POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es

www.industriales.upm.es

05 TRABAJO FIN DE GRADO

INDUSTRIALES

Koro Irusta Gonzalo

TRABAJO FIN DE GRADO

# APRENDIZAJE EN ROBOTS SOCIALES

TRABAJO FIN DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

SEPTIEMBRE **2017**

**Koro Irusta Gonzalo**

**DIRECTOR DEL TRABAJO FIN DE GRADO:**

**Ramón Galán López**

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

*A mi tutor Ramón, por la ayuda recibida y los conocimientos transmitidos.*

*A mi familia, por estar siempre a mi lado.*

*A Marta, por su apoyo y amistad en este camino.*

# RESUMEN DEL PROYECTO

Desde los años 60, han surgido diversas formas de modelar el conocimiento, como es el caso de las redes semánticas, que consisten en la representación del conocimiento a través de un grafo en el que se relacionan los conceptos. En consecuencia, uno de los objetivos del aprendizaje automático, en lo relativo a la ciencia de la computación, es desarrollar técnicas que permitan a las computadoras aprender. En otras palabras, se trata de desarrollar programas con la capacidad de generalizar comportamientos a partir de una información no estructurada en forma de ejemplos, tratándose de una disciplina que se basa en el análisis de datos. Es, por tanto, un proceso de inducción de conocimiento que trata el estudio de la complejidad computacional de los problemas.

Actualmente, la investigación en el campo del aprendizaje automático gira en torno a tres temas, fundamentalmente. El primero son los estudios orientados a tareas, en los que se parte de una tarea y se analizan los sistemas de aprendizaje aplicados para mejorar el desarrollo de la misma. Otro es la exploración teórica de los métodos de aprendizaje existentes independientemente de la tarea que se quiera desarrollar con ellos. Por último, se encuentra la simulación cognoscitiva, que es la investigación y simulación informática del proceso de aprendizaje humano.

Cabe destacar que en el aprendizaje automático es posible obtener tres tipos de conocimiento. Por un lado está el crecimiento, se trata del conocimiento que se va adquiriendo del entorno y se almacena en la memoria; por otro lado, la reestructuración, que se produce cuando el individuo razona al interpretar la información y que genera un nuevo conocimiento. Y por último el ajuste, que se obtiene cuando se generalizan varios conceptos o se generan nuevos conceptos propios. El objetivo de este trabajo es la creación de una base de conocimiento, en este caso en forma de ontología, a través de un aprendizaje automático.

El término ontología proviene del griego y significa conocimiento del ser. Se trata de un término tomado de la filosofía, en la cual la ontología es una rama de la metafísica que estudia cualquier cosa que es o existe. Esta rama de la metafísica se ocupa del estudio de varios tipos de existencia, haciendo especial hincapié en las relaciones entre lo particular y lo universal, la esencia y la existencia. Además, el objetivo de este tipo de análisis consiste en dividir el mundo en conjuntos para descubrir las categorías en las cuales los objetos del mundo están naturalmente.

Sin embargo, el término ontología en el campo de la informática, que es el que nos ocupa, ha adoptado unas connotaciones alejadas de este sentido filosófico original. Desde principios de la década de 1990 han proliferado numerosas definiciones de dicho término.

Una de las primeras definiciones es la acuñada por Neches y otros en 1991 quienes afirmaron lo siguiente:

"Una ontología define los términos básicos y relaciones que constituyen el vocabulario del área de un tema, así como las reglas para combinar términos y relaciones para definir extensiones del vocabulario."

En la citada definición se establece qué hacer para construir una ontología y que no sólo se incluyen términos en ella, sino que también el conocimiento puede ser inferido. Sin embargo, la definición más citada y extendida ha sido la dada por Tom Gruber en 1993:

"Una ontología es una especificación explícita de una conceptualización. El término proviene de la filosofía, donde una ontología es un recuento sistemático de la existencia. En sistemas de Inteligencia Artificial, lo que existe es lo que puede ser representado."

En resumen, podemos concluir que una ontología es un esquema conceptual dentro de uno o varios dominios, con la finalidad de facilitar la comunicación entre sistemas o entidades, que posibilita almacenar, buscar y recuperar información. Además, define los términos y las relaciones básicas para la comprensión de un área de conocimiento, así como las reglas que permiten combinar los diferentes términos para definir un vocabulario.

A pesar del rigor de las definiciones anteriores, es posible afirmar que todos poseemos ontologías en nuestro cerebro en las cuales se representa lo que se observa a nuestro alrededor. Por ejemplo, cuando citamos avión en nuestra mente estamos representando un medio de transporte que vuela. Esto normalmente no se formaliza debido a que todos entendemos este tipo de conceptos y de asociaciones, sin embargo, las máquinas carecen de estas ontologías y es necesario desarrollarlas en documentos o con diferentes metodologías y métodos para que, las máquinas, sean capaces de entender mejor el mundo y de comunicarse.

Para la creación de una ontología se requiere un lenguaje lógico y formal. En este caso, se ha utilizado el lenguaje OWL, Ontology Web Language, que está basado en el lenguaje RDF pero posee una mayor capacidad expresiva. Surge como revisión al lenguaje DAML-OIL para que sirva como estándar para los investigadores de la Web semántica. El OWL es el resultado del trabajo del Web Ontology Working Group (W3C), que se formó en noviembre de 2001. Uno de sus principales usos deriva de la necesidad de la información almacenada en los documentos de ser procesada bien por programas o bien por aplicaciones.

Este lenguaje también utiliza el modelo de tripletas del RDF, aunque con un mayor poder expresivo puesto que se puede utilizar tanto para representación de los términos como de las relaciones entre los mismos, fundamental a la hora de realizar una ontología. Además incluye propiedades como restricciones entre clases, inclusiones o equivalencias, así como alguna funcionalidad para el razonamiento que amplían el lenguaje RDF y sobre todo, sus usos. También permite expresar la cardinalidad, es decir, el número de elementos que puede componer una clase, o un objeto.

