Host_Replication*: Este atributo sirve para diferencia a los virus de los demás tipos de Células. El rango de este atributo toma como en el anterior los valores True o False.

- *Susceptible_to_antimicrobial_agents*: Este atributo sirve para diferenciar a las Bacterias de las Células Archaeas ya que las dos forman parte de la familia de las Células Procariotas. El rango para este atributo por lo tanto será valores True o False.

Con estos tres atributos se consiguen distinguir perfectamente los cuatro conceptos que derivan de la raíz Célula como se comprueba a continuación.

El siguiente concepto para el cual se buscaron los atributos es **Archaea**. En primer lugar se buscó los atributos que se heredaron del padre para comprobar que estaban bien elegidos. Buscando en Wikipedia el término Archaea, se puede comprobar que estas células no tienen núcleo, no necesitan un cuerpo para replicarse y son susceptible a agentes antimicrobiales. Por lo tanto se le puede dar valor a los tres atributos que tiene que heredar de su padre. Seguidamente, se pasó a buscar atributos para poder diferenciar a los conceptos hijos de la Archaea, que en este caso son **Euryarchaeota** y **Crenarchaeota**.

El atributo encontrado para distinguir estos dos tipos es:

- *Optimun_PH*: Este atributo indica el PH óptimo para que ciertas células puedan vivir. El rango que toma es un valor Real.

Con este atributo se consigue distinguir los dos tipos de Células **Archaeas** que contiene la Taxonomía. El siguiente paso es encontrar los valores de este atributos anterior para poder distinguir a los dos tipos de Células y poder ver que la Ontología no esta mal construida. Por un lado, el tipo **Euryarchaeota** tiene un valor de PH optimo de 0.7 y el tipo **Crenarchaeota** lo tiene entre 1 y 4 por lo que están bien distinguidas.

Una vez solucionada esta rama de la ontología se pasó a buscar atributos para poder distinguir a los hijos del concepto **Eukaryota**. Este concepto tiene siete hijos: **Mycetozoa, Parabasalidea, Alveolata, Euglenozoa, Eumetazoa, Fungi, Viridiplantae.** Buscando de nuevo en Wikipedia en primer lugar se buscan los atributos que Eukaryota heredó de Célula para comprobar que los cumple. Una vez buscados se llega a la conclusión que tiene núcleo, no necesita un cuerpo para replicarse y no son susceptibles a agentes antimicrobiales. Seguidamente se pasó a buscar los atributos para poder diferenciar a los hijos de este concepto.

Los atributos encontrados son los siguientes:

- *Has_Plasmodio*: Este atributo indica si un concepto tiene plasmodio (agregado en forma de masa gelatinosa). El rango de posibles valores es True o False.

- *Has_Flagelo*: Indica si un concepto tiene o no Flagelo (apéndice movible con forma de látigo).

- *Has_Alveolos_Corticales*: Distingue a los conceptos que tienen alvéolos corticales de los que no los tienen.

- *Has_Cytostome*: Este atributo diferencia a los conceptos que tienen citostoma (abertura por donde entran las partículas alimenticias). El rango de posibles valores es True o False.

- *Has_Tissue*: Atributo para distinguir a los conceptos que pueden tejer y usar tejidos.

- *Has_Cell_Wall*: Indica que los conceptos que tienen el valor Sí tienen pared celular mientras que los que tienen el valor No carecen de ella.

- *Has_Cloroplast*: Sirve para distinguir a los conceptos que tienen cloroplastos de los que no los tienen.

Con todos estos atributos se podría perfectamente diferenciar a todos los conceptos hijos de **Eukaryota**, como se verá a continuación.

El primer concepto a comprobar en Wikipedia es **Mycetozoa**. Buscando posibles atributos se descubre que tiene plasmodio, no tiene flagelos, no tiene alvéolos corticales, tampoco tiene citostoma, no utiliza tejidos, no tiene pared celular, y tampoco cloroplastos.

El siguiente concepto es **Parabasalidea**. Sus características son que no tiene plasmodio, tiene flagelo, no contiene alvéolos corticales, no tiene citostoma, no utiliza tejidos, no posee pared celular y tampoco cloroplastos.

Siguió a estos conceptos la **Alveolata**. Buscando en Wikipedia se llegó a la conclusión que sus atributos son que no tiene plasmodio, si tiene flagelo, si tiene alvéolos corticales, no tiene citostoma, no utiliza tejidos, no posee pared celular y tampoco cloroplastos.

Las características del concepto **Euglenozoa** son que no tiene plasmodio, si tiene flagelo, no tiene alvéolos corticales, tiene citostoma, no utiliza tejidos, no contiene pared celular y si tiene cloroplasto. Además se encontró otro atributo:

- *Size*: Es el tamaño que pueden tomar estos organismos, cuyo rango de valores se sitúa entre 15 y 40 micrómetros.

El siguiente concepto a buscar sus características fue **Eumetazoa**. Las características principales son que tiene plasmodio, no tienen flagelos, no poseen alvéolos corticales, no tienen citostoma, utilizan tejidos, no contienen pared celular y tampoco cloroplastos. Además se encontró otro atributo que podría ser útil para poder diferencias a los hijos de este concepto. El atributo es:

- *Type_of_simetry*: Es el tipo de simetría que pueden tener estos organismos. Puede tomar valor Radial o Bilateral.

El siguiente concepto de esta rama es **Fungi**. Buscando en Wikipedia se llegó a la conclusión que las principales características de este tipo de organismos es que no tiene plasmodio, tampoco flagelos, ni alvéolos corticales, no posee citostoma, no utiliza tejidos, si posee pared celular y no tiene cloroplastos.

Por último el último concepto de ese nivel en la rama que se esta explicando es **Viridiplantae**. Para este concepto se detectó que sus características son que no tiene plasmodio, si tienen flagelo, no poseen alvéolos corticales, no tiene citostoma, no utilizan tejidos, si poseen pared celular y también cloroplastos. Por último y para poder diferenciar a los conceptos hijos de este se encontró otro atributo:

- *Nº_of_living_species*: El número de especies de cada subtipo. Se encontró un rango significativo de valores naturales dentro del cual se encuentran el número de especies de cada subtipo que hay en la naturaleza aproximadamente. Este rango está comprendido entre [3800,6000] especies, dentro de este rango están contenidos todos los subtipos.

Continuando por la misma rama, se encuentran dos conceptos hijos de Eumetazoa, que son: Cnidaria y Bilateria. Estos dos conceptos además de todos los atributos heredados del padre tienen un atributo comentado anteriormente llamado Type_of_simetry. Buscando ambos concepto en Wikipedia se llegó a la conclusión que Cnidaria tenía valor Radial y Bilateria tomaba como valor Bilateral.

Para concluir con esta rama el concepto **Viridiplantae** también tiene dos conceptos hijos, que son: Streptophyta y Chlorophyta. Para poder distinguir a estos dos conceptos se introdujo el atributo comentado más arriba Nº_of_living_species. Para este atributo el concepto Streptophyta toma el rango [4000, 6000] y el concepto Chlorophyta [3800, 3999]. Por lo que estos dos conceptos quedan totalmente diferenciados.

A continuación se pasó a ver la rama que deriva de la **Bacteria**. En primer lugar se busco este concepto en la Wikipedia para poder averiguar el valor de los atributos heredados del concepto padre Célula. Gracias a la búsqueda en Internet se encontró que la Bacteria no tiene núcleo, tampoco otro cuerpo para replicarse y no son sensibles a agentes antimicrobianos. Por otro lado también se buscaron posibles atributos para poder diferenciar a los hijos de este concepto. Entre ellos se encontraron:

- *Gram_Type*: Indica el tipo de Gram que poseen los organismos. Puede tomar los valores Positivo o Negativo.

- *Repiration*: Indica el tipo de respiración que tienen los organismos. Toma valor Aeróbico o Anaeróbico.

Seguidamente se buscaron los posibles valores para los atributos heredados del padre. Enprimer lugar se buscó para la Cianobacteria. Para este concepto el atributo Gram_Type toma valor negativo y para Repiration valor aeróbico.

El siguiente concepto es **Flavobacteria**. Este concepto para sus atributos heredados toma valor negativo y aeróbico. Seguidamente se prosiguió con Actinobacteria que toma el valor positivo y aeróbico.

Se prosiguió con el concepto **Firmicutes**. Este concepto toma como valores de lo atributos heredados positivo y para el atributo Respiration puede tomar los valores {anaeróbico o aeróbico}. Por otro lado también se ha conseguido encontrar otro atributo que será de gran utilidad a la hora de comprobar si esta ontología ha sido creada con un buen criterio. El atributo es:

- *Length*: Indica el tamaño que tiene un organismo **Firmicutes**. Puede tener un rango de valores entre {0.2 o 0.3} micrómetros.

Como siguiente concepto a buscar sus atributos se tomó **Mollicutes**. Los atributos de este concepto toman como valor, positivo y aeróbico. Para este concepto también se encontró el mismo atributo que en concepto anterior. En este caso toma un valor {0.2 o 0.3} micrómetros.

Los dos últimos conceptos de esta rama son: **Proteobacteria** y **Spirochaetes**. Estos conceptos toman ambos para el valor de sus atributos heredados el valor negativo y anaeróbico.

Por último, queda por explicar los atributos de la rama de **Virus**. Primero se buscó en la Wikipedia los valores de los atributos heredados del padre de este concepto, Célula. Una vez buscado se encontraron los valores para estos atributos. Un Virus no tiene núcleo, si necesita otro cuerpo para poder replicarse y no son sensibles a agentes microbianos. Para poder diferenciar a los conceptos hijos de este se ha encontrado un atributo:

- *Affect_cell*: Indica a que tipo de célula ataca este virus. Puede tomar los valores Eukaryota o Bacteria.

Los conceptos hijos de Virus son: **Eukaryota virus**, **Bacterial virus**. El primero toma el valor Eukaryota para el atributo heredado del concepto padre, mientras el segundo toma como valor Bacteria.

Con todo lo anterior, la ontología queda totalmente creada. Es indiscutible que el diseño y construcción de una ontología es un proceso complejo que requiere de mucho tiempo y esfuerzo.

3.4 Metodología para la creación de estructuras de conocimiento.

Al igual que se explicó en el aparado de arquitectura del sistema el módulo de creación de estructuras de conocimiento consta de varias partes. Gracias a JENA podemos ir construyendo la ontología paso a paso. A continuación veamos cuales son las partes de este módulo y las Clases Java que se utilizan.

Crear Nueva Estructura de Conocimiento

Para este apartado se usa la Clase OntModel y ModelFactory de JENA, simplemente se comprueba que dicha ontología no exista previamente y se crea con la factoría correspondiente:

```
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.ModelFactory;

private final String BASE =
"http://www.owlontologies.com/unnamed.owl";

private OntModel modelo;

public void crearNuevaOntologia(String nombreOntologia) {

      if (!this.existeOntologiaPrevia(nombreOntologia)) {

            this.modelo = ModelFactory.
            createOntologyModel(OntModelSpec.OWL_DL_MEM);
            this.modelo.createOntology(BASE);
      }
}
```

Crear Nuevo Concepto

Distinguimos en este apartado 2 casos, el primer concepto de la estructura que es el concepto raíz y los demás conceptos que son todos los demás. Todos ellos son de la misma clase y se crean con un método de la clase OntModel vista en el apartado anterior llamado createClass.

```
this.modelo.createClass(URI + nombreConcepto);
```

Antes de ejecutar la instrucción anterior se comprueba que no exista un concepto llamado nombreConcepto con un método como el siguiente:

```
public boolean checkNameConceptoEnOntologia(String nameConcepto) {
	if (this.modelo.getOntClass(URI + nameConcepto) != null)
		return false;
	else
		return true;
}
```

Crear Nuevo Atributo

Este apartado es más complicado que los anteriores porque es dónde se pone el tipo de cada atributo y las restricciones que se le indican a cada atributo. AVIS permite poner a los atributos los siguientes tipos:

- *Desconocido*: En este caso el usuario desconoce el tipo de rango que tiene el atributo para cierto Concepto.
- *Cadena de Caracteres*: El tipo del atributo será una cadena de caracteres.
- *Booleano***:** True o False.
- *Reales***:** El tipo de atributo en este caso serán los números Reales tanto positivos como negativos ya que OWL no los distingue como hace con los números Enteros.
- *Enteros*: El tipo de atributo en este caso serán todos los números enteros, tanto positivos como negativos.
- *Enteros Negativos*: En este caso el rango del atributo estará compuesto por tan solo números enteros negativos. Es decir desde (-∞,0).
- *Enteros Positivos*: En este caso el rango del atributo estará compuesto por tan solo números enteros positivos. Es decir el rango: (0, ∞).
- *Enteros Positivos y Cero*: En este caso el rango del atributo estará compuesto por el cero y todos los números enteros positivos. [0, ∞).

La clase de JENA que representa a los atributos de los tipos anteriores es DataTypeProperty, y la clase que representa a los tipos anteriores es Resource.

Una vez seleccionado el tipo de atributo se pasa a la pantalla de elección del tipo de restricción que puede ser:

- *No hay restricción:* No se aplica ninguna restricción y el valor del atributo puede ser el que sea compatible con el tipo del atributo.

- *Mayor o igual que:* Se indica una cota inferior, el valor del atributo en los hijos debe ser mayor o igual a éste.

- *Menor o igual que:* Se indica una cota superior, el valor del atributo en los hijos debe ser menor o igual a éste.

- *Entre 2 valores:* Se indican 2 cotas una inferior y otra superior, debe existir en los hijos la representación del todo el rango completo comprendido entre estos 2 valores para que la ontología sea correcta.

La clase de JENA utilizada para indicar las restricciones es HasValueRestriction.

Comprobación

Se comprueba que la estructura creada esté bien categorizada y además se comprueba que todos los conceptos tengan al menos 2 subconceptos. La correcta categorización de una estructura de conocimiento quiere decir que todos los valores con rango de valores posible de los atributos de los conceptos en cuestión estén representados en el conjunto de sus hijos. Por ejemplo si un concepto tiene un rango del tipo naturales tiene los valores {4,5,7,8}, entre todos sus hijos tiene que estar representado. Si este concepto tiene tres hijos, uno puede tener como rango {4,5}, otro {7} y el tercero {8}, de esta forma la estructura estaría bien categorizada. En cambio si nos falta algún valor por ser representado tenemos una estructura mal categorizada y por tanto AVIS no permite su finalización.

Finalizar Eestructura y Lanzar Protégé

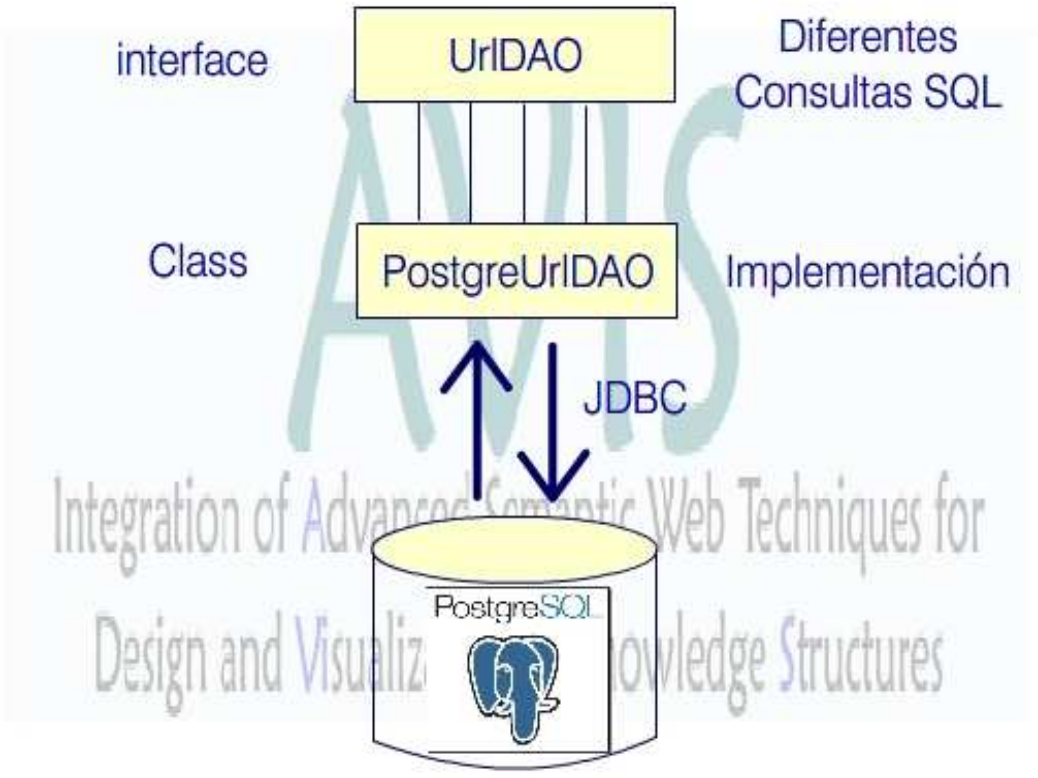
Una vez comprobada que la estructura creada es correcta entonces se permiten usar 2 opciones, una de ellas es finalizar la estructura y lo que se hará es almacenar la ontología creada en base de datos utilizando la un método de JENA que almacena en la base de datos del sistema y ya genera por si solo las tablas que sean necesarias. La otra opción es lanzar la estructura que acabamos de crear con Protégé, esta opción también almacena en la base de datos la estructura y abre el Protégé usando la API de AVIS para tal efecto.

3.5 Metodología para consultar estructuras de conocimiento.

Al igual que se explicó en el aparado de arquitectura del sistema el módulo de consultas de estructuras de conocimiento consta de varias consultas posibles a realizar a estructuras creada en AVIS y que se encuentren en su base de datos y también como ya sabemos se pueden realizar consultas a estructuras de conocimiento remotas como es el caso de gene ontology.

Las Consultas Remotas se realizan sobre estructuras remotas se registran en la base de datos con sus direcciones URL, se pueden registrar tantas URL como se quiera, en esta caso se realiza un Acceso a la Base de Datos con el patrón DAO (Data Acces Object) que consiste en una Factoría Abstracta implementada en todos aquellos objetos que se quieran almacenar.

El patrón DAO consta de una interface donde se declaran los métodos de consultas, inserción, eliminación, y modificación de registros, y una clase concreta donde se implementan. Véase la siguiente figura.



Patrón DAO con la Clase Url

Uno de los problemas más importantes encontrados a la hora de desarrollar esta parte es la **asignación de una clave primaria a la hora de insertar las URL's** en la base de datos. En realidad una URL no tiene un identificador único y cuando esto sucede debemos crear un mecanismo de asignación de identificadores únicos formado por la IP del cliente, la hora, minuto y segundo en el que se realiza la inserción, y un número aleatorio. Con estas tres variantes podemos construir un identificador asegurando que no se produzca ninguna repetición del mismo.

Este procedimiento de generación de claves primarias se conoce como Generador de Claves UUID y está desarrollado siguiendo el patrón Singletón y de esta manera su utilización es tan sencilla como se muestra en la siguiente línea:

```
GeneradorClaves.getInstance().getId();
```

A continuación veamos más en detalle las clases JENA que se han utilizado para realizar los 4 tipos de consultas.

CONSULTA 1.

Cualquier Ocurrencia de: Esta consulta consiste en buscar en las estructuras establecidas de búsqueda (todas las locales o varias remotas) los conceptos y atributos cuyos nombres contengan el patrón de búsqueda introducido.

Para las consultas locales se hace un recorrido de todas las estructuras almacenada en la base de datos y si buscan conceptos y atributos que contengan dicho patrón. Veamos las clases involucradas:

```
listModels(); // Para listar todas las estructuras almacenada en BD.

openModel(nombre); // Para abrir una estrcutura

ont.listNamedClasses(); // Para listar todos los conceptos.

clase.getLocalName.contains(patron);
```

La clave de esta consulta consiste en que no se busca un patrón exacto, sino que se busca que un concepto contenga un patrón de búsqueda.

Para los atributos hay que realizar una búsqueda para cada tipo de atributo que según OWL pueden ser de 7 tipos y en JENA estas clases se listan de la siguiente manera:

OntModel ont;

- *DatatypeProperty*

  ont.listDatatypeProperties();

- *ObjectProperty*

  ont.listObjectProperties();

- *FunctionalProperty*

  ont.listFunctionalProperties();

- *InverseFunctionalProperty*

  ont.listInverseFunctionalProperties();

- *AnnotationProperty*

  ont.listAnnotationProperties();

- *SymmetricProperty*

  ont.listSymmetricProperties();

- *TransitiveProperty*

  ont.listTransitiveProperties();

CONSULTA 2.

Tipos / SubClases de: Con esta consulta podemos comprobar todos los hijos de un determinado concepto, en el caso de que no tenga ninguno se muestra por pantalla un mensaje de que no se han encontrado conceptos pertenecientes al introducido.

Como en la estructuras locales sabemos todos los conceptos, AVIS muestra por pantalla un desplegable para que seleccione uno de ellos en concreto, para las consultas remotas como no se sabe los conceptos de los que se compone este campo tiene que ser escrito por el usuario.

El procedimiento a seguir en esta consulta es buscar el concepto que queremos obtener sus subconceptos y ejecutar la instrucción que lista sus subclases como se ve en las siguientes líneas.

```
if (clase.getLocalName().equals(nombreConcepto)) {
      ExtendedIterator itSubClases = clase.listSubClasses();
      ...
}
```

Véase que en este caso se usa *equals* y no *contains* porque se buscan los subconceptos de un concepto en concreto.

CONSULTA 3.

SuperClase de: Con esta consulta se pretende saber lo contrario de la consulta anterior que es a qué concepto pertenece este otro concepto, es decir, cuál es el concepto padre o super-concepto de este otro.

En este caso se busca un solo concepto padre como en las siguientes líneas:

```
OntClass cPadre = clase.getSuperClass();
```

Si el resultado es nulo quiere decir que nos encontramos ante la raíz de la jerarquía.

CONSULTA 4.

Conceptos con el Atributo: Esta consulta se encarga de buscar aquellos conceptos que contienen un atributo llamado como el patrón introducido. Se realiza una búsqueda de cualquiera de los tipos de atributos que vimos en la consulta 1, pero en este caso el patrón tiene que coincidir exactamente.

En este caso se trabaja con los dominios que pueda tener un determinado atributo de entrada y de esta manera se tienen los conceptos que los contiene. En JENA se usa la instrucción de listado de dominios.

```
ExtendedIterator itDom = tipoDeAtributo.listDomain();
```

Donde tipoDeAtributo puede ser cualquiera de los que vimos en la consulta 1. Al recorrer este iterador obtenemos los conceptos buscados. Si el iterador resulta vacío es porque ese atributo no está presente en ningún concepto de la estructura.

3.6 Ejemplo de aplicación para diseñar una BBDD del Sistema de Información Económica y Contable de Explotaciones Agrarias (SIECAGRI).

Sistema de Información Económica y Contable de Explotaciones Agrarias (SIECAGRI) [67] es un aplicación web de la Asociación Mediterránea de Organizaciones y Productores Agrarios (AMOPA) [68] desarrollada en J2EE y contiene una base de datos diseñada a partir de una estructura de conocimiento (Ontología) aplicando las métricas con las que AVIS crea sus propias estructuras de conocimiento.

La base de datos de SIECAGRI está formada por 30 conceptos. Desde el concepto raíz que es "Parcela" (refiriéndose a la superficie de cultivo de una especie y una variedad en concreto) parten todos los demás como son por ejemplo, "maquinaria", "riegos", "productos fertilizantes", "productos fitosanitarios", "materiales empleados", "trabajo directo", "trabajos de otras empresas", "impuestos", "ingresos", "subvenciones", etc.

Mediante esta metodología fue diseñada la BBDD de SIECAGRI y así es como funciona actualmente esta aplicación, resultando tener la información almacenada de una forma muy eficiente y muy estructurada. La utilización de ontologías de calidad como las que son resultado de AVIS resultan de una gran ayuda para el diseño de BBDD.

La explotación agraria, considerada como la unidad técnico-económica básica de producción aparece, de forma natural como la unidad elemental de dimensión espacial en tanto que constituida tanto por la componente territorial, bajo unos límites definidos, como por la componente técnica, económica, humana, social, etc., de forma asimismo determinada. Por el contrario la empresa aparece como una unidad heterogénea que, independientemente de su definición societaria legal, puede estar constituida por varias explotaciones, incumpliendo el principio de unidad territorial marcada por limites ciertos y determinados, lo que en este caso, la invalida como unidad simple o elemental de análisis.

Por otro lado, la explotación, en relación con su carácter de soporte básico de un proceso productivo determinado, presenta o contiene elementos de heterogeneidad, ya que puede comprender diversas unidades en cuanto a características cualitativas y estructurales del suelo, cultivos y sistemas de cultivo diferentes, etc., aunque mantenga el carácter de unidad técnico-económica de producción, especialmente en cuanto a la dotación de medios de producción, de capital y de gestión y dirección unitaria.

Para resolver este problema, en el marco de análisis propuesto, se opta por una solución híbrida pero sintética: mantener la consideración de la explotación como unidad técnico –económica básica pero desagregándola en tantas unidades espaciales homogéneas como la compongan. En este caso, por tanto, se considera a la **parcela (Concepto Raíz de la Ontología SIECAGRI)** como unidad básica y homogénea de producción y a la **explotación**, como una unidad de rango superior, una "unidad de unidades" integradas en relación con la estructura técnico-económica y organizacional,

que las agrupa. En este sentido se establecen los siguientes elementos metodológicos complementarios:

- Al considerar la parcela como unidad productiva diferenciada, con carácter similar al de los departamentos de una empresa, se prescinde del concepto de explotaciones auxiliares y, si se produce flujo de materiales o servicios entre diferentes parcelas, se imputan respectivamente a costes e ingresos los resultados de dicho flujo.
- Se establece una diferencia neta entre la unidad de producción y el titular de la misma –empresario, agricultor familiar, etc.- detentador de competencias y capacidades, y de un fondo de trabajo individual o familiar. El análisis se centra exclusivamente en la parcela, y por agregación, en la explotación, prescindiendo de las características peculiares del detentador o titular de la misma.

Conocimiento para la creación de la ontología de SIECAGRI.

I	II	III
1. COSTES DIRECTOS Semillas y Plantas. Fertilizantes. Fitosanitarios. Otros suministros. 2. COSTES DE MAQUINARIA Trabajos Contratados. Carburantes y Lubricantes. Reparaciones y Repuestos. 3. COSTES DE MANO DE OBRA Mano de obra especializada. Mano de obra general. 4. COSTES INDIRECTOS PAGADOS Cargas Sociales. Seguros. Intereses y gastos financieros. Arrendamientos. Contribuciones e impuestos. Conservación de edificios y mejoras. Otros gastos generales. 5. AMORTIZACIONES Instalaciones. Maquinaria y Aperos. Sistemas de riego. 6. OTROS COSTES INDIRECTOS Renta de la tierra. Intereses de capitales propios. Mano de obra familiar. 7. COSTES TOTALES DE LA PRODUCCIÓN	1. COSTO DE INSUMOS PRIMARIOS 1.1 Semillas y Plantas. 1.2 Abonos y Fertilizantes. 1.3 Productos Fitosanitarios. 1.4 Carburantes y Lubricantes. 1.5 Alquiler de maquinaria. 1.6 Agua para riego. 1.7 Otros Insumos. 2. COSTO DE INSUMOS HUMANOS 2.1 Mano de obra. 2.2 Dirección. 3. COSTO CAPITAL DE EXPLOTACION 3.1 Amortización. 3.2 Conservación. 3.3 Riegos. 4. COSTO CAPITAL TERRITORIAL 4.1 Renta de la tierra. 4.2 Interés del capital territorial. 5. INTERES DEL CAPITAL DE EJERCICIO 5.1 Interés del Capital de Explotación. 5.2 Interés del Capital Circulante. 6. SEGUROS DE COSECHAS 7. CONTRIBUCIONES E IMPUESTOS 8. COSTO TOTAL.	1. PAGO DE COMISIONES Y COSTE DE CONTROL DE LA RECOLECCIÓN. 2. COSTES VARIABLES DE LOS FACTORES DE PRODUCCIÓN 2.1 Materias Primas. 2.1.1 Agua de Riego. 2.1.2 Fertilizantes. 2.1.3 Insecticidas, fungicidas, herbicidas, abono foliar, etc. 2.1.4 Otras materias primas. 2.1.5 Semillas y Plantas. 2.2 Alquiler de maquinaria 2.3 Costes variables de la maquinaria propia. 2.4 Mano de obra. 3. INTERES DEL CAPITAL CIRCULANTE 4. COSTES FIJOS 4.1 Costes fijos de la maquinaria propia. 4.2 Amortización de la plantación. 4.3 Interés de la plantación. 4.4 Amortización del capital de instalaciones. 4.5 Interés del capital de instalaciones. 4.6 Costes de reposición de árboles y mantenimiento de instalaciones y obra fija. 4.7 Renta de la Tierra. 4.8 Impuestos y Seguros. 5. TOTAL COSTES

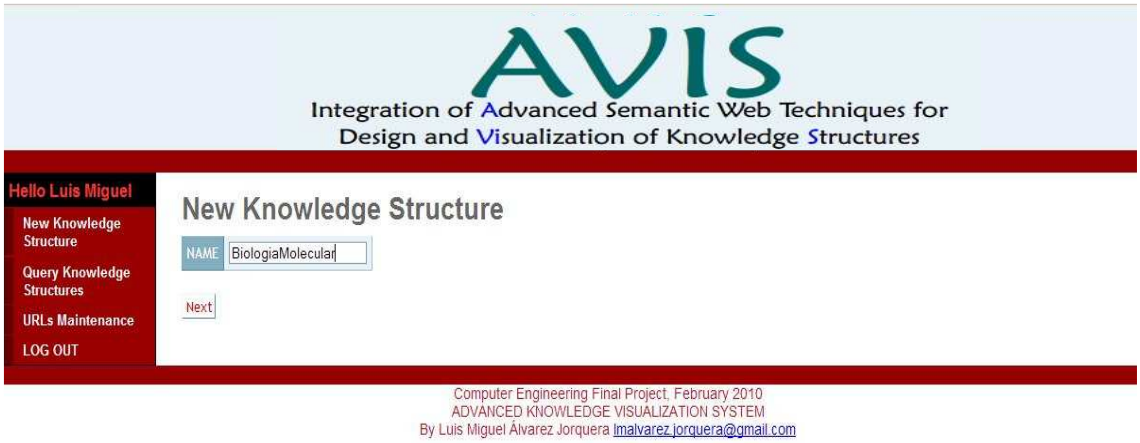
SIECAGRI – Pantalla Inicial

3.7 Ejemplos de funcionamiento de AVIS.

En este apartado se muestra un ejemplo de funcionamiento de AVIS con cualquiera de sus opciones tanto de creación de una nueva estructura como de consultas de estructuras locales y remotas mostrando con volcados de pantalla como va funcionando la aplicación con los datos introducidos.

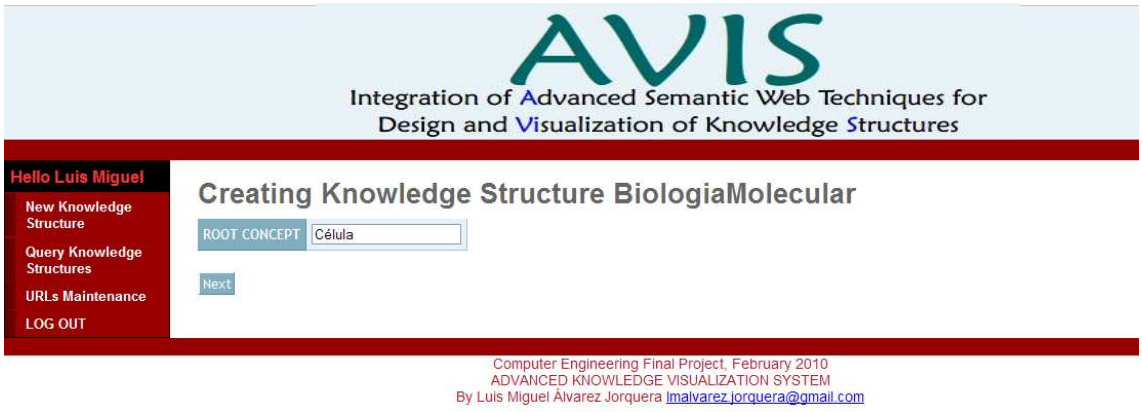
3.7.1 Creación de la estructura de biología molecular del apartado 3.3

Lo primero que hacemos una vez entrados en AVIS con nuestro usuario y contraseña proporcionados por el administrador es pinchar en la opción de Nueva Estructura de conocimiento y escribimos el nombre de la estructura a crear.



Nombre de la nueva estructura

En este caso le hemos llamado **Biología Molecular** y una vez comprobado que en la base de datos no hay ninguna estructura con el mismo nombre, la creamos y pasamos a escribir el concepto raíz.