OWL se separó en tres versiones distintas en función de su expresividad semántica, siendo éstas OWL Lite, la versión más simple que extiende RDF y reúne las características más comunes de OWL, por lo que está destinada a quienes quieren crear clases y utilizar las relaciones más sencillas; OWL DL, que incluye el vocabulario completo de OWL y por último OWL Full que incluye todo el vocabulario y carece de limitaciones para explorar su potencial.

Además, este lenguaje permite la interoperabilidad y posee propiedades que facilitan la importación y exportación de clases, algo que resulta sumamente útil cuando se integran o se mezclan diversas ontologías. Asimismo, las ontologías que se crean con OWL son documentos web y por tanto, se referencian gracias a un identificador de recursos uniforme (URI), aspecto fundamental, puesto que esto permite la distribución de la ontología que se desarrolle.

También es necesario un editor de ontologías. En este caso, se ha seleccionado Protégé, herramienta gratuita creada en la Universidad de Stanford y probablemente la herramienta de

construcción de ontologías más utilizada. Se trata de una herramienta de software integrado, de código abierto, que permite la creación de sistemas basados en el conocimiento. Sus primeras versiones datan de 1998.

Con esta herramienta se puede crear una ontología, insertar datos en la misma y hasta optimizar los datos de entrada. Además el editor posee una biblioteca de extensiones que permite aumentar sus funcionalidades. Contiene un conjunto de estructuras para el modelado del conocimiento y diversos formatos para la representación y visualización de los contenidos. Con este editor es posible manejar ontologías con más de 100.000 conceptos.

Su código Java es libre y ofrece el acceso a la API utilizada en su desarrollo con la que poder crear aplicaciones Java para la manipulación de ontologías. Esto es algo fundamental de cara a la aplicación del diseño de bases de conocimiento para agentes artificiales a través de su representación mediante ontologías.

Una de las utilidades de esta herramienta a destacar es que permite la visualización de las ontologías, lo que facilita la interacción con los seres humanos, ya sea en las labores de inspección o en la comprensión por parte de personas que no estén familiarizadas con este tema.

Este editor ofrece la posibilidad de tener dos formas de modelar el conocimiento, el editor Protégé-Frames, que permite la construcción de ontologías basadas en marcos, o el editor Protégé-OWL que permite construir ontologías para la web semántica en lenguaje OWL y que es la funcionalidad escogida para la realización del proyecto.

Protégé-OWL permite abrir y guardar las ontologías creadas en ficheros con extensión *.owl o bien desde una dirección de internet. Este tipo de ontologías también posee clases, propiedades e instancias y facilita la inferencia de conocimiento deducible a través de las relaciones semánticas. Se ha seleccionado este entorno puesto que el poder crear un archivo *.owl facilita su utilización por diversos sistemas. Es por esto que ha sido la herramienta seleccionada para permitir la interacción humana con la base de conocimiento creada, y permitir la creación de los ficheros en los cuales se almacenará la información de la misma.

Por otro lado, la ontología que se ha creado en el proyecto está orientada a ser utilizada en un diálogo robot-humano, luego está pensada para ser implantada en los robots sociales. Un robot social es un robot autónomo que interactúa y se comunica con las personas, de una manera sencilla y agradable siguiendo una serie de comportamientos y normas sociales. Para que esta interacción robot-humano sea posible, es necesario que además de poseer sistemas perceptivos robustos, posea una serie de habilidades que se agrupan dentro de lo que se conoce como inteligencia social. Para esta interacción social, el robot debe poseer un modelo cognitivo-afectivo, es decir, capacidades como la comunicación, la comprensión y el aprendizaje, siendo este último el objeto de estudio de este documento.

Existen diversos tipos de robots sociales, los robots socialmente evocativos (aquellos que se antropomorfizan para animar a las personas a interactuar con la tecnología), robots de interfaz social, socialmente receptivos y sociables. En este trabajo son interesantes estos últimos, que son aquellos que con sus propias metas y motivaciones internas tratan de involucrar a las personas de una manera social no sólo para el beneficio de las mismas, sino para beneficiarse ellos mismos, tratando por tanto de modelar a las personas en términos sociales y cognitivos para interactuar con ellos.

Para la creación de un programa que permita la realización de un aprendizaje automático es fundamental poseer una base de conocimiento en la que ir almacenando lo aprendido. Por ello, a través del editor de ontologías Protégé, se ha creado una ontología en la que se irán guardando los conceptos que el robot vaya aprendiendo.

Este aprendizaje es posible de dos formas, en primer lugar, el programa permite el tratamiento de un fichero de texto en forma de tripletas que puede ser gestionado por el usuario desde una interfaz gráfica. En segundo lugar, se ha creado un bucle de aprendizaje en el que el agente puede realizar búsquedas en la enciclopedia Wikipedia. Esta enciclopedia proporciona una información general en cada artículo, sin embargo gracias al acceso a Wikidata, almacenamiento estructurado de los datos de los proyectos hermanos del grupo Wikimedia, se obtiene una información más concreta y más sencilla de clasificar. Este aprendizaje automático puede ser para ampliar algún conocimiento sobre algo que se haya almacenado previamente en la ontología, o para añadir conocimiento totalmente nuevo mediante una búsqueda de contenido aleatoria en Wikipedia o por motivación del usuario. También es posible realizar un aprendizaje sistemático de dicha enciclopedia.

Además, esta información obtenida es previamente analizada por Freeling (analizador sintáctico) para poder reducir la complejidad de las frases que se obtienen de la misma y reescribirlo en forma de tripletas para que pueda ser añadido a la ontología. Los términos que se introducen en la misma van acompañados por un índice de confianza que nos indica la fiabilidad de esos conceptos según los parámetros que se han establecido para ello.