Nombre del concepto raíz

Para el concepto **Célula**, los atributos introducidos tendrán los siguientes valores:

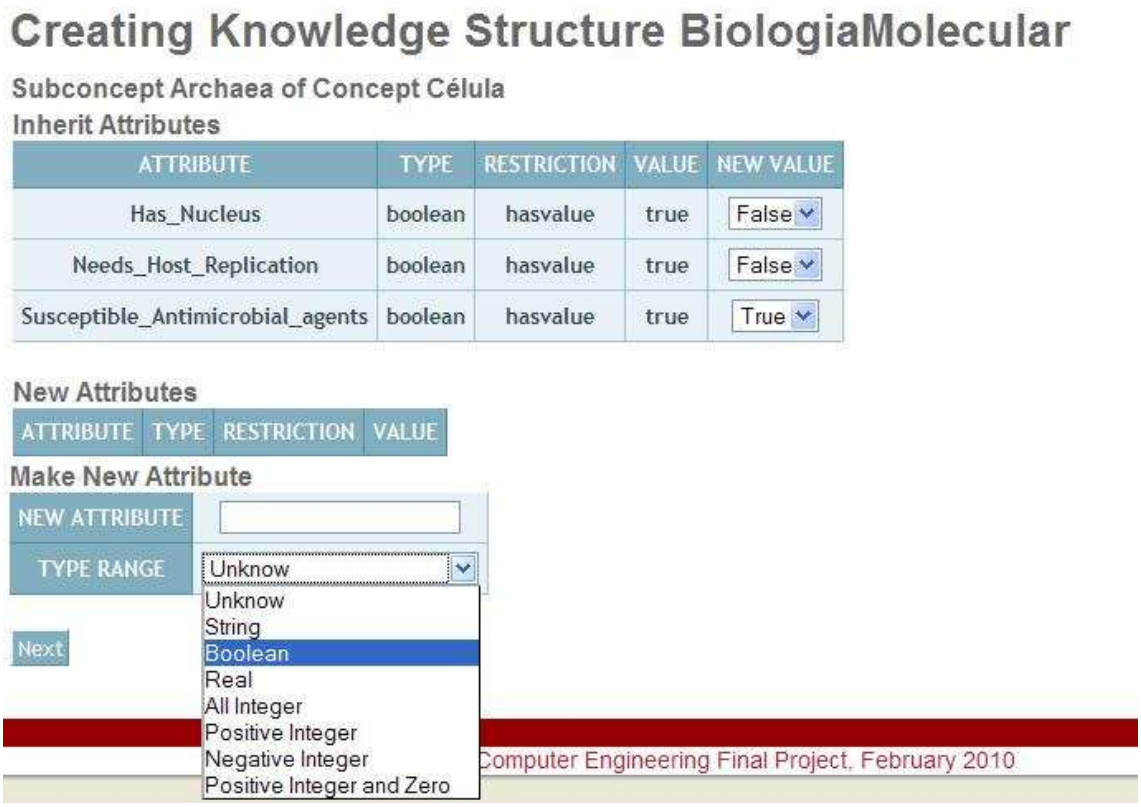
- **Has_Nucleus:** Este atributo puede tomar los valores *Boolean.*
- **Needs_Host_Replication:** Este atributo puede tomar los valores *Boolean.*
- **Susceptible_to_Antimicrobial_Agents:** Tiene un rango de valores *Boolean.*

Para el concepto **Eukaryota,** los atributos introducidos tendrán el valor:

- **Has_Nucleus:** Este atributo toma el valor *True.*
- **Needs_Host_Replication:** Toma el valor *False.*
- **Susceptible_to_Antimicrobial_Agents:** Toma el valor *False.*

Para el concepto **Archaea,** los atributos que se añaden en la estructura son:

- **Has_Nucleus:** Este atributo toma el valor *False*
- **Needs_Host_Replication:** Toma el valor *False*
- **Susceptible_to_Antimicrobial_Agents:** Toma el valor *True*



Creación del concepto Archaea como subconcepto de Célula

El concepto **Bacteria** tiene los valores para los atributos introducidos:

- **Has_Nucleus:** Este atributo toma el valor *False*
- **Needs_Host_Replication:** Toma el valor *False*
- **Susceptible_to_Antimicrobial_Agents:** Toma el valor *False*

El concepto **Virus,** los conceptos introducidos son:

- **Has_Nucleus:** Este atributo toma el valor *False*
- **Needs_Host_Replication:** Toma el valor *True*
- **Susceptible_to_Antimicrobial_Agents:** Toma el valor *False*

Para el concepto de Archaea se introdujo un nuevo atributo que heredaran sus hijos y tomara un valor. Por otro lado sus dos hijos también heredaran los atributos que inicialmente tiene el concepto padre:

- **Optimun_PH:** Puede tomar cualquier valor Real.

Para el concepto hijo **Euryarchaeota** tiene los valores para el atributo introducido:
Optimun_PH: Tiene un valor fijo de 0.7

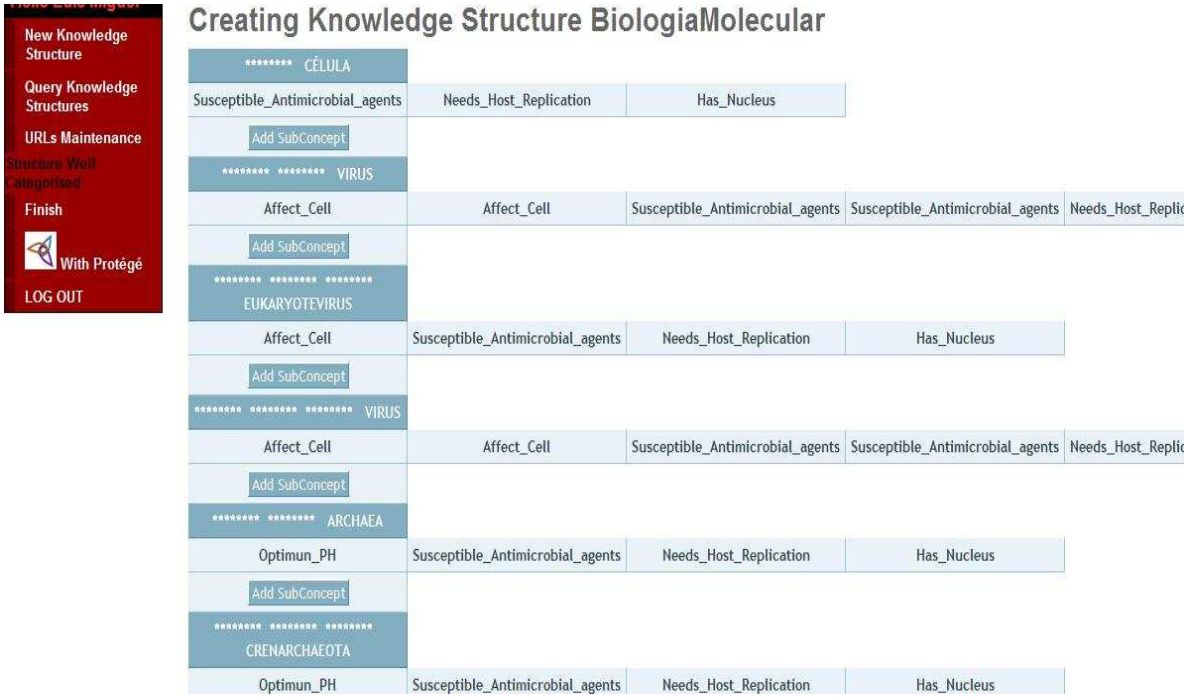
Para el concepto hijo **Crenarchaeota** tiene los valores para el atributo introducido:
Optimun_PH: Tiene un rango de valores entre [1,4].

El siguiente paso es introducir la rama del concepto **Eukaryota,** como esta rama tiene los siguientes conceptos: **Mycetozoa, Parabasalidea, Alveolata, Euglenozoa, Eumetazoa, Bilateria, Fungi, Viridiplantae, Streptophyta y Chlorophyta.** Y los atributos de estos son:

- **Has_Plasmodio**: *Boolean*.
- **Has_Flagelo**: *Boolean*.
- **Has_Alveolos_Corticales:** *Boolean*.
- **Has_Cytostome:** *Boolean*.
- **Has_Tissue**: *Boolean*.
- **Has_Cell_Wall**: *Boolean*.
- **Has_Cloroplast:** *Boolean*.

La rama del concepto bacteria contiene los siguientes conceptos: **Cianobacteria, Flavobacteria, Actinobacteria, Firmicutes**, **Mollicutes, Proteobacteria y Spirochaetes.** Y los atributos son :

- **Type_Gram:** *Boolean*.
- **Respiration** *Boolean*.
- **Length:** Real.



Creando estructura de conocimiento en AVIS

Una vez finalizada a estructura podemos ver en la imagen anterior que en el menú de la izquierda aparecen unas opciones llamadas “Finish” y “With Protégé” que sirven para finalizar la ontología y almacenarla en la base de datos o para abrir la ontología con el Protégé con la API de AVIS. Si la estructura no está correctamente creada y bien categorizada estas 2 opciones no estarán disponibles.

Estructura Bien Categorizada

Si pulsamos en la opción “With Protégé” se abrirá en Protégé la estructura que acabamos de crear en AVIS, concretamente esta estructura de biología molecular que estamos tratando en este ejemplo, una vez introducida en AVIS podemos abrir Protégé y ver lo siguiente:

Vista en Protégé de la estructura de conocimiento creada en AVIS

3.7.2 Consultas de la estructura “The gene ontology”.

Para realizar consultas a una estructura remota, primero tenemos que dar de alta o registrar las direcciones URL de estas estructuras remotas, para ello, seleccionamos la opción del menú de AVIS correspondiente al mantenimiento de URLs.

Mantenimiento de URLs

En esta pantalla podemos registrar todas las URLs que necesitemos consultar, también las podemos modificar y eliminarlas. Una vez concluido esta alta de URLs debemos pasar a la opción de consultar remotas donde el sistema nos dará la opción de realizar 4 tipo de consultas distintas sobre las estructuras remotas que deseemos. Para seleccionar las estructuras que deseamos consultar hay seleccionar los checkboxes correspondientes.

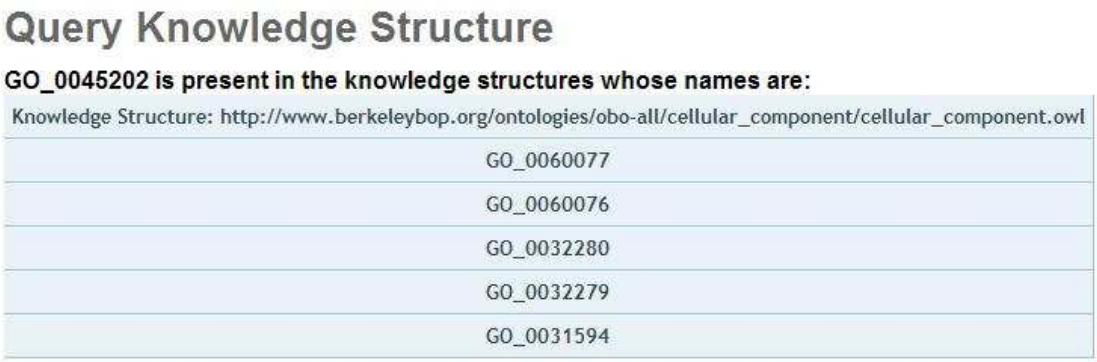
Consulta remota del tipo cualquier ocurrencia

Podemos consultar varias estructuras para un mismo patrón, véase en la figura que la estructura del gene ontology es la url que está más abajo.

A continuación vamos a realizar varias consultas sobre esta estructura que ya explicamos en apartados anteriores.

Consulta tipos / Clases del concepto GO_0045202

Y el resultado es el siguiente.



Query Knowledge Structure

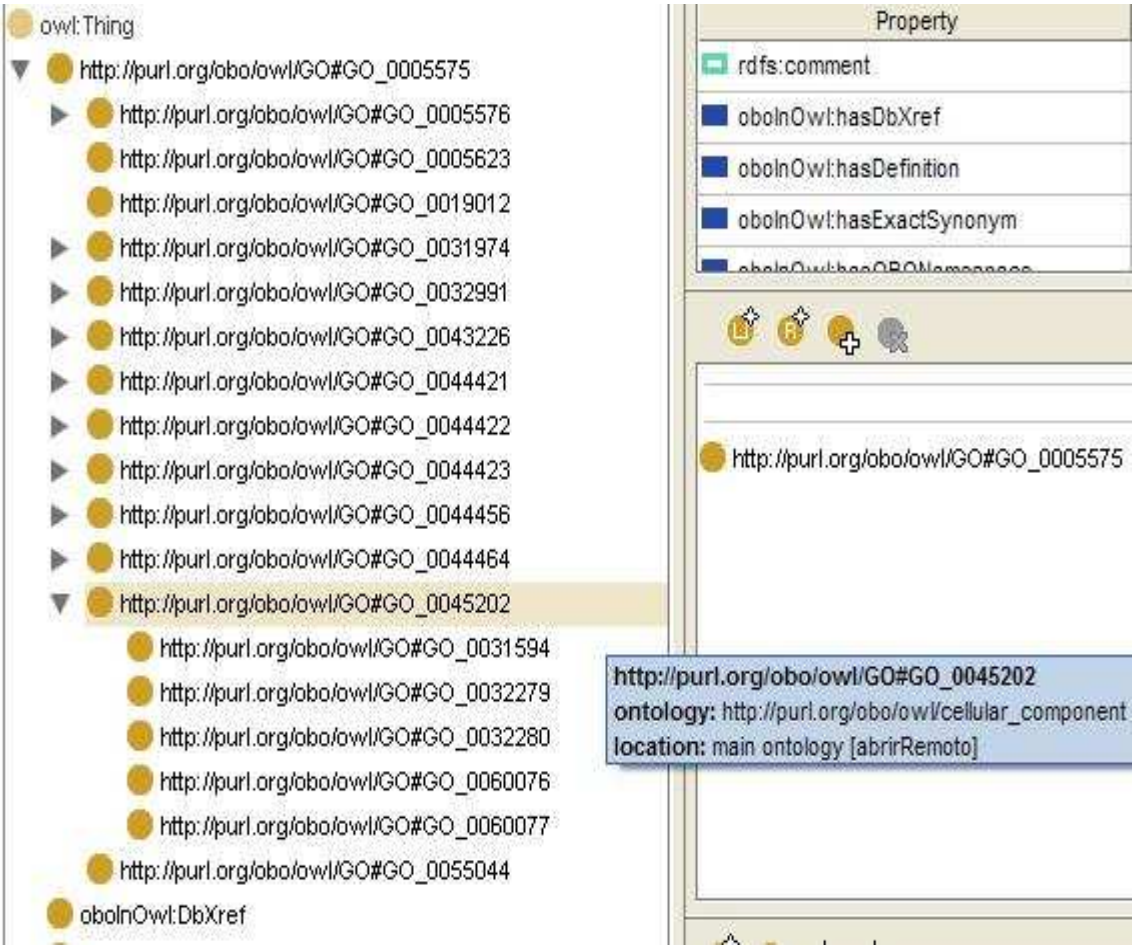
GO_0045202 is present in the knowledge structures whose names are:

Knowledge Structure: http://www.berkeleybop.org/ontologies/obo-all/cellular_component/cellular_component.owl
GO_0060077
GO_0060076
GO_0032280
GO_0032279
GO_0031594

Resultado de una consulta en AVIS

Y podemos ver que el concepto buscado tiene 5 subconceptos. Los expertos en biología y en la genética saben que el consistorium de la gene ontology llaman la GO_0045202 al lugar de la **comunicación interneuronal y la zona sináptica**, y también sabe que los conceptos hijos a éste tratan sobre zonas más concretas de la zona sináptica como es el GO_0060077 que corresponde a la **hendidura sináptica**.

Para comprobar que no nos equivocamos en la consulta realizada podemos abrir esta estructura desde Protégé directamente con la opción que nos da esta aplicación, y una vez abierto podemos consultar visualmente la jerarquía de conceptos de la estructura.



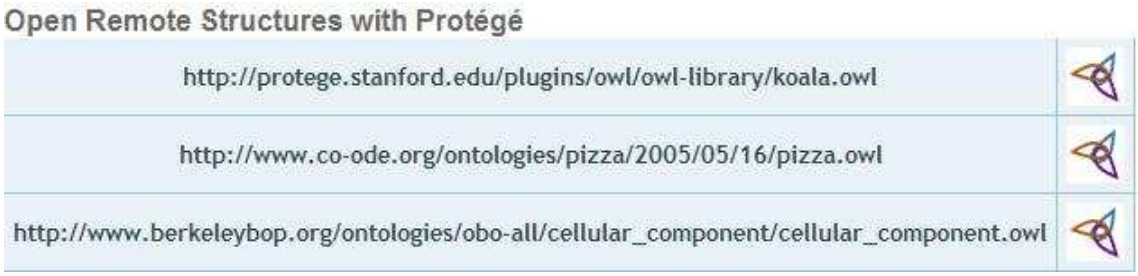
Concepto GO_0045202 y sus hijos en la jerarquía GO desde Protégé

Si realizáramos la consulta enfocada de forma contraria, es decir, quiero saber ahora cuál es el concepto al cual pertenece el concepto de **hendidura sináptica GO_0060077.** Para esta consulta utilizamos la opción de consultar superclases de AVIS y el resultado sería el siguiente:

Resultado de una consulta en AVIS

Y el resultado de la consulta es el esperado, es decir, GO_0045202 (zona sináptica) es padre del concepto GO_0060077 (hendidura sináptica).

AVIS también ofrece la posibilidad de abrir todas las estructuras remotas con Protégé, tan sólo hay que pulsar el botón del icono de Protégé en la estructura correspondiente.



Opción de abrir estructura remota con Protégé

4. Conclusiones y Vías Futuras.

La rápida evolución experimentada por las tecnologías de la información y las Comunicaciones, el uso masivo del ordenador personal como herramienta de procesamiento de información y conexión a redes, así como el crecimiento de Internet, han propiciado que sea necesaria una nueva forma de almacenamiento y acceso a las Bases de Conocimiento. Se pasa del modelo relacional con un alto número de tablas, con muchísimas columnas y relaciones, a utilizar ontologías para almacenar todo este conocimiento, y entonces podemos ver a las ontologías como otro tipo de "Bases de Datos" o simplemente estructuras de conocimiento ordenadas y eficientes para almacenar información sobre un conocimiento en concreto que se puede ir conectando con otras estructuras.

En este proyecto se propuso crear un sistema que fuera capaz de crear y evaluar estas nuevas estructuras de conocimiento, a partir del conocimiento de expertos en el dominio que se quiera trabajar. Así se abre un camino para la implementación de la web semántica y la comunicación hombre máquina.

La iniciativa de la web Semántica busca que los contenidos sean manipulables por las computadoras utilizando esquemas comunes mediante ontologías. Este trabajo se ha creado para que cualquier persona que no fuera experta en la web semántica, pudiera utilizar esta aplicación y ella misma pudiera crear la estructura de conocimiento de manera natural y manual.

El potencial de la web semántica llegará a su plenitud cuando se tengan programas que recojan contenidos web de diversas fuentes, procesen la información e intercambien información con otros programas. La efectividad de esos agentes software se incrementará de forma exponencial cuantos más contenidos semánticos existan en la web y mayor sea el número de servicios web que estén disponibles.

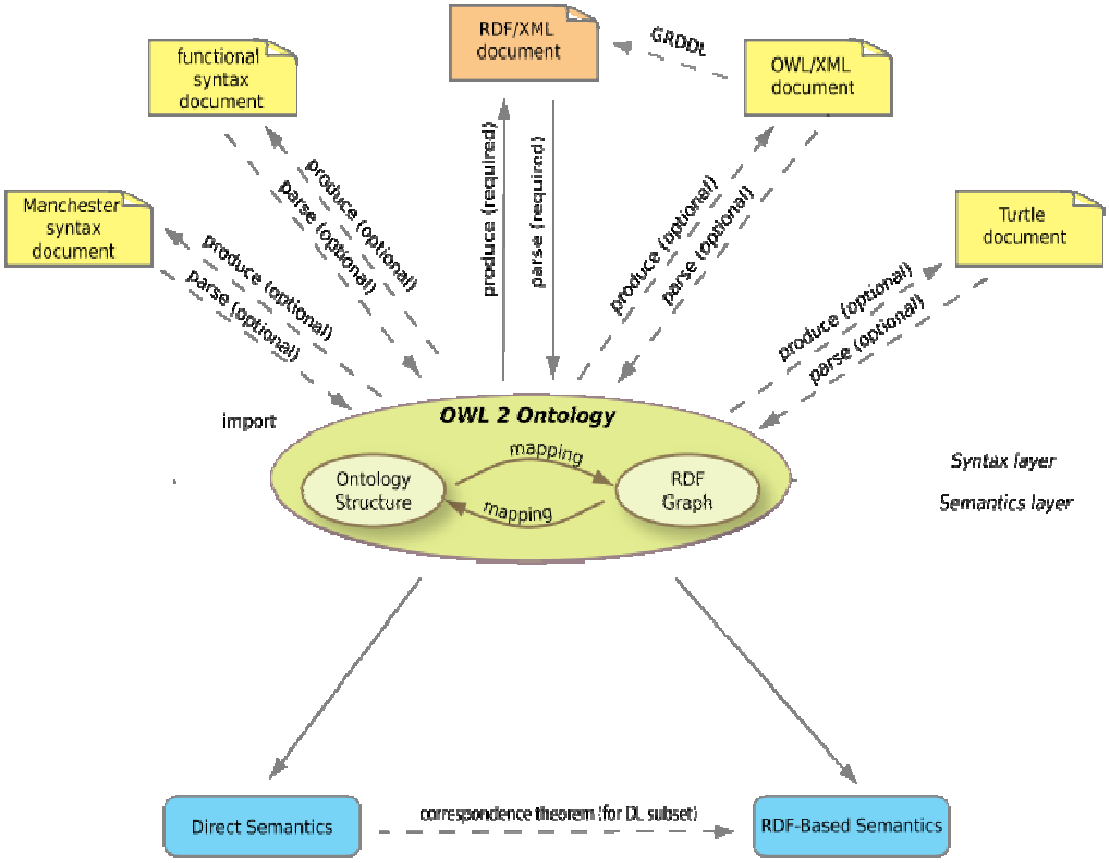
Este proyecto abre las puertas a nuevas ramas de investigación con respecto a la creación de estructuras de conocimiento (ontologías) y sobre todo a avanzar más sobre sus posibles consultas como punto más importante a tener en cuenta ya que al fin y al cabo es lo que más útil va a resultar para investigadores expertos de distintos tipos de conocimientos.

Las vías futuras de este proyecto corresponden a aumentar las posibilidades de la API, es decir, ir mostrando paso a paso en Protégé la creación de conceptos y de atributos, relaciones entre ellos, etc. En el apartado de las consultas en AVIS se realizan 4 tipos diferenciados básicos pero a partir de aquí se pueden pensar otras nuevas o combinaciones de varias, y esto abre una importante línea de investigación.

También se pueden aplicar otras tecnologías como Hibernate para la persistencia entre los objetos de la aplicación y la base de datos, y se le puede aplicar un log para posibles incidencias que puedan surgir con la tecnología "logger".

Tras la aparición de el nuevo estándar de OWL (**OWL2 el 27 de Octubre de 2009)** resulta otra vía futura correspondiente a la adaptación de estas estructuras a este nuevo OWL2 que se puede consultar en la web de w3c.[10]

Veamos una figura resumen del nuevo OWL 2 y que sería un futuro trabajo de este proyecto la adaptación a este nuevo estándar.



OWL 2

La adaptación al lenguaje de consultas sobre ontologías **OWL QL** desarrollado por el grupo de investigación de estructuras de conocimiento de la Universidad de Stanford también queda como vía futura de este proyecto [69].

Resumiendo, quiero señalar que este proyecto abre las puertas a las ramas de investigación dichas anteriormente y parece que apuntan por buen camino, y pienso que esta aplicación puede mejorar el trabajo de muchas personas que trabajan con estructuras de conocimiento.

Este proyecto será publicado para su uso en organizaciones de investigación de la biología molecular y del sector agrícola al igual que ha sido publicado el Sistema de Información Económica y Contable de Explotaciones Agrarias (SIECAGRI).

5. Bibliografía.

[1] T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web," *Scientific American*, May 2001, pp. 34-43.

[2] R. Studer, R. Benjamins and D. Fensel, "Knowledge Engineering: Principles and Methods," *Data and Knowledge Engineering*, vol. 25, no. 1_2, 1998, pp. 161- 197.

[3] Gómez-Pérez, O. Corcho and M. Fernández-Lopez, *Ontological Engineering*, Springer-Verlag 2nd Ed., 2004.

[4] N. Guarino and C. Welty, "Supporting Ontological Analysis of Taxonomic Relationships," *Data and Knowledge Engineering*, vol. 39, no. 1, 2001, pp. 51-74.

[5] N. Guarino and C. Welty, "Evaluating ontological decisions with OntoClean," *Communications of the ACM*, vol. 45, no. 2, 2002, pp. 61-65.

[6] Volker, J., Vrandecic, D., Sure, Y. (2005) Automatic Evaluation of Ontologies (AEON). Institute AIFB, University of Karlsruhe.

[7] Solo Ciencia. http://www.solociencia.com/biologia/bioinformatica-concepto.htm

[8] NCBI. (2001). http://www.istl.org/02-winter/internet.html

[9] Shahar, Y. & Musen, M.A. 1996. Knowledge_Based Temporal Abstraction in Clinical Domains. Artificial Intelligence in Medicine, 8(3):267-298.

[10] W3C. http://www.w3.org/2001/sw/WebOnt; http://www.w3c.org

[11] Schulz,S., Romacker M., Faggioli, G., & Hahn, U. 1999. From knowledge import to knowledge finishing automatic acquisition and semi-automatic refinement of medical knowledge. In Proceedings of KAW'99. http://sern.ucalgary.ca/KSI/KAW/KAW99/papers/Schulz1/ Schulz.pdf

[12] Dieng, R., Corby, O., Giboin, A., & Ribieres, M. (1999): Methods and tools for corporate knowledge management. International Journal of Human_Computer Studies 51:567-598.

[13] Martinez-Bejar, R., and Martin-Rubio, F. (1997) *A Mathematical Fuctionsbased Approach for Analysing Elicited Knowledge*. International Conference on Software Engineering and Knowledge Engineering. Knowledge Systems Institute. USA. 76

[14] J. T. Fernandez-Breis and R. Martinez-Bejar, "A cooperative framework to integrate ontologies" *International Journal of Human-Computer Studies*, vol. 56, no. 6, 2002, pp. 665-720.

[15] Torralba-Rodriguez, F.J., Fernandez-Breis, J.T. Valencia-Garcia, R.,Ruiz-Sanchez, J.M., Martinez-Bejar, R.Gomez-Rubi, J.A. (2003) An Ontological Framework for Representing and Exploiting Medical Knowledge. Expert Systems with Applications 25(2):211-230.

[16] Vladan, Devedzic. Understanding Ontological Engineering. Commun. ACM 45(4): 136-144 (2002)

[17] S. Kremer, "Spatio-Temporal Connextionist Networks: A Taxonom and Review," *Neural Computation*, vol. 13, 2001, pp. 249-306.

[18] Chamiel, G., Pagnucco, M.: Exploiting ontological information for reasoning with preferences. In: Proceedings of the Fourth Multidisciplinary Workshop on Advances in Preference Handling.(2008)

[19] Kiefer, C., Bernstein, A., Stocker, M.: The fundamentals of SPARQL – A virtual triple approach for similarity_based semantic web tasks. In: ISWC '07. (2007).

[20]Mizoguchi, R. A step towards ontological engineering. *Proceedings of The 12th National Conference on AI of JSAI* (June 1998), 24–31.

[21] D.B. Lenat and R.V. Guha, *Building large knowledge-based systems*, Addison-Wesley Publising Company Inc., 1990.

[22] M. Uschold and M. King (1995): Towards a methodology for building ontologies. Proceedings IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Canada.

[23] Mark S. Fox, Michael Gruninger, Yin Zhan. Enterprise engineering: An information systems perspective.

[24] A. Bernaras, I. Laresgoiti and J. Corera (1996): Building and Reusing Ontologies for Electrical Network Applications. Proceedings European Conference on Artificial Intelligence (ECAI'96).

[25] M. Fernandez, A. Gomez_Perez and N. Juristo (1997): METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Symposium on Ontological Engineering of AAAI, Stanford, California, 1997, pp. 33-40.

[26] Swartout, R. Patil, K. Knight, T. Russ (1997): Toward Distributed Use of Large-Scale Ontologies, Symposium on Ontological Engineering of AAAI, 77 Stanford, California,1997.

[27] Staab, S., Schnurr, H.P., Studer, R., Sure, Y.: Knowledge processes and ontologies.IEEEIntelligent Systems 16 (2001)

[28] A. Bernaras, I. Laresgoiti and J. Corera (1996): Building and Reusing Ontologies for Electrical Network Applications. Proceedings European Conference on Artificial Intelligence (ECAI'96).

[29] Holsapple, C.W., Joshi, K.D.: A collaborative approach to ontology design. Commun. ACM 45 (2002) 42–47

[30] J. Euzenat (1995): Building consensual knowledge bases: context and architecture. In "Building and sharing large knowledge bases", IOP Press, pp. 143- 155.

[31] J. Euzenat (1996): Corporative memory through cooperative creation of knowledge bases and hyperdocuments. Proceedings 10th KAW, Banff (Canada).

[32] S. Decker, M. Erdmann, D. Fensel and Rudi Studer (1999): Ontobroker: Ontology Based Access to Distributed and Semi_Structured Information. In "Semantic Issues in Multimedia Systems", Proceedings of DS_8, Kluwer Academic Publisher, 1999, pp. 351-369.

[33] Andreas Hotho, Robert J, Christoph Schmitz, Gerd Stumme. BibSonomy: A Social Bookmark and publication Sharing System.

[34] Diaz, A., Baldo, G., Canals, G. "Co_Protege: Collaborative Ontology Building with Divergences" 17th International Conference on Database and Expert Systems Applications (DEXA'06) pp. 156-160 Kozaki, K., Sunagawa, E., Kitamura, Y, Mizoguchi, R. "Distributed Construction ofOntologies Using Hozo". The Institute of Scientific and Industrial Research (ISIR), Osaka University 8_1 Mihogaoka, Ibaraki, Osaka, 567_0047 Japan.

[35] Kozaki, K., Sunagawa, E., Kitamura, Y., Mizoguchi, R. "Distributed Construction of Ontologies Using Hozo". The Institute of Scientific and Industrial Research (ISIR), Osaka University 8-1 Mihogaoka, Ibaraki, Osaka, 567_0047 Japan.

[36] Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F. "The DBin Semantic Web platform: an overview". Universita Politecnica delle Marche (Italy).

[37] Hepp, M.; Bachlechner, D.; Siorpaes, K.: OntoWiki: Communitydriven Ontology Engineering and Ontology Usage based on Wikis. Proceedings of WikiSym 2005 78

[38] Braun, S., Schmidt, A., Zacharias, V. "SOBOLEO: vom kollaborativen Tagging zur leichtgewichtigen Ontologie". In: Tom Gross (eds.): *Mensch & Computer - 7. Fachubergreifende Konferenz - M&C 2007*, Oldenbourg Verlag, 2007, pp. 209-218

[39] S. Staab, H_P. Schnurr, R. Studer and Y. Sure, "Knowledge Processes and Ontologies," *IEEE Intelligent Systems*, vol. 16, no. 1, 2001.

[40] C. Tempich, S. Pinto and S. Staab, "Ontology Engineering Revisited: An Iterative Case Study," *Lecture Notes in Computer Science*, vol. 4011, 2006, pp.110- 124.

[41] A. Gomez-Perez and M. D. Rojas, "Ontological reengineering and reuse," In: D. Fensel and R. Studer, Editors, *11th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW'99)Lecture Notes in Artificial Intelligence* vol. 1621, Springer, Berlin, 1999, pp. 139–156.

[42] B. Ganter and R. Wille, "*Formal Concept Analysis: Mathematical Foundations*", Springer-Verlag, 1999.

[43] Punera K., Rajan S., Ghosh J.(2006): Autonomic Construction of N_ary Tree Based Taxonomies. University of Texas at Austin. Sixth IEEE International Conference on Data Mining – Workshops (ICDMW'06)

[44] Fridman Noy, Natalya and Musen Mark, A. "SMART: Automated Support for Ontology Merging and Alignment", KAW'99Twelfth Workshop on Knowledge Acquisition, Modeling and Management, Alberta, Canada, 4-7:1-20.

[45] Maedche, Alexander. University of Karlsruhe, Germany, "Ontology Learning for the semantic Web", Kluwer Academic Publishers. 2002.

[46]Gomez-Perez, Asuncion, "Evaluation of Ontologies". International Journal of Intelligent Systems 16(3):391-409. 2001.

[47] Barry Boehm TRW Professor of Software Engineering, Computer Science Department Director, USC Center for Software Engineering. Accessdate 2008-10- 18

[48] Introducción al lenguaje Java.
http://www.sc.ehu.es/sbweb/fisica/cursoJava/Intro.htm

[49] JENA. *A Semantic Web Framework for Java*. http://jena.sourceforge.net/

[50] JENA. *An Introduction to RDF and the Jena RDF API.*
http://jena.sourceforge.net/tutorial/RDF_API/index.html.

[51] JENA. *Jena 2 Ontology API*. http://jena.sourceforge.net/ontology/index.html.79

[52] PostgreSQL**:** The world's most advanced open source database**.**
http://www.postgresql.org/

[53] Manual PostgreSQL 8.3.
http://www.postgresql.org/docs/8.3/interactive/index.html

[54] Manual PostgreSQL Espanol.
https://forja.rediris.es/docman/view.php/312/454/Postgres-User.pdf

[55] Tomcat. Manual Struts. http://struts.apache.org/1.x/userGuide/installationtc.html

[56] Wikipedia. http://www.wikipedia.org

[57] BERNERS-LEE, T; CONNOLLY, D; SWICK, R. *Web Architecture: Describing and Exchanging Data*. http://www.w3.org/1999/04/WebData, 1999.

[58] GAMMA, HELM, JOHNSON, VLISSIDES. *Patrones de diseno: elementos de software orientado a objetos reutilizables*. Pearson Addison-Wesley, noviembre de 2002.

[59] LARMAN. UML y Patrones. *Introduccion al analisis y diseno orientado a objetos.* Prentice Hall, 1999.

[60] SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. *OWL Web Ontology Language Guide*. http://www.w3.org/TR/owl-guide/, 10 de febrero de 2004.

[61] BECHHOFER, S.; PATEL-SCHNEIDER, P. F.; TURI, D. *OWL Web Ontology Language Concrete Abstract Syntax*. http://owl.man.ac.uk/2003/concrete/latest/, 10 de diciembre de 2003.