Este aprendizaje es gestionado mediante un sistema de ficheros que van almacenando la información en cada uno de los pasos. En primer lugar, de la búsqueda de contenido en Wikipedia, se extrae un fichero que contiene las sentencias con la información de los términos buscados al que se ha denominado input.txt. Por otro lado, una vez que este fichero pasa por el analizador sintáctico, se obtiene otro fichero de salida, al que se ha denominado output.txt. Este fichero de salida contiene el análisis sintáctico de toda esta información. En él la primera palabra es la palabra que toma del fichero input.txt, la segunda es la palabra genérica que hace referencia a la anterior; por ejemplo si se tiene la palabra es, la segunda palabra que aparece será el verbo ser. Por último, en tercer lugar aparece el tipo de palabra de la que se trata: verbo, sintagma nominal, adjetivo... proporcionando también información sobre el género y número si procede; la cuarta palabra es el índice de confianza del análisis de dicho término.

Este fichero output.txt se pasa a la clase TripletaAnalizador. En esta clase lo que se hace es leer este fichero de salida del analizador y extraer una tripleta en base a esta salida. Una vez que se tienen creadas las tripletas, este archivo tiene que ser añadido a la ontología. Para ello lo que se ha hecho es crear una clase, TratFrases, en la que se realiza esta acción.

Si la tripleta no cumpliese con la estructura deseada, lo que se hace es copiar esta frase a un fichero denominado rechazados.txt y no se almacena aún en la ontología. De este modo, en un futuro se puede buscar más información sobre las frases que vayan siendo rechazadas y de esta forma generar tripletas correctamente para ser añadidas a la ontología. Este sería un proceso que podría ser llevado a cabo en el robot en momentos de inactividad, fomentando la capacidad de analizar por qué no ha entendido antes esa información y realizando búsquedas en internet que le permitan comprender esos conceptos.

Por otro lado, si se observa que las tripletas contienen la estructura deseada y cumplen los requisitos, son añadidas a la ontología. Una vez que son añadidas, estas tripletas se copian

en un fichero denominado historicos.txt. Este fichero contendrá todas las tripletas que se han ido procesando en todos los aprendizajes realizados.

Todo este aprendizaje es gestionado desde una interfaz gráfica desde la que el usuario puede ver también estadísticas sobre la ontología . Esta opción permite conocer el número de clase, instancias y relaciones que conforman la ontología. Así mismo, es posible consultar si existe en ella algún término, y en caso afirmativo el programa proporciona información sobre qué papel ocupa en ella (clase, relación o instancia). También es posible mostrar las clases de una relación o las instancias de una clase.

Para facilitar la comunicación con el robot Urbano, el robot social desarrollado por el grupo de investigación de Control Inteligente de la Escuela Técnica Superior de Ingenieros Industriales, para quien en un principio ha sido desarrollado este trabajo, el programa posee una función que permite también la comunicación vía TCP.

Es importante destacar que con esto queda resuelta la memoria a medio y largo plazo en los robots sociales, puesto que dota al agente de la capacidad de aprender de una forma autónoma, con búsquedas en internet y almacenar los conocimientos para que pueda usarlos cuando los necesite. De este modo, se intenta convertir a los robots sociales en algo más sociables, ya que les permite interactuar con las personas no sólo para el beneficio de las mismas, si no para el suyo propio, intentando satisfacer alguna motivación interna.

En última instancia, a través del programa lo que se obtiene es un archivo con la extensión *.owl que contiene una base de conocimiento. La ontología se puede ir actualizando y es posible distribuirla para su implantación en robots sociales o en otros agentes.

El trabajo realizado permite demostrar la capacidad de aprendizaje en un robot social mediante búsquedas en la enciclopedia Wikipedia o por el tratamiento de un fichero de texto en formato de tripletas. La información que se obtiene de Wikipedia es una información general de cada artículo, sin embargo con el acceso a Wikidata se puede obtener una información concreta que permite su clasificación.

En un futuro, el trabajo que se plantea es modelar el aprendizaje en función de los gustos o preferencias del usuario.

*Palabras clave: control inteligente, robots sociales, ingeniería emocional, interacción humano robot, modelo emocional.*

# 1. INTRODUCCIÓN

## 1.1. SITUACIÓN ACTUAL DEL CONOCIMIENTO

El razonamiento que poseen los seres humanos tiene como factor fundamental el sentido común, que los ayuda a prever situaciones y a encontrar soluciones. Sin embargo, en máquinas o robots el razonamiento se hace muy complicado de implementar debido a que las reglas son en muchos casos inexactas y a menudo son verdades incompletas.

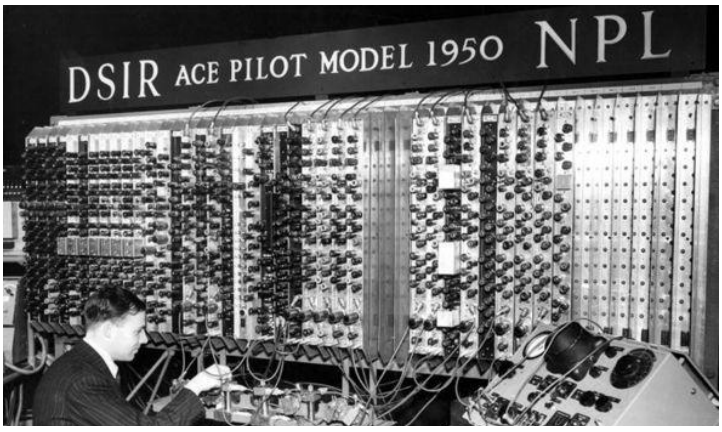
En este proyecto se va a abordar el aprendizaje en robots sociales, para ello es necesario comenzar desarrollando la evolución de la robótica y del conocimiento a lo largo del tiempo.