[62] The Open Biomedical Ontologies. http://www.obofoundry.org/

[63] The Gen Ontology. http://www.geneontology.org/GO.downloads.shtml

[64] *Fco. Javier Bermúdez Ruíz* - Desarrollo de Aplicaciones Distribuidas. Servlets, JSP, RMI, CORBA, J2EE. http://dis.um.es/~jbermudez/dad

[65] Pellet. http://clarkparsia.com/pellet/

[66] AJAX. http://es.wikipedia.org/wiki/AJAX

[67] Segura Artero P., Martínez Béjar R., Álvarez Jorquera Luis M., Constantini B. *Sistema de Información Económica y Contable de Explotaciones Agrarias*. SIECAGRI 2009.

[68] Asociación Mediterránea de Organizaciones de Productores Agrarios AMOPA www.amopa.es

[69] http://www-ksl.stanford.edu/projects/owl-ql/

ANEXOS

A. Instalación de AVIS.

Los requisitos hardware para poder instalar la aplicación vendrá dados por la máquina virtual Java(JVM) o JRE que se tenga instalada o se vaya a instalar (es necesaria para el
funcionamiento correcto del sistema) y que pueden ser consultados en la siguiente dirección: http://java.sun.com/ . Para poder instalar la aplicación se necesita, como se ha comentado anteriormente, tener instalada la máquina virtual de java. Para el desarrollo de este proyecto se ha utilizado JRE 6 (Java Runtime Environment), que asegura compatibilidad con versiones anteriores.

En este documento se explicará la instalación de la JRE 6, aunque es similar en otras versiones. Se comienza descargando desde la página:

http://java.sun.com/javase/downloads/index.jsp el instalador que se desee según el sistema operativo. Si el sistema operativo es Windows sólo es necesario hacer doble clic sobre el ejecutable descargado. Si el sistema operativo es Linux se debe ejecutar desde la consola el ejecutable .bin que se ha descargado.

El segundo paso es la instalación de PostgreSQL desde la página:
http://www.postgresql.org /. Una vez descomprimido el fichero más actualizado de la web anterior tan solo hay que ejecutar su Instalación. Para poder instalar correctamente se puede utiliza la página:
http://www.monografias.com/trabajos28/instalacion-postgresql/instalacion-postgresql.shtml en la cual explica paso a paso como instalar PostgreSQL.

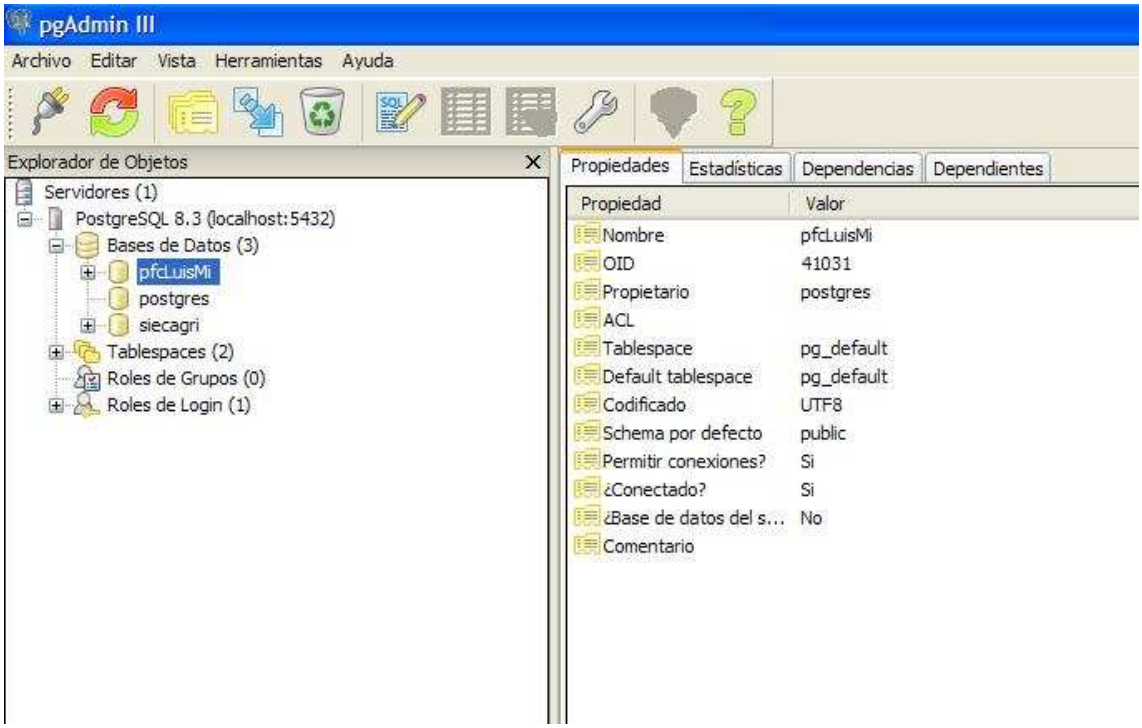
Una vez instalada la base de datos PostgreSQL la arrancamos de la siguiente manera, teniendo en cuenta que está instalada en la carpeta C:\pgsql.

```
C:\pgsql\bin\initdb.exe

C:\pgsql\bin\pg_ctl start
```

Y de esta manera ya tenemos la base de datos postgre lista en nuestro servidor. Para comenzar a utilizarla tenemos que lanzar la aplicación PgAdminIII donde crearemos las base de datos de AVIS.

Veamos en la siguiente figura como es PgAdminIII.



PgAdminIII pantalla pricipal

AVIS necesita para su funcionamiento 2 tablas que debemos crear nosotros que son "usuarios" y "urls". Las demás tablas para almacenar las estructuras de conocimiento ya se encarga JENA de crearlas.

Los scripts SQL de creación de estas dos tablas son los siguientes:

```
CREATE TABLE usuarios
(
  nombre character varying(128) NOT NULL,
  apellidos character varying(128) NOT NULL,
  nif character varying(9) NOT NULL,
  direccion character varying(128) NOT NULL,
  cp character varying(5) NOT NULL,
  email character varying(128) NOT NULL,
  telefono character varying(9) NOT NULL,
  clave character varying(9) NOT NULL,
  CONSTRAINT clave_primaria PRIMARY KEY (nif)
)

CREATE TABLE urls
(
  id_url character varying(39) NOT NULL,
  url character varying(128) NOT NULL,
  CONSTRAINT id_url_pkey PRIMARY KEY (id_url)
)
```

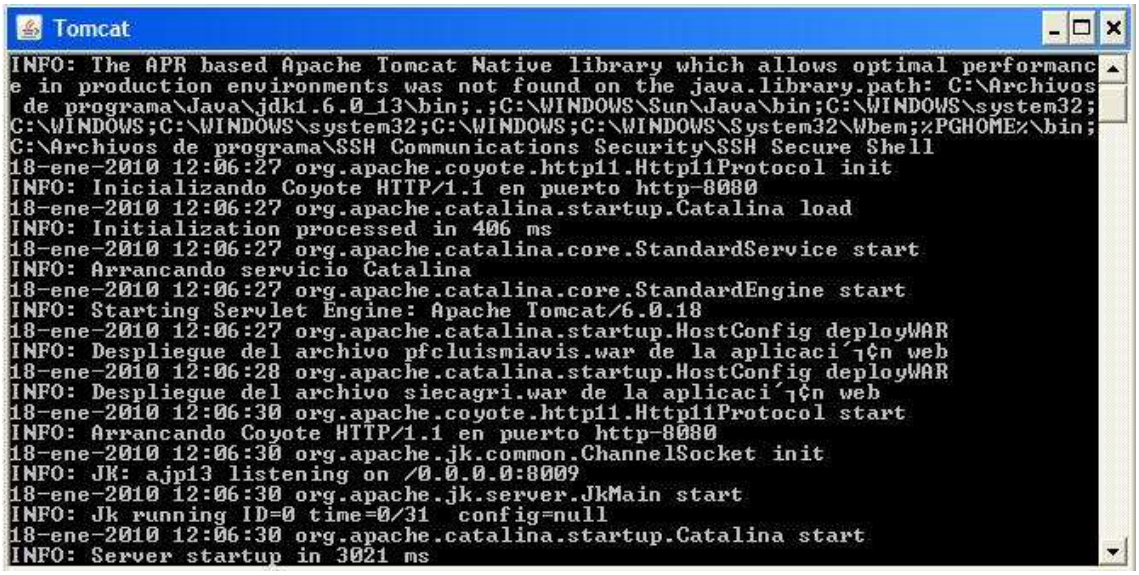
Scripts SQL de creación de tablas en AVIS

Una vez lista la base de datos ya solamente queda instalar el servidor y contenedor de servlets TomCat. Para ello debemos ir a la web de tomcat que es http://tomcat.apache.org/ y nos descargamos la versión 6.0.20. La instalación es muy sencilla, tan sólo hay que descomprimir el fichero y alojarlo en la carpeta que queramos en nuestro disco, por ejemplo en C:\apache-tomcat6.0.20. Para que tomcat arranque correctamente debemos indicarle con 2 variables de entorno JAVA_HOME y TOMCAT_HOME dónde está la máquina virtual de java instalada en nuestra máquina y cuál es la ruta definitiva de tomcat.

Como AVIS es una aplicación J2EE, va empaquetada en un fichero .WAR. Ese fichero empaquetado lo colocamos en la carpeta webapps de nuestro tomcat.

Concluidos todos estos pasos y con la base de datos funcionando, simplemente nos vamos a la carpeta C:\apache-tomcat6.0.20\bin y ejecutamos el script startup.bat para arracarlo, y cuando lo queramos parar ejecutamos el script shutdown.bat.

Esto es lo que vemos en pantalla con el tomcat arrancado.

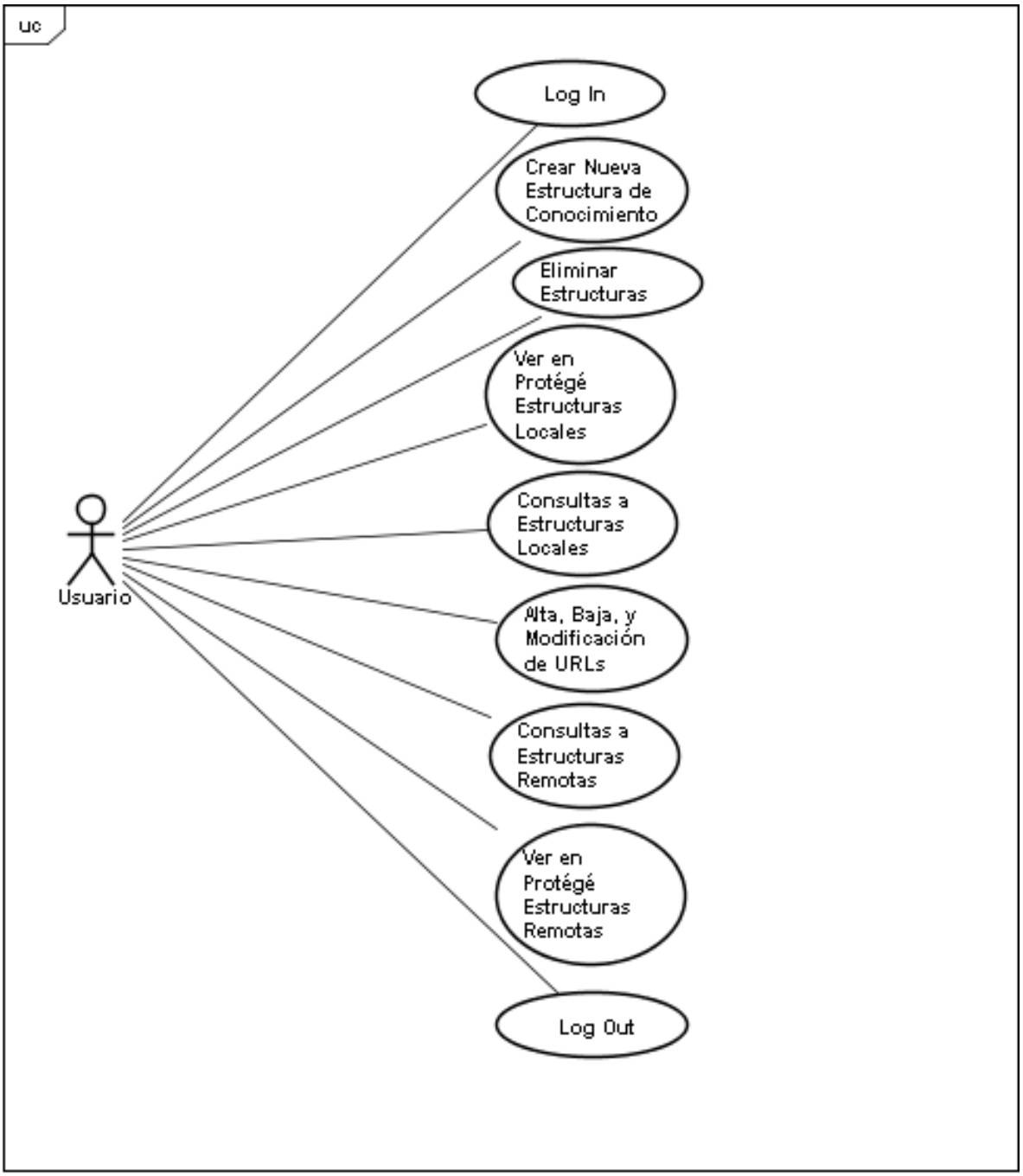


Consola Tomcat

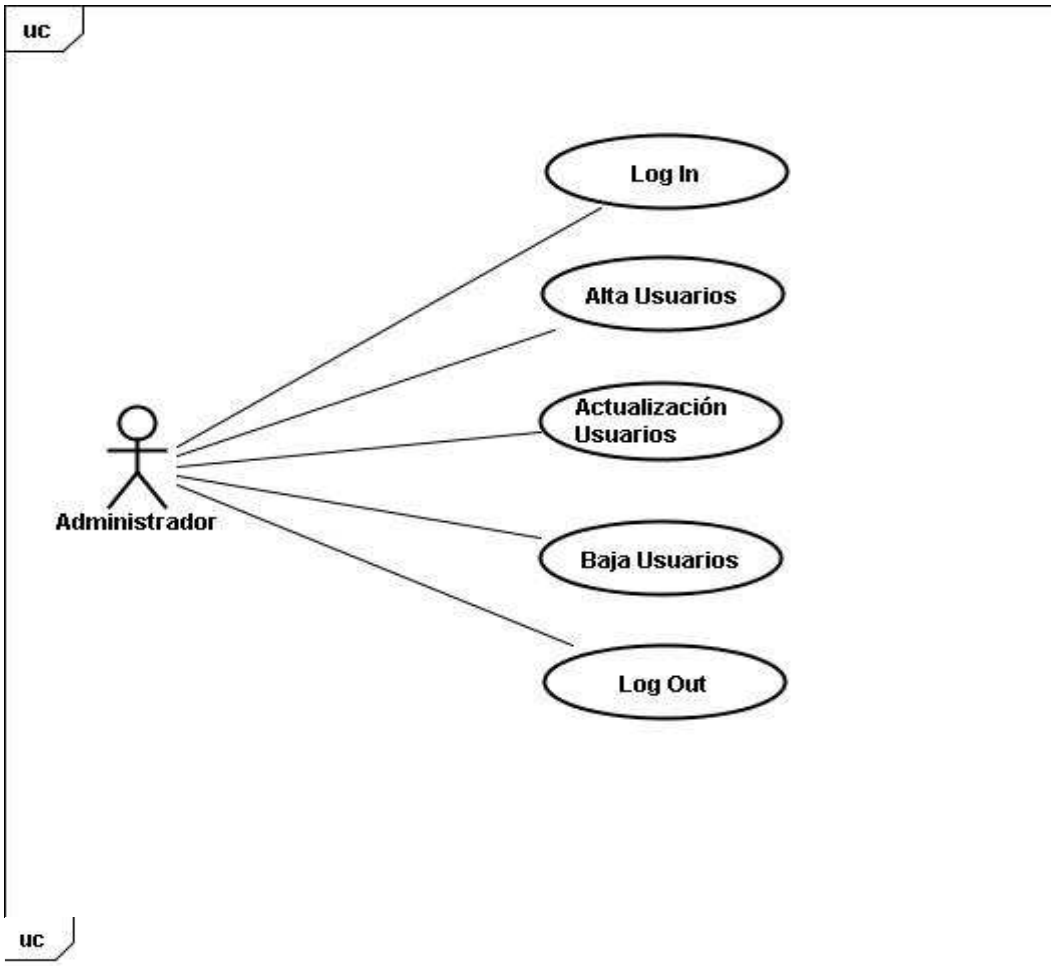
En esta pantalla se puede ver como se van desempaquetando las aplicaciones WAR que tengamos en la carpeta WEBAPPS.