### 1.1.1. Inteligencia artificial (AI).

La inteligencia artificial estudia la creación y el diseño de sistemas capaces de resolver problemas cotidianos por sí mismos utilizando como paradigma la inteligencia humana. Es decir, se encarga de crear procesos que al ser ejecutados sobre una arquitectura física, produzcan acciones o resultados que maximicen una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en dicha arquitectura. Para ello, se vale de ciencias como las matemáticas, la computación, la lógica y la filosofía.

Para contextualizar la inteligencia artificial hay que remontarse muchos años atrás. Se consideran antecedentes de la inteligencia artificial algunos estudios como el realizado cerca del año 300 a.C por Aristóteles, en el que definió los silogismos, una forma de razonamiento deductivo que consta de dos proposiciones como premisas y otra como conclusión, siendo la última una inferencia necesariamente deductiva de las otras dos. Otro estudio a destacar sería el de Geoge Boole en 1847 sobre la lógica booleana. El desarrollo de este tipo de ideas se vería impulsado años más tarde gracias a Alan Turing que además de construir el primer computador electromecánico, en 1950 consolidó el campo de la AI con el artículo "*Computing Machinery and Intelligence*". En dicho artículo proponía una prueba concreta para determinar si una máquina era o no inteligente, la conocida Prueba de Turing, por lo que se le considera el padre de esta ciencia.

**Figura 1. Computadora de Alan Turing.**

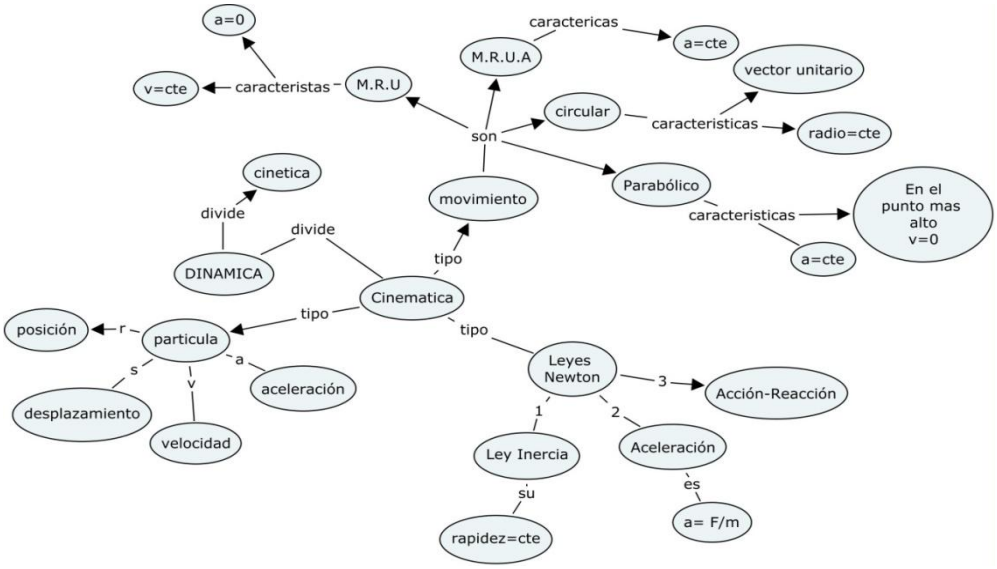



Fuente: Una Historia curiosa.

No fue hasta 1956 en la conferencia de Dartmouth cuando se creó el término de Inteligencia Artificial. En este congreso se hicieron previsiones triunfalistas a diez años con objeto de crear una máquina que fuese más allá del cálculo numérico, es decir, se deseaba que las computadoras llevasen a cabo funciones inteligentes. Estas previsiones que incluían reuniones periódicas no se cumplieron, lo que llevó a un abandono de estas investigaciones durante unos quince años.

En los años 60, se desarrollaron las redes semánticas como forma de modelar el conocimiento, que consisten en la representación del conocimiento a través de un grafo en el que se relacionan los conceptos, y que son muy utilizadas en este ámbito.

**Figura 2. Ejemplo de red Semántica.**




Fuente: hikaruka.wordpress.com

Además, en esta época, comienzan a aparecer los Sistemas Expertos, que son programas ideados para la resolución de problemas relacionados con asuntos especialmente complicados y que requieren un elevado nivel de conocimiento. Pasados unos años, en 1987 cabe destacar la descripción de los atributos de un agente inteligente que hicieron Martin Fischles y Oscar Firschein:

1. Tiene actitudes mentales tales como creencias e intenciones.
2. Posee la capacidad de obtener conocimiento, es decir, aprender.
3. Puede resolver problemas, incluso particionando problemas complejos en otros más simples.
4. Entiende. Posee la capacidad de crearle sentido, si es posible, a ideas ambiguas o contradictorias.
5. Planifica, predice consecuencias, evalúa alternativas (como en los juegos de ajedrez)
6. Conoce los límites de sus propias habilidades y conocimientos.
7. Es capaz de distinguir a pesar de la similitud de las situaciones.
8. Puede ser original, creando incluso nuevos conceptos o ideas, y hasta utilizando analogías.
9. Puede generalizar.
10. Es capaz de percibir y modelar el mundo exterior.

Actualmente, se ha avanzado con la creación de agentes inteligentes, sin embargo aún estamos lejos de cumplir la famosa prueba de Turing: " Existirá Inteligencia Artificial cuando no seamos capaces de distinguir entre un ser humano y un programa de computadora en una conversación a ciegas".

### 1.1.2. Aprendizaje automático.

El aprendizaje puede definirse como la capacidad de adquirir nuevos conocimientos, pero también es aprendizaje el desarrollo de habilidades cognoscitivas y motoras, a través de la práctica o el descubrimiento de nuevas teorías o factores, mediante la observación o experimentación. Resolver este problema es uno de los objetivos de la inteligencia artificial a largo plazo. El estudio y el modelado en forma de programas de estos tipos de aprendizaje es el objeto de investigación del aprendizaje automático.

Por tanto, uno de los objetivos del aprendizaje automático, en lo relativo a la ciencia de la computación, es desarrollar técnicas que permitan a las computadoras aprender. En otras palabras, se trata de desarrollar programas con la capacidad de generalizar comportamientos a partir de una información no estructurada en forma de ejemplos, tratándose de una disciplina que se basa en el análisis de datos. Es por ello un proceso de inducción de conocimiento que trata el estudio de la complejidad computacional de los problemas.

Actualmente, la investigación en el campo del aprendizaje automático gira en torno a tres temas fundamentalmente. El primero son los estudios orientados a tareas, en el que se parte de una tarea y se analizan los sistemas de aprendizaje aplicados para mejorar el desarrollo de la misma. El segundo es la exploración teórica de los métodos de aprendizaje existentes, independientemente de la tarea que se quiera desarrollar con ellos. Por último, en el tercero se encuentra la simulación cognoscitiva, que es la investigación y simulación informática del proceso de aprendizaje humano.

Hay dos formas básicas de aprender, adquiriendo conocimiento o por refinamiento de habilidades, siendo el primero el fundamental en el área de inteligencia artificial. Para adquirir dicho conocimiento se suelen barajar dos posibilidades, no realizar inferencia, que en este caso el conocimiento se incrementa pero el esfuerzo cognoscitivo corre a cuenta del programador, o realizar una inferencia, de manera que si el sistema descubre nuevos conceptos o ideas, a través de la inferencia se disminuye la carga puesta en el maestro.

Dependiendo del enfoque que se adopte, existen varias formas de que se produzca el aprendizaje automático en función del tipo de conocimiento adquirido:

- Árboles de decisión: se trata de una estructura informática básica que se utiliza en este caso como modelo predictivo. A partir de una base de datos se crea un diagrama de construcciones lógicas que sirven para representar y categorizar una serie de condiciones que ocurren de manera sucesiva a la hora de resolver un problema. Por tanto, a partir de unas entradas, el árbol devuelve una respuesta que será función de dichas entradas. Los nodos de dichos árboles corresponden a atributos de los objetos y los extremos a valores predeterminados de las alternativas para esos atributos. Las hojas del árbol serán los conjuntos de objetos con la misma clasificación.

- Reglas de asociación: se utilizan para descubrir relaciones entre una serie de datos. Se han desarrollado diversos algoritmos que realizan búsquedas de reglas de asociación en bases de datos como por ejemplo el algoritmo Apriori o el Eclat.

- Algoritmos genéticos: se denomina algoritmo a una serie de pasos organizados que describe el proceso a seguir para resolver un problema determinado. Los algoritmos genéticos se inspiran en la evolución biológica, usando métodos como la mutación o el cruzamiento para generar nuevas clases que sean solución para los problemas planteados.

- Algoritmos de agrupamiento: es una técnica de análisis de tipo no supervisado, que consiste en la clasificación de observaciones en subgrupos según criterios como pueden ser la similitud o la distancia entre diferentes grupos.

- Redes Bayesianas: es un modelo grafo probabilístico que representa un conjunto de variables y sus dependencias condicionales a través de un grafo acíclico dirigido.

Por último, cabe destacar que en el aprendizaje automático es posible obtener tres tipos de conocimiento. Por un lado está el crecimiento, que se trata del conocimiento que se va adquiriendo del entorno y que se almacena en la memoria; por otro lado la reestructuración, que se produce cuando el individuo razona al interpretar la información y genera un nuevo conocimiento y además el ajuste, que se obtiene cuando se generalizan varios conceptos o se generan nuevos conceptos propios. Dicho esto, es el momento de centrarse en el área del conocimiento.