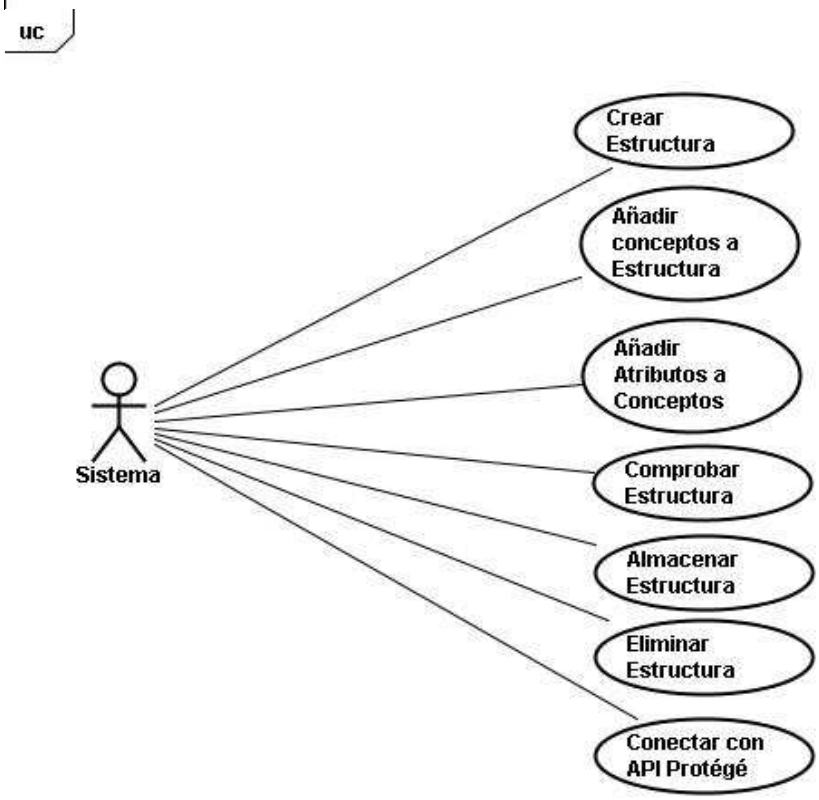
B. Casos de Uso de AVIS, diagramas UML.

Caso de uso del usuario normal.

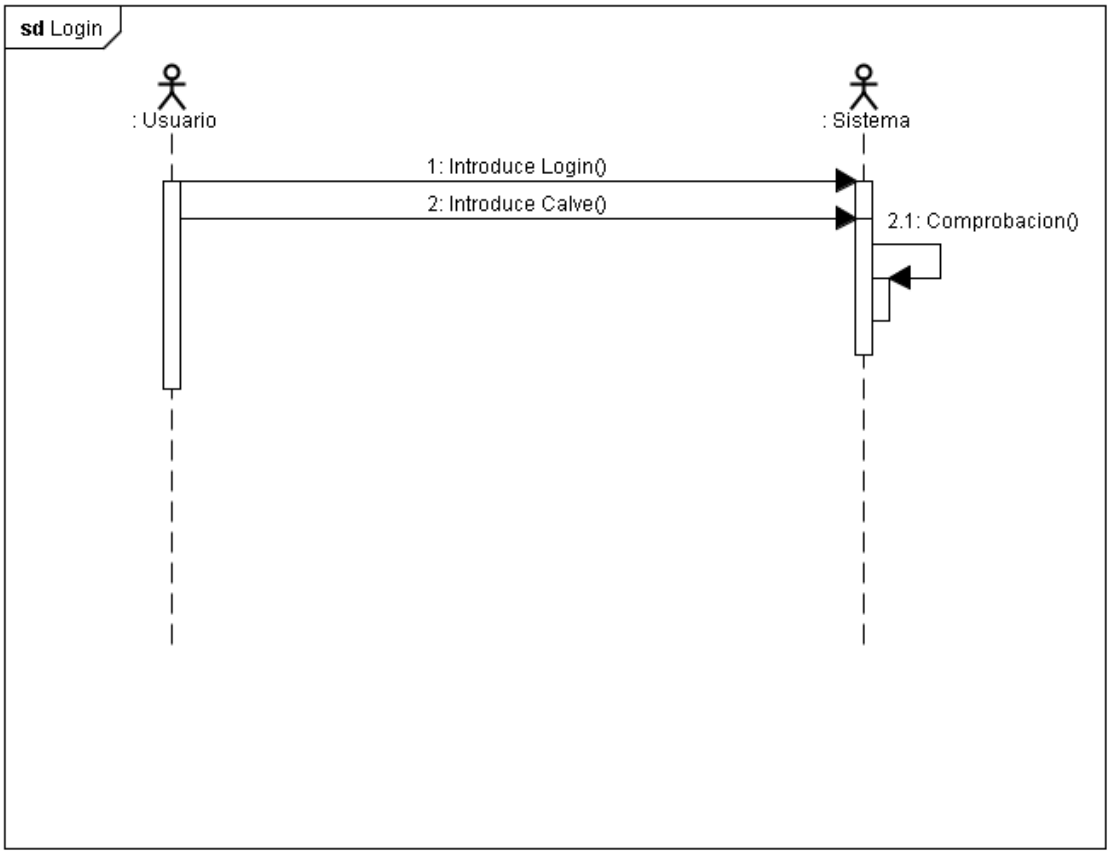


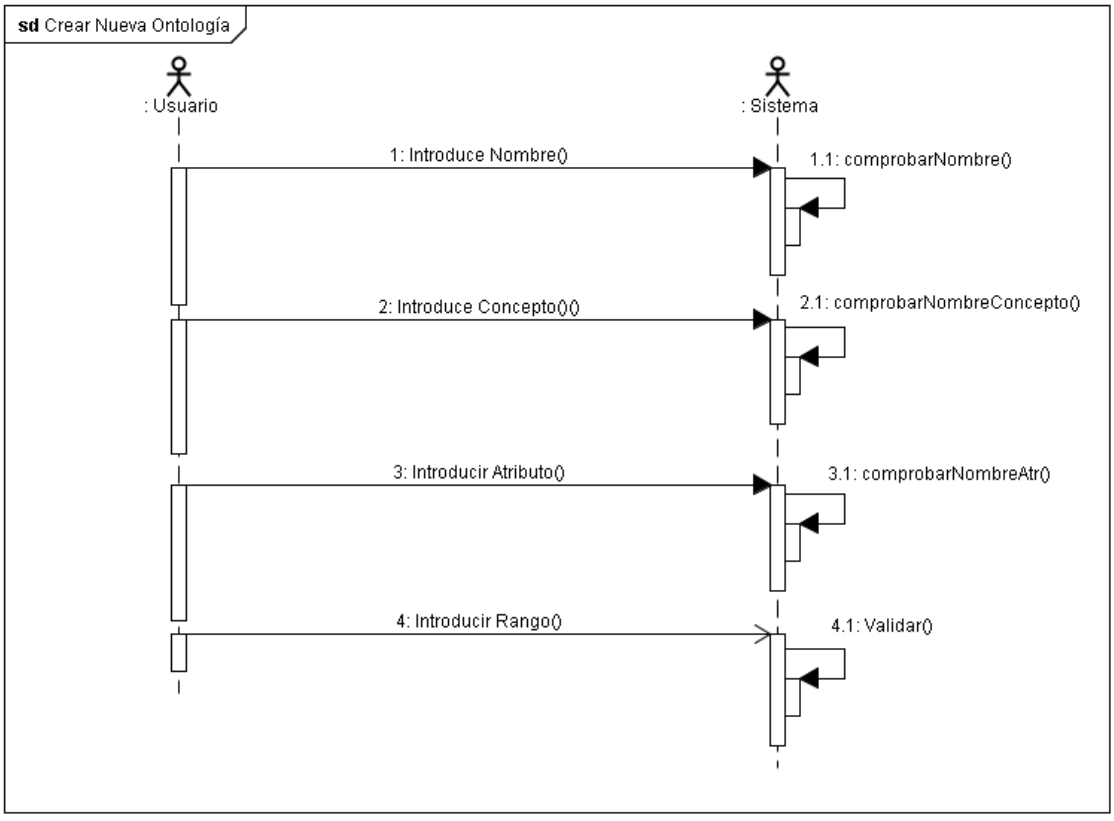
Casos de uso del usuario administrador y del sistema.

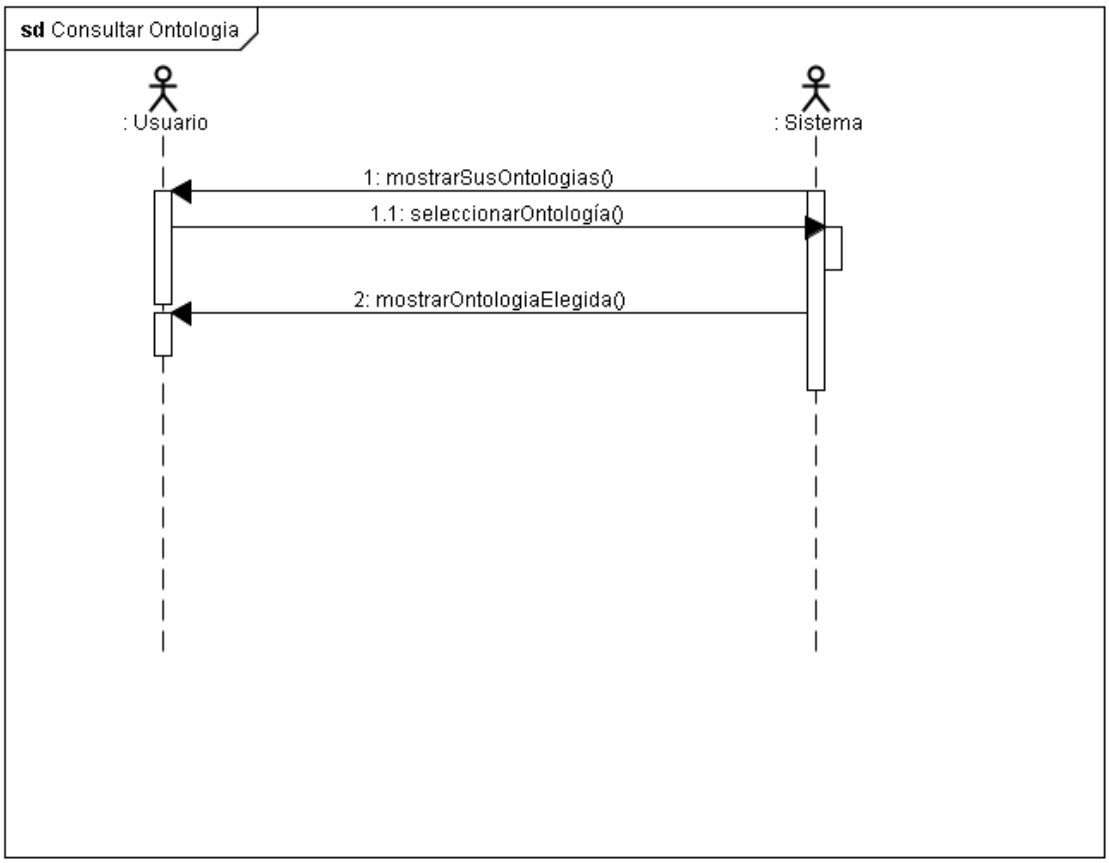


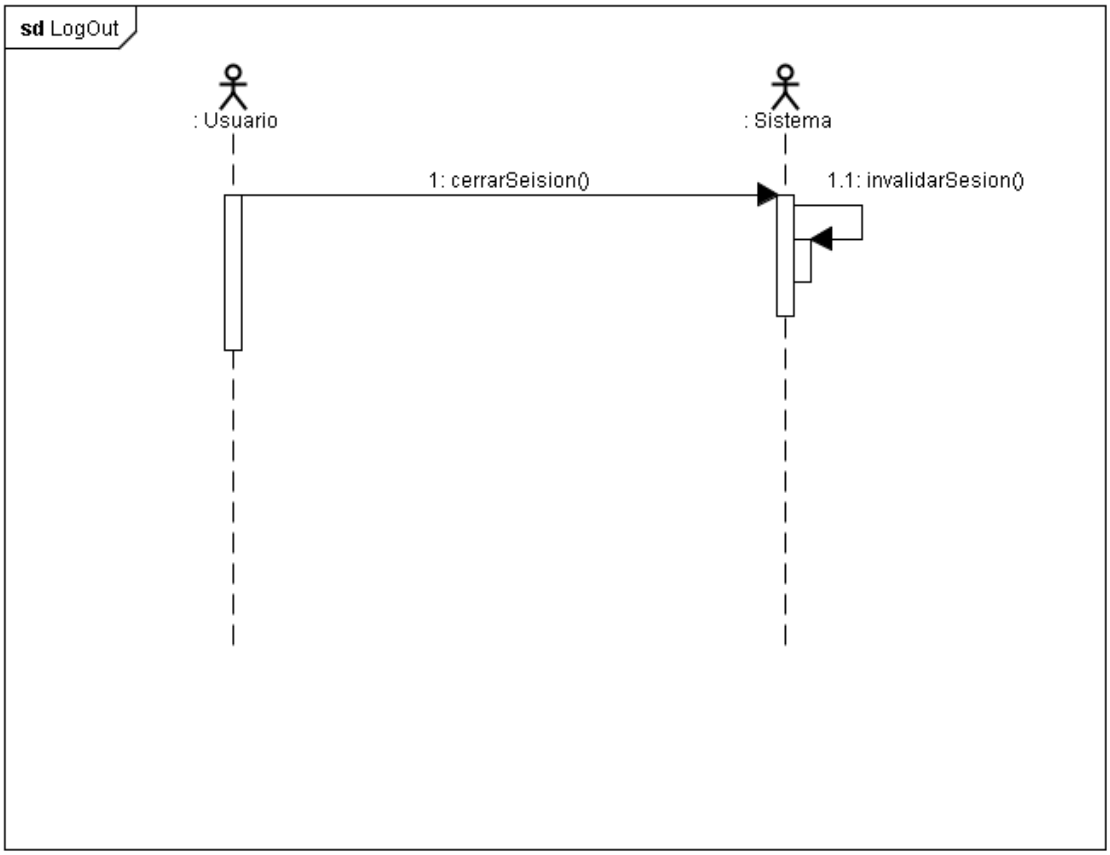


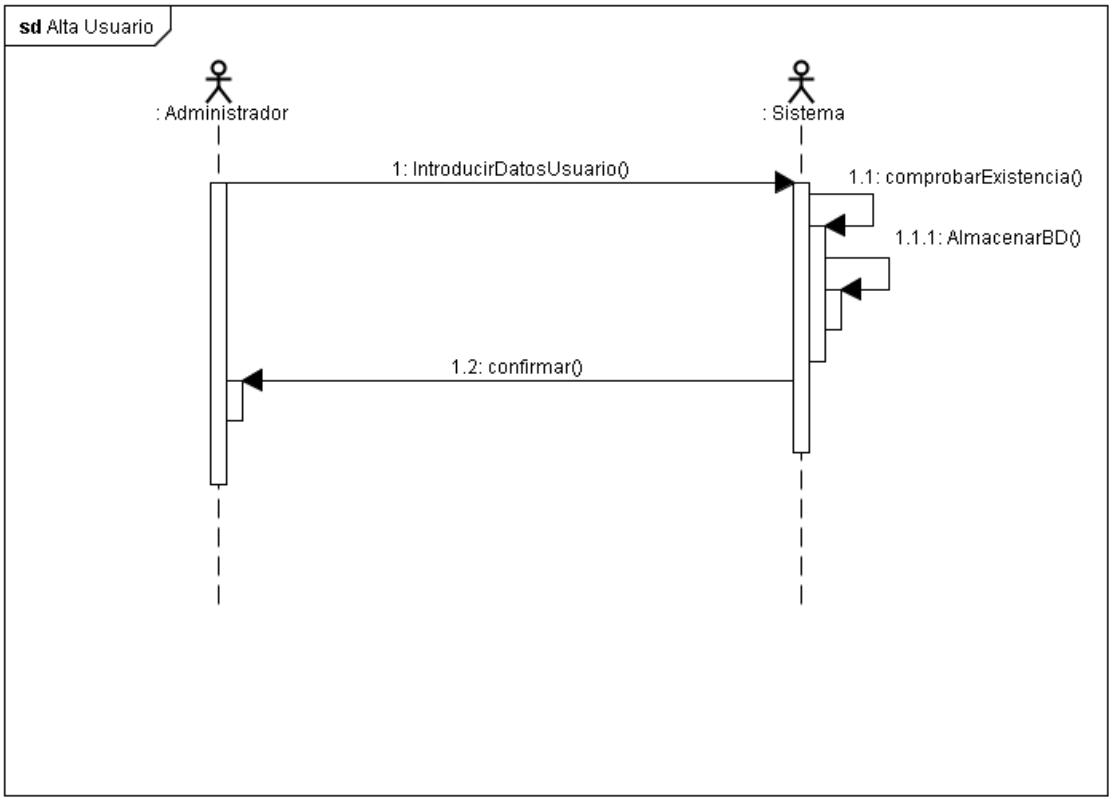
Diagramas de Secuencia UML de los CU mas comunes.