### 1.1.3. Ingeniería del conocimiento.

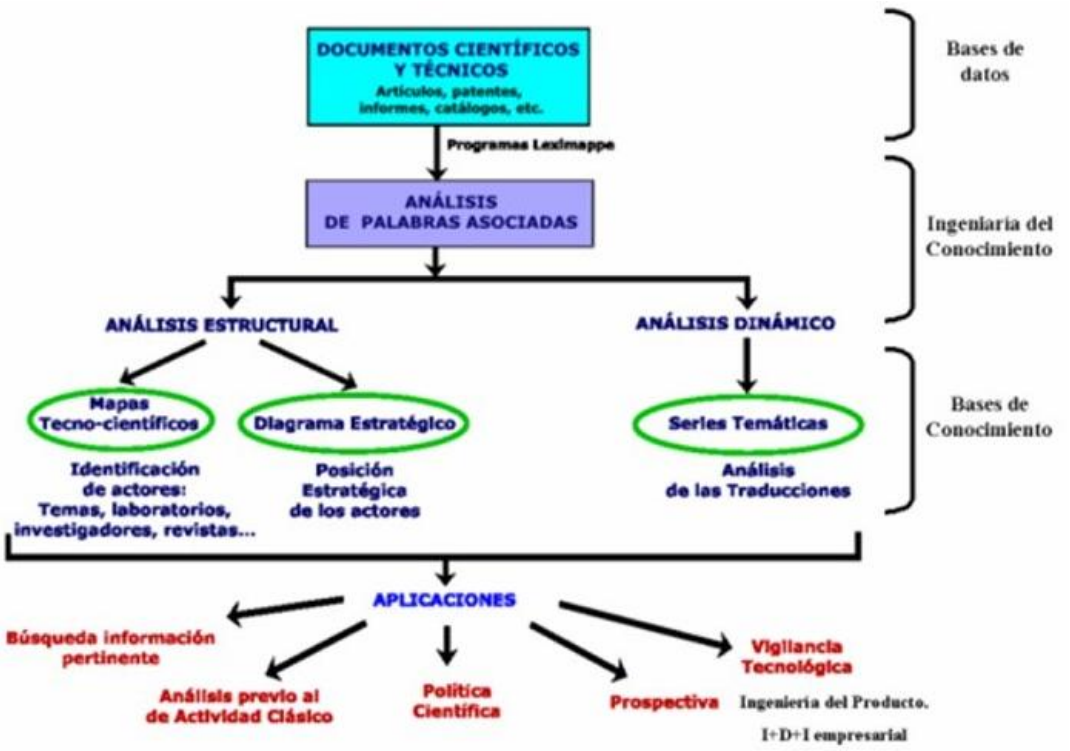
Dentro del área de Inteligencia Artificial, en este proyecto, se va a tratar la ingeniería del conocimiento, que va encaminada a sistematizar y apoyar primero el proceso de construcción de modelos, y posteriormente, la reducción de esos modelos a programas. Consiste en generar un nuevo conocimiento a partir de la información contenida en bases de datos documentales y mediante el cruce del contenido de los documentos, partiendo de la teoría actor-red y poniendo de manifiesto redes o generando otras nuevas. Las redes son descritas matemáticamente mediante la teoría de grafos y para determinar la intensidad de los enlaces se emplea el índice de similitud o cercanía, utilizándose el análisis de palabras asociadas, que se basa en leer los documentos y generar redes socio-cognitivas a partir de la asociación de las palabras que hay en los documentos. Esto se lleva a cabo mediante softwares específicos, originándose de esta manera las Bases de Conocimiento en oposición a las Bases de Datos, que únicamente contienen datos.

Hay tres tipos de conocimiento que debemos tener en cuenta. Por un lado, el **conocimiento declarativo**, es aquel tipo de conocimiento pasivo, que se basa en sentencias para expresar hechos del mundo que nos rodea. Posee una mayor capacidad expresiva y menor capacidad creativa o computacional. Un ejemplo serían las bases de datos. Se pueden representar con modelos relacionales (en forma de grafos, árboles o redes semánticas) o mediante esquemas basados en lógica (tanto posicional como de predicados). Por otro lado, el **conocimiento procedimental**, es aquel que es compilado y se refiere a una forma de realizar una tarea, es decir, el saber cómo hacerlo. Pueden estar caracterizados por gramáticas

formales, usualmente implantadas por sistemas o lenguajes procedimentales y sistemas basados en reglas. Por último, el **conocimiento heurístico**, es algo especial para resolver problemas complejos. Se trata de un criterio, estrategia, método o proceso que simplifica la resolución de problemas.

Estos tres tipos de conocimiento, mediante declaraciones y procedimientos, nos permiten representar el conocimiento en la AI, de manera que pueda ser aplicado a la adquisición de nueva información para un sistema, a la clasificación de conocimiento para obtener hechos y relaciones particulares asociados a una tarea específica o al razonamiento, es decir, al proceso en el que se combina el conocimiento almacenado.

**Figura 3. Ingeniería del conocimiento.**




Fuente: Cognosfera.

Para poder realizar una buena representación del conocimiento se debe tener en cuenta que deben aparecer tanto los objetos como las relaciones de forma conjunta. Además se debe realizar de forma clara y concisa, siendo fundamental la modularidad y el entendimiento, y debe ser rápida y computable. Por ello, el tratamiento del conocimiento en este proyecto se va a llevar a cabo mediante el uso de las ontologías, donde se englobarán los tres tipos de conocimiento desarrollado y se llevarán a cabo estas prácticas para obtener una estructura que permita a las máquinas realizar aprendizajes automáticos.

## 1.2. ONTOLOGÍAS.

### 1.2.1. Definición de Ontología.

El término ontología proviene del griego y significa conocimiento del ser. Se trata de un término tomado de la filosofía, en la cual la ontología es una rama de la metafísica que estudia cualquier cosa que es o existe. Esta rama de la metafísica se ocupa del estudio de varios tipos de existencia, haciendo especial hincapié en las relaciones entre lo particular y lo universal, la esencia y la existencia. Además, el objetivo de este tipo de análisis consiste en dividir el mundo en conjuntos para descubrir las categorías en las cuales los objetos del mundo están de forma natural.

Sin embargo, el término ontología en el campo de la informática, que es el que nos ocupa, ha adoptado unas connotaciones alejadas de este sentido filosófico original. Desde principios de la década de 1990 han proliferado numerosas definiciones de dicho término. Una de las primeras definiciones es la acuñada por Neches y compañeros en 1991, quienes dijeron lo siguiente:

"Una ontología define los términos básicos y relaciones que constituyen el vocabulario del área de un tema, así como las reglas para combinar términos y relaciones para definir extensiones del vocabulario."

En la citada definición se establece qué hacer para construir una ontología. Además muestra que no sólo se incluyen términos en ella, sino que también el conocimiento puede ser inferido. Sin embargo, la definición más citada y extendida ha sido la dada por Tom Gruber en 1993:

"Una ontología es una especificación explícita de una conceptualización. El término proviene de la filosofía, donde una ontología es un recuento sistemático de la existencia. En sistemas de Inteligencia Artificial, lo que existe es lo que puede ser representado."