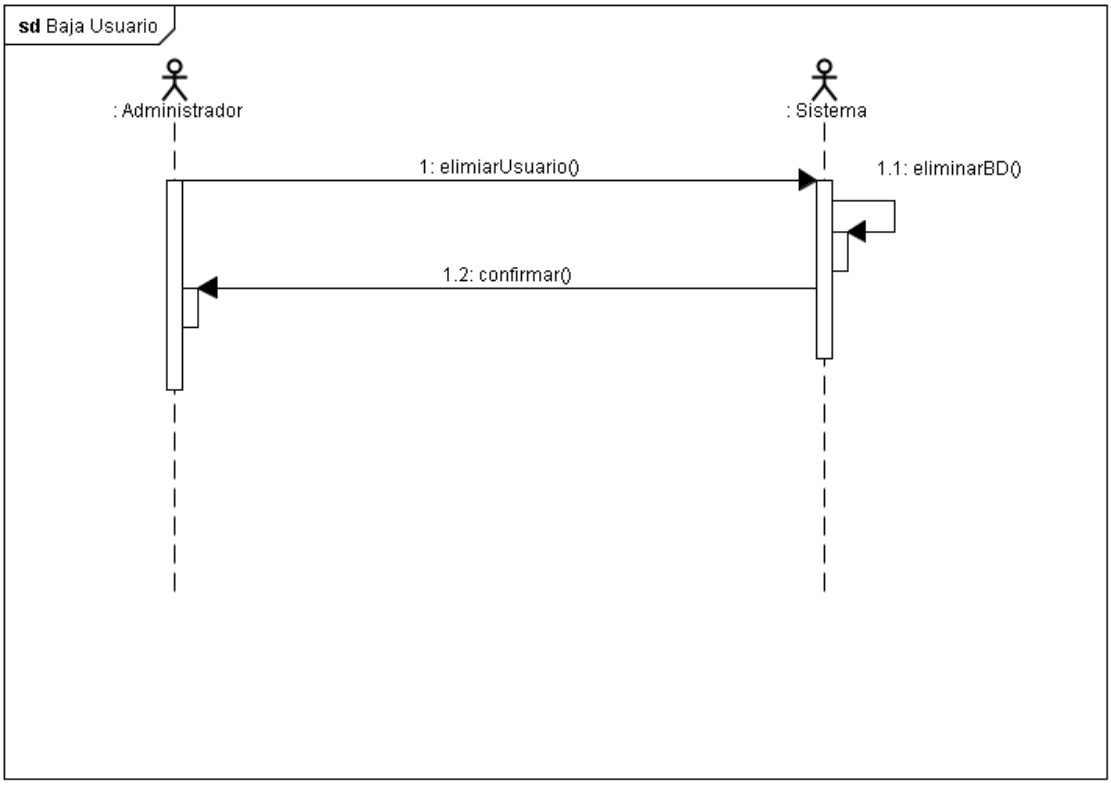


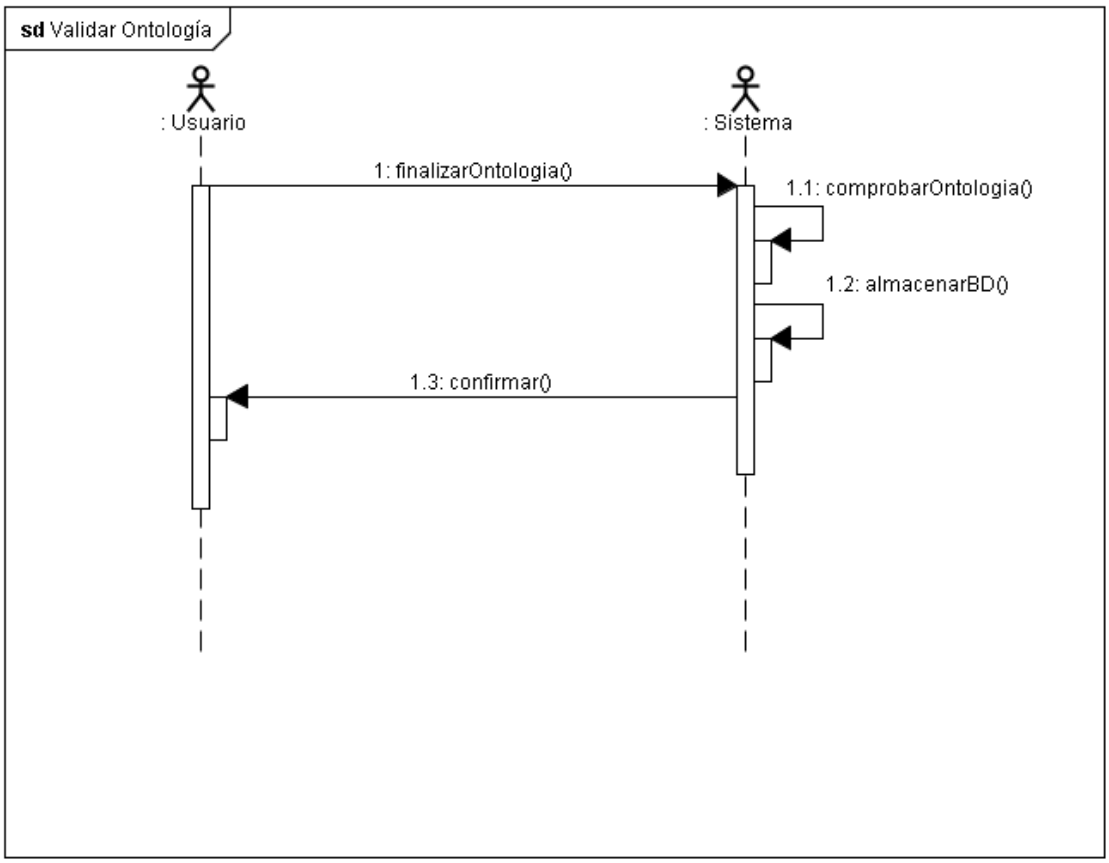


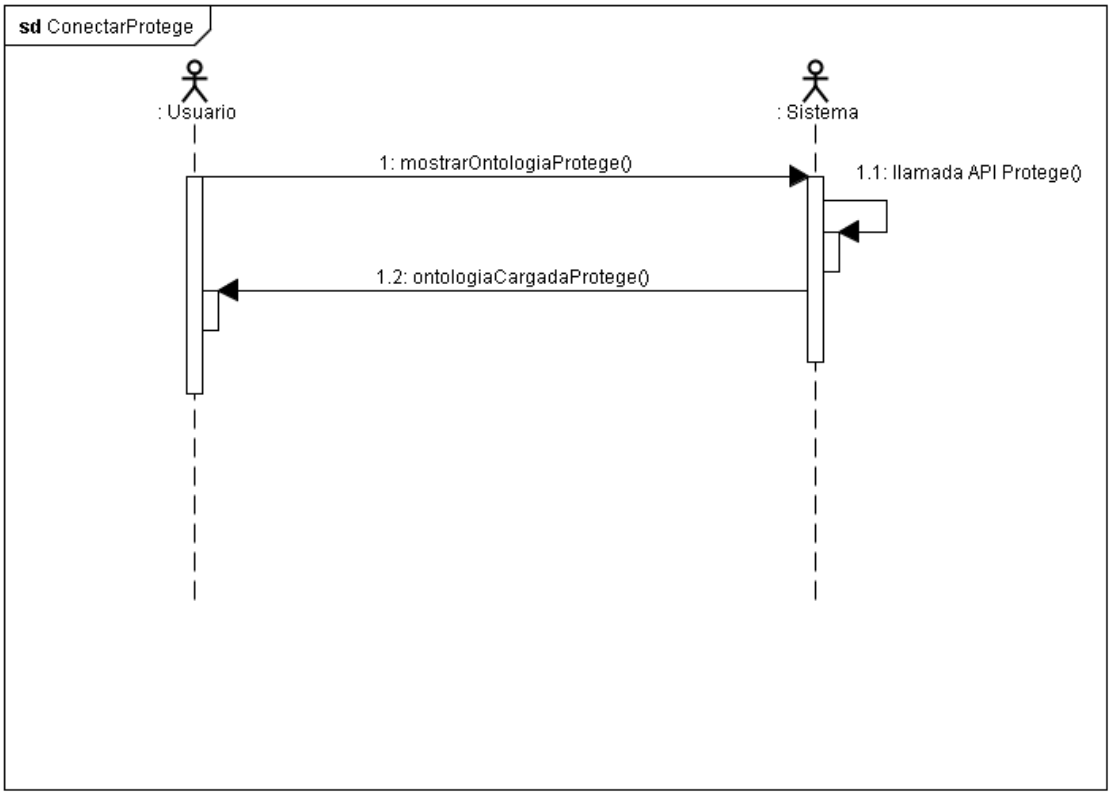

sd Consultar Ontologia
: Usuario
: Sistema
1: mostrarSusOntologias()
1.1: seleccionarOntología()
2: mostrarOntologiaElegida()


sd LogOut
: Usuario
: Sistema
1: cerrarSeision()
1.1: invalidarSesion()


sd Alta Usuario
: Administrador
: Sistema
1: IntroducirDatosUsuario()
1.1: comprobarExistencia()
1.1.1: AlmacenarBD()
1.2: confirmar()


sd Baja Usuario
: Administrador
: Sistema
1: elimiarUsuario()
1.1: eliminarBD()
1.2: confirmar()

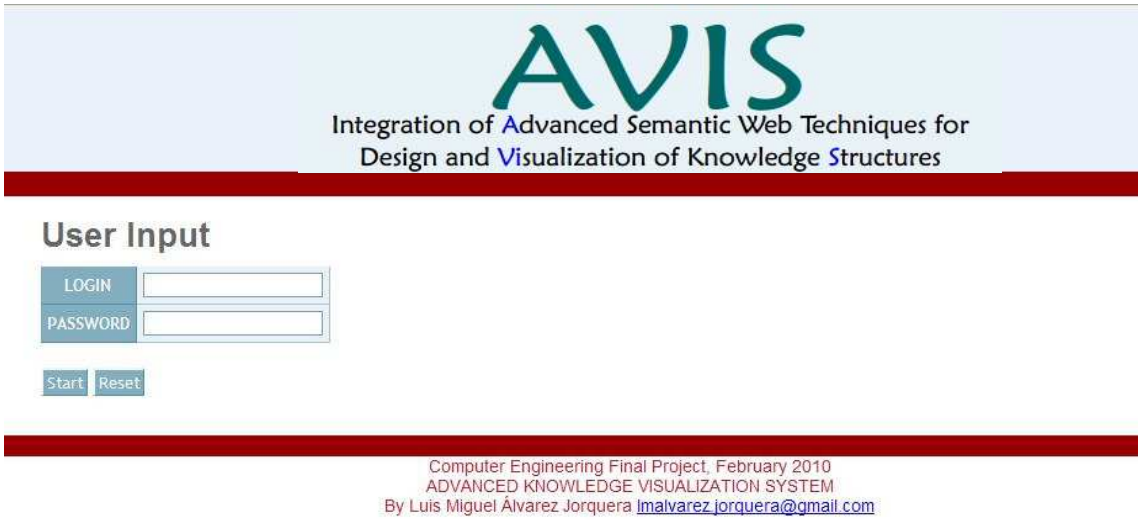

sd Validar Ontología
: Usuario
: Sistema
1: finalizarOntologia()
1.1: comprobarOntologia()
1.2: almacenarBD()
1.3: confirmar()


sd ConectarProtege
: Usuario
: Sistema
1: mostrarOntologiaProtege()
1.1: llamada API Protege()
1.2: ontologiaCargadaProtege()

C. Manual de Usuario.

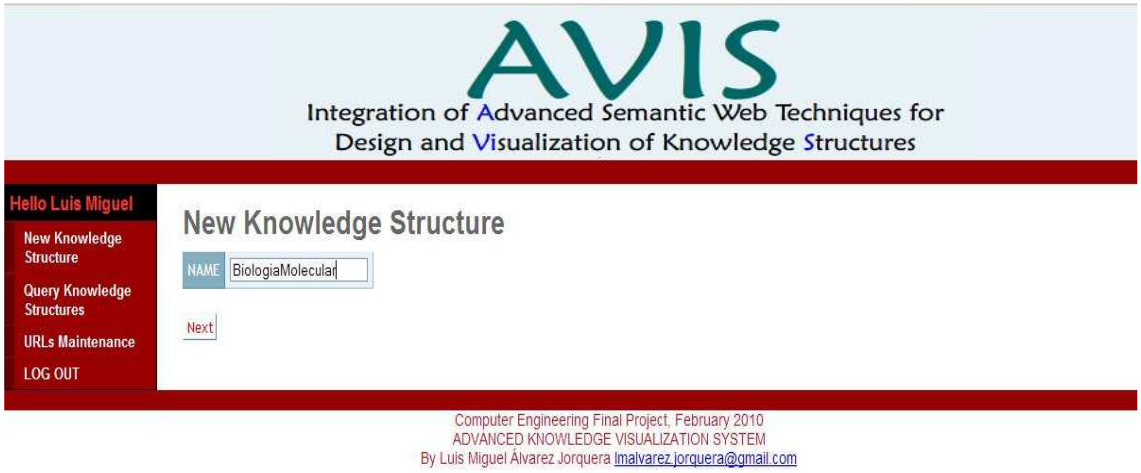
Log In

La pantalla principal es la pantalla de login o la de entrada de usuarios, debemos escribir nuestro usuario (que será nuestro DNI con que no han dado de alta) y una contraseña que tengamos asignada por el administrador.



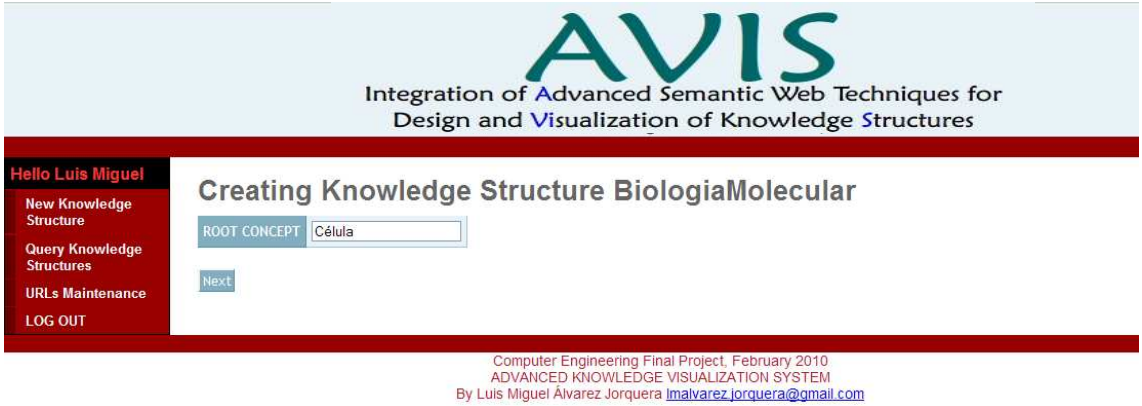
Nueva estructura de conocimiento

Seguidamente tenemos en lado izquierdo un menú con todas las opciones. La primera es la de creación de una nueva estructura de conocimiento.

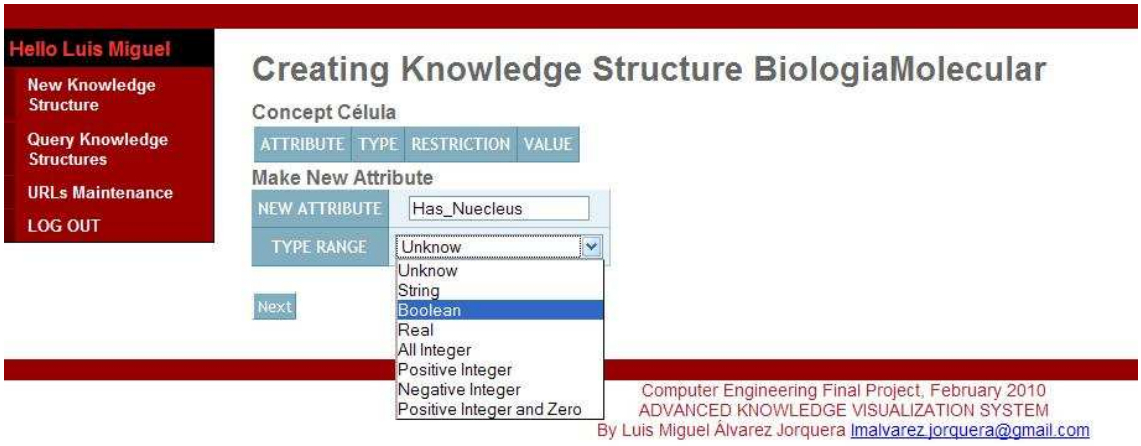


Y a partir de aquí empezamos a crear conceptos con sus determinados atributos.

Comenzando por el concepto raíz.