Esta definición tiene su fundamento en la idea de conceptualización propuesta por Genesereth y Nilson en 1987, que se refiere al conjunto de conceptos y de relaciones entre éstos que son relevantes en un dominio o aplicación. Los términos explícita y formal hacen alusión a que los conceptos deben ser claramente definidos y procesables por una máquina. Esta definición fue clarificada por Borst en 1997 enfatizando en dos ideas, en que una ontología debe ser especificada usando un lenguaje formal que pueda ser procesado por ordenadores y que se trata de conocimiento compartido, fruto del consenso dentro de un grupo.

Han sido muchas las definiciones alternativas de ontología que se han propuesto a lo largo de los años, como las dadas por Nicola Guarino, según el cual las ontologías utilizadas en aplicaciones reales son realmente adaptaciones ingenieriles de ontologías, no ontologías propiamente dichas. Pero si se quiere una definición más formal, es importante citar partes del documento de Web Ontology Language de la W3C que dice que:

"Una ontología define los términos a utilizar para describir y representar un área de conocimiento. Las ontologías son utilizadas por las personas, las bases de datos y las aplicaciones que necesitan compartir un dominio de información (un dominio es simplemente un área de temática específica o un área de conocimiento, tales como medicina, fabricación de herramientas, reparación automovilística entre otras). Las ontologías incluyen definiciones de

conceptos básicos del dominio, y las relaciones entre ellos, que son útiles para los computadores. Codifican el conocimiento de un dominio y también el conocimiento que extiende los dominios. En este sentido, hacen el conocimiento reutilizable."

En resumen, podemos concluir que una ontología es un esquema conceptual dentro de uno o varios dominios, con la finalidad de facilitar la comunicación entre sistemas o entidades, que posibilita almacenar, buscar y recuperar información. Además, define los términos y las relaciones básicas para la comprensión de un área de conocimiento, así como las reglas que permiten combinar los diferentes términos para definir un vocabulario.

A pesar del rigor de las definiciones anteriores, es posible afirmar que todos poseemos ontologías en nuestra cabeza, en las cuales se representa lo que se observa a nuestro alrededor. Por ejemplo, cuando citamos avión en nuestra cabeza estamos representando un medio de transporte que vuela. Esto normalmente no se formaliza debido a que todos entendemos este tipo de conceptos y de asociaciones, sin embargo, las máquinas carecen de estas ontologías y es necesario desarrollarlas en documentos o con diferentes metodologías y métodos que se detallarán más adelante para que, las máquinas, sean capaces de entender mejor el mundo y de comunicarse.

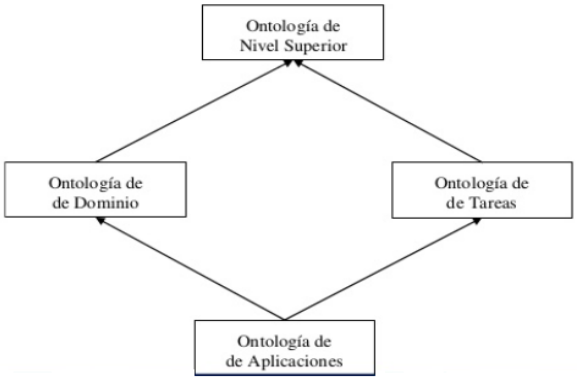
### 1.2.2. Tipos de ontologías.

Una vez que se conoce el concepto de ontología, es fundamental analizar los diferentes tipos que pueden encontrarse. Se pueden clasificar según el conocimiento que contienen o el alcance de su aplicabilidad.

Según el conocimiento podemos encontrar:

- Ontologías terminológicas: especifican los términos utilizados para representar conocimiento en un dominio determinado. Uno de sus principales usos es para unificar vocabulario en un ámbito.
- Ontologías de información: especifican la estructura de los registros de la base de datos.
- Ontologías para modelar el conocimiento: especifican conceptualizaciones del conocimiento. Son el tipo de ontologías que posee una estructura interna más rica y son las que interesan a los desarrolladores de sistemas basados en conocimiento.

**Figura 4. Clasificación de las ontologías según el alcance.**




Fuente: Guarino(1998).

Según el alcance, encontramos los siguientes tipos de ontologías que podemos observar en la figura y que se concretan a continuación:

- Ontologías de dominio: proporcionan el léxico necesario para describir un ámbito determinado, como puede ser el arte, el sector inmobiliario, etc.
- Ontologías generales: representan una serie de conceptos generales, que no son específicos de un dominio determinado. Éstas contienen descripciones generales sobre objetos, relaciones temporales o modelos de comportamiento entre otros.
- Ontologías de tareas: establecen la forma en que puede utilizarse el conocimiento del dominio para realizar una actividad determinada.
- Ontologías de aplicación: son aquellas que describen conceptos que dependen tanto de una ontología de dominio como de una de tarea a través de la especialización de los conceptos definidos en éstas. Dependen de la aplicación.

Como se ha podido observar, contamos con numerosos tipos de ontologías, pero en lo relativo a este proyecto se desarrollará una ontología general, puesto que se quiere dotar a un robot social de un aprendizaje automático y se pretende que abarque tantos dominios como sea capaz de investigar.

### 1.2.3. Elementos de las ontologías.

A la hora de representar el conocimiento en una ontología, existen diferentes componentes:

Los conceptos o clases son las ideas básicas de las cuales se quiere obtener una representación formal. Constituyen elementos definidos por unas características comunes a todos los miembros. Estas entidades pueden ser cosas físicas o conceptuales y constituyen el núcleo de la ontología. Es posible incluir unas clases en otras estableciendo así una jerarquía de clases, es decir, una clase puede tener subclases que representan conceptos más específicos que el de su superclase.

Por otro lado, se utilizan las instancias para representar objetos de un concepto o elementos particulares de una clase, siendo por tanto el nivel más básico de una ontología. Incluyen elementos específicos del dominio de ésta, como personas, animales, coches, etc. La decisión de dónde acaban las clases y comienzan las instancias depende de los elementos más específicos que se quieren representar en la base de conocimiento, puesto que a ellos deben referirse las instancias.