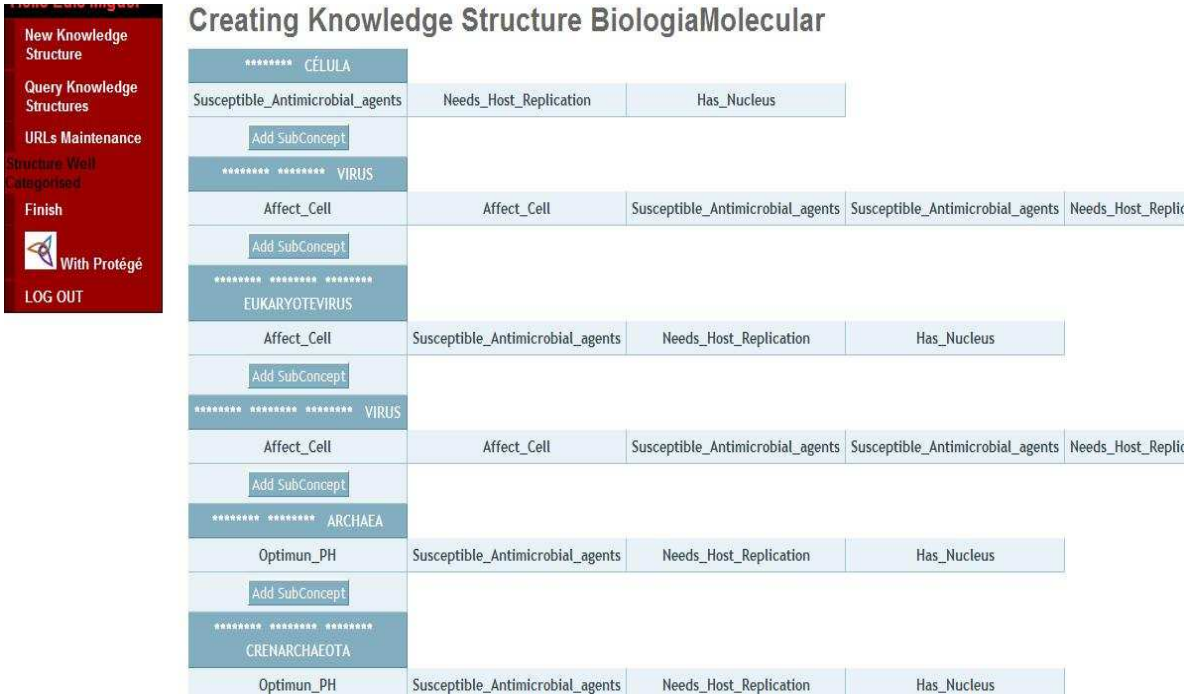


Y continuado con los atributos.



Par continuar añadiendo atributos o para finalizar el concepto que estemos creando tenemos estos 2 botones:

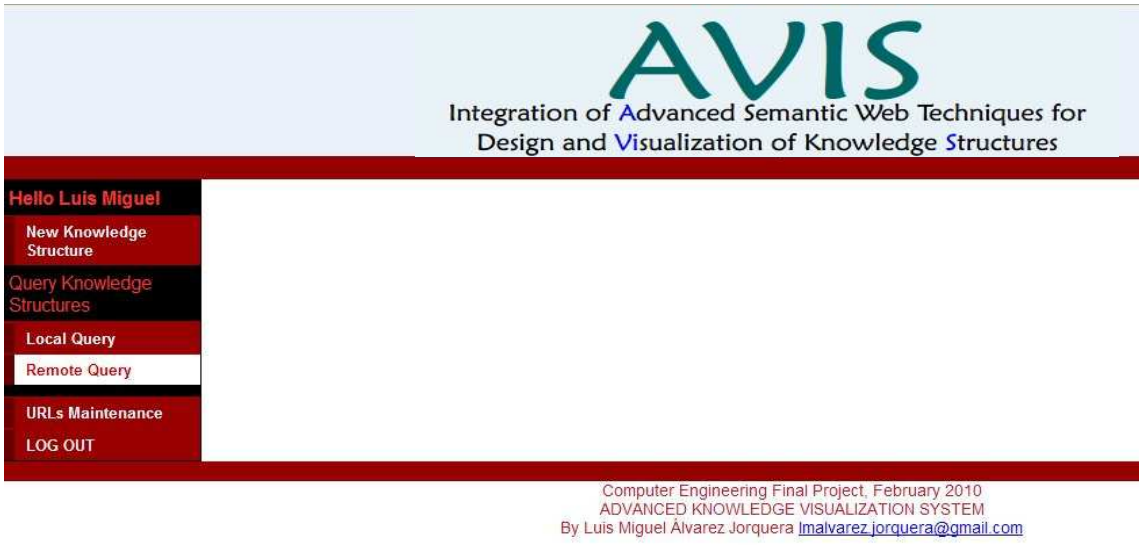
Si pulsamos en Add New Attribute volvemos a la pantalla de nuevos atributos para este concepto, si le pulsamos de Finís Concept se finaliza el concepto y nos muestra como queda hasta el momento la estructura que estamos creando, en un especie de listado como vemos en la siguiente figura:



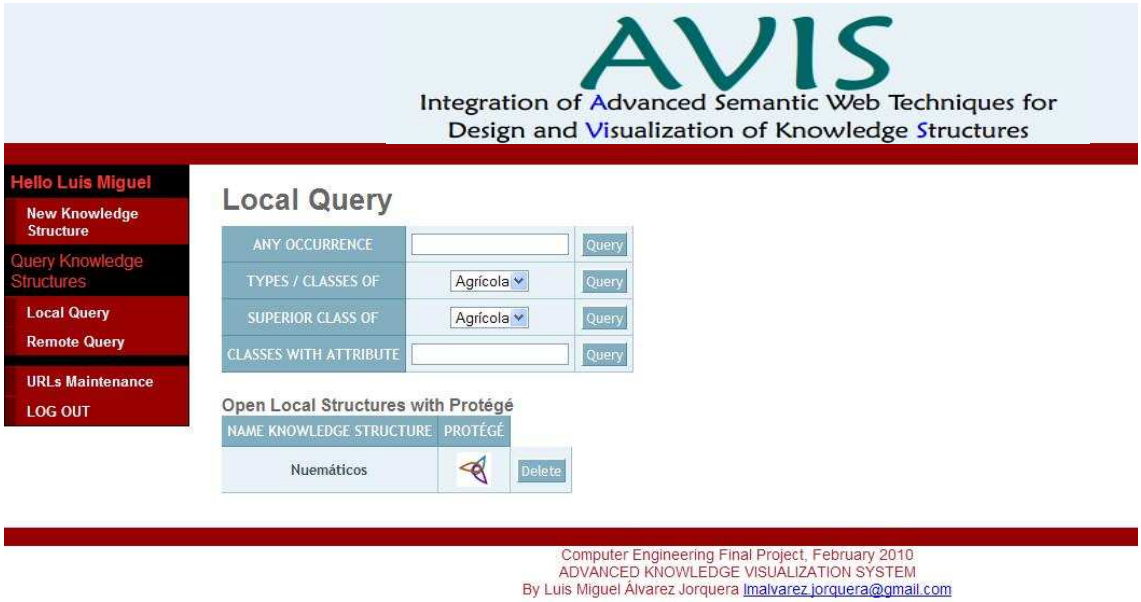
Véase que cada concepto lleva a su izquierda una ristra de asteriscos ********** , esto indica el nivel en el que se encuentra este concepto, por ejemplo el concepto raíz sólo tiene una ristra, sus hijos tendrán 2 ristras, los hijos de los hijos tendrán 3 ristras y así sucesivamente, con lo cual resulta una gran ayuda visual a la hora de crear estructuras muy largas.

Consultas a estructuras de conocimiento.

La siguiente opción en el menú de la izquierda de AVIS es la de consultas que si le pulsamos nos aparece 2 opciones como vemos en la figura siguiente:



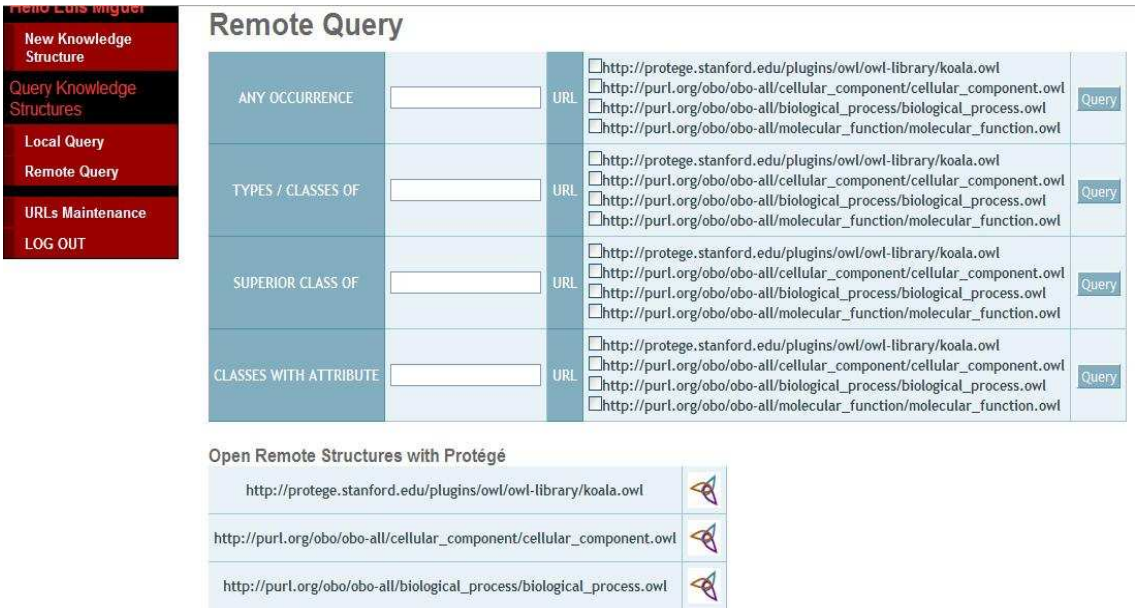
Como ya sabemos tenemos las consultas locales y las consultas remotas, a continuación vemos una imagen de las locales:



Como podemos ver en la imagen anterior AVIS nos ofrece 4 tipo de consultas locales, 2 de ellas tenemos que introducir patrones y las otras 2 nos da a elegir el concepto o patrón de búsqueda a consultar.

AVIS en esta pantalla nos muestra un listado de las estructuras que tiene almacenadas en la base de datos, como vemos en el listado que hay debajo de las 4 consultas que en este caso solamente hay 1, y podemos comprobar como AVIS nos da la opción de abrirlas con Protégé o de borrarlas.

La opción de consultas remotas podemos verlas en la siguiente imagen.



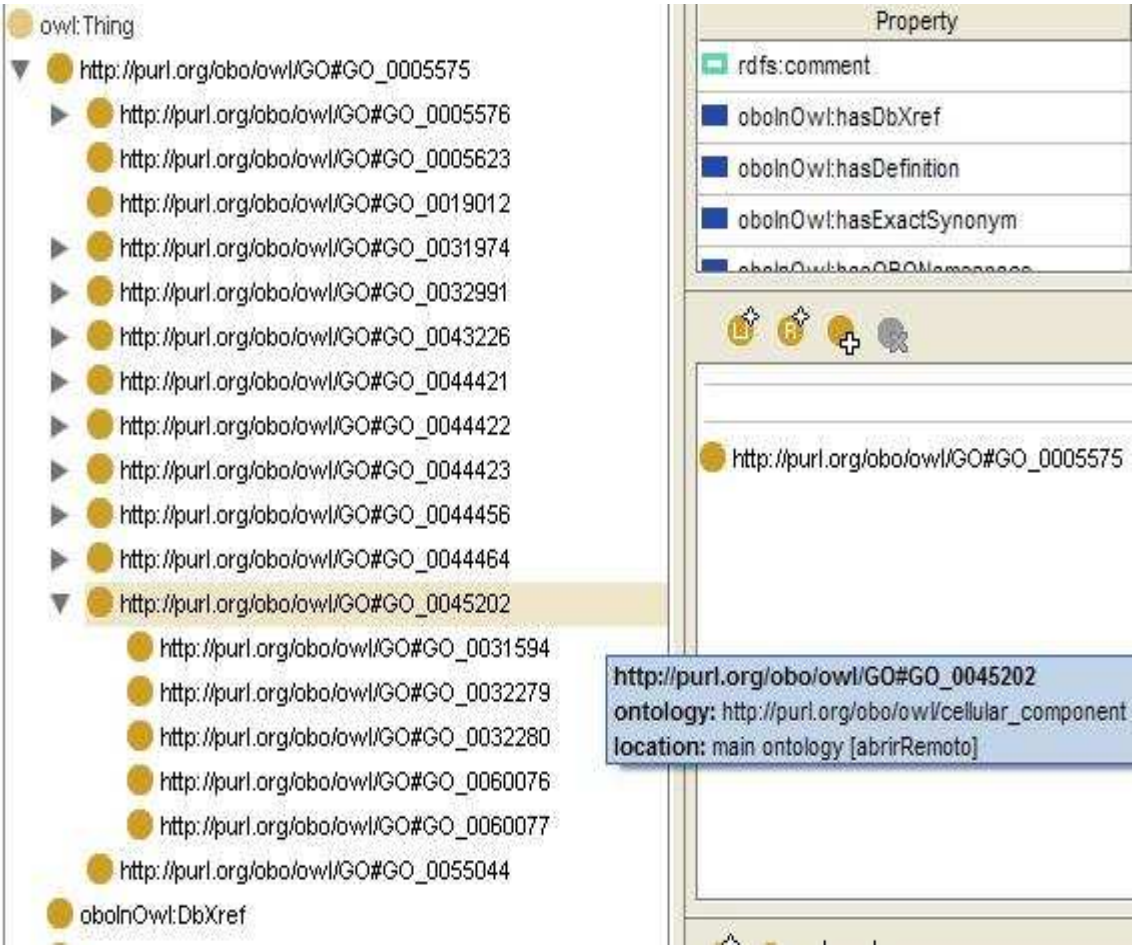
Podemos comprobar como seguimos teniendo 4 tipos de consultas igual que en las consultas locales, pero en este caso es el usuario el que debe de escribir todos los

campos de los patrones de búsqueda. En esta pantalla debemos seleccionar las estructuras remotas que queremos consultar marcando los checkboxes correspondientes. Un ejemplo de consulta a la estructura de gene ontology estudiada en apartados anteriores es el siguiente:

Query Knowledge Structure

GO_0045202 is present in the knowledge structures whose names are:

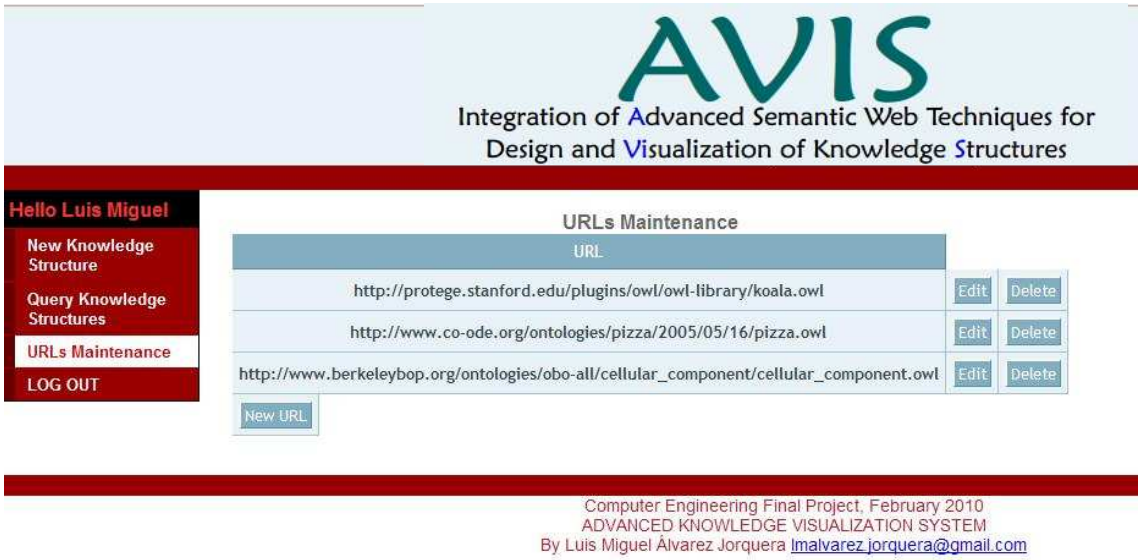
Knowledge Structure: http://www.berkeleybop.org/ontologies/obo-all/cellular_component/cellular_component.owl
GO_0060077
GO_0060076
GO_0032280
GO_0032279
GO_0031594



Concepto GO_0045202 y sus hijos en la jerarquía GO desde Protégé

Mantenimiento de URLs

La siguiente y última opción de AVIS es la de mantenimiento de URLs donde podemos consultar las que tenemos registradas, podemos registrar nuevas URLs, y las que ya tenemos las podemos modificar (porque pueden cambiar) y las podemos eliminar si es necesario. Es tal y como vimos en anteriores apartados.



Y finalmente queda la opción de ***Log OUT*** que finaliza la sesión y vuelve a la pantalla inicial de entrada de usuarios.