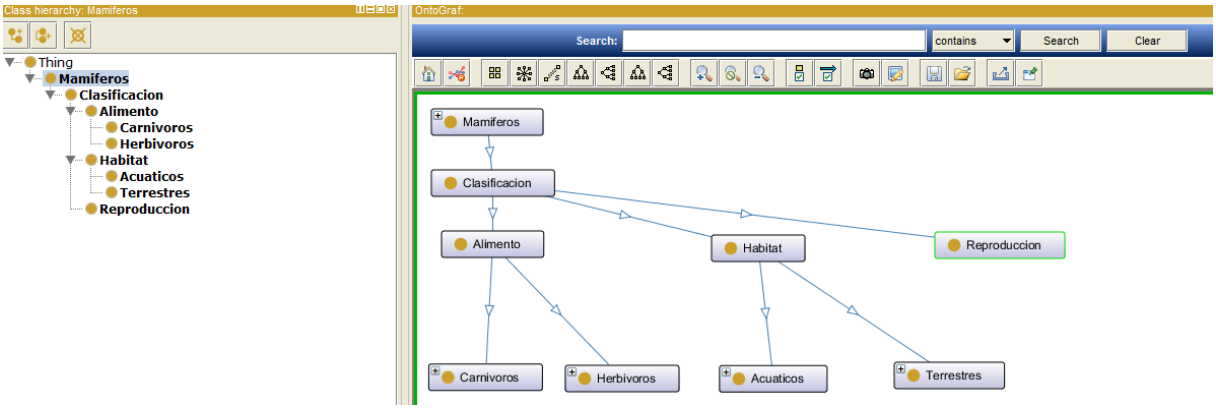
Las propiedades son atributos que poseen las entidades que pertenecen a una clase y que sirven para describir las características más relevantes de los elementos que constituyen las clases o de las instancias.

La unión de los conceptos de un ámbito del conocimiento se realiza a través de las relaciones, que representan un tipo de interacción como puede ser "parte de", "subclase de"..... Además, las funciones son un tipo de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología. Podemos encontrar relaciones que asocian individuos o instancias a una determinada clase con relaciones como "es un" o que asocian clases con subclases. También existen relaciones para asociar propiedades tanto a clases como a instancias, siendo un ejemplo de ello la relación "tiene el valor".

Por último, se encuentran los axiomas o reglas, que son una serie de teoremas que se declaran sobre las relaciones y que deben cumplir uno o varios de los elementos de la ontología. Éstos son los que permiten inferir conocimiento que no esté indicado explícitamente en la taxonomía de conceptos.

A continuación se muestra un ejemplo de una ontología que ha sido creada con el programa Protégé, en la cual se pueden identificar los elementos descritos con anterioridad.

**Figura 5. Ejemplo de Ontología creada con Protégé.**



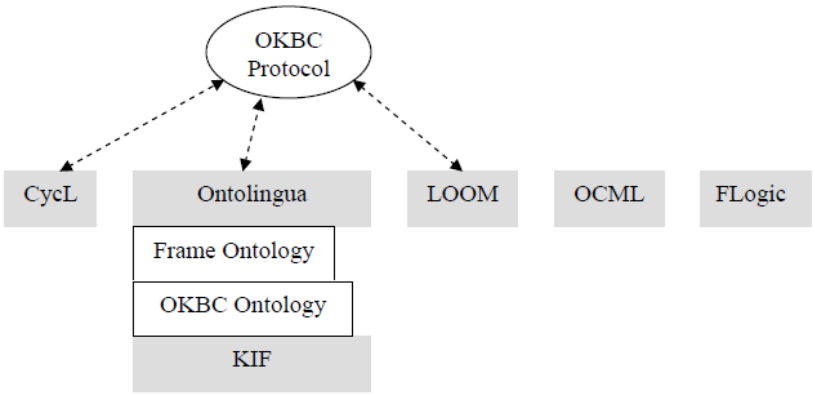

### 1.2.4. Lenguajes para construir ontologías.

Las ontologías requieren de un lenguaje lógico y formal para poder ser expresadas, puesto que deben ser comprensibles para las máquinas. Por ello, en la inteligencia artificial, se han desarrollado diversos lenguajes para lograr dicho fin, siendo muchos de ellos propuestos por los desarrolladores de las diversas herramientas ontológicas. No todos ellos han sido creados para permitir la interacción entre máquinas y humanos, si no que algunos como es el caso del lenguaje KIF, Knowledge Interchange Format, surge para intercambiar conocimiento entre sistemas de computación. Éste es un lenguaje intermedio que fue desarrollado en 1992 por el grupo de trabajo Interlingua de la universidad de Standford con el objetivo de resolver el problema de la heterogeneidad en los lenguajes de representación del conocimiento.

Otros lenguajes, conocidos como lenguajes tradicionales de ontologías, surgen en una etapa en la que sus creadores, para representar el conocimiento, se basan unos en la lógica de predicados, ofreciendo poderosas primitivas de modelado, mientras que otros lo hacen en frames, es decir, con taxonomías de clases y atributos, que por tanto ofrecen una mayor capacidad expresiva, aunque menor poder de inferencia. Dentro de esta etapa, cabe destacar el lenguaje Ontolingua, desarrollado durante el proyecto Knowledge Sharing Effort del departamento de defensa de los Estados Unidos. Ontolingua se basa en KIF pero para permitir la descripción de ontologías utiliza los paradigmas de frames, surgiendo con su creación términos como clase, instancia o subclase. También son destacables los lenguajes Flogic, que integra marcos y calcula predicados de primer orden o el OCML ( Operational Conceptual Modeling Language), que se basa en marcos y al cual se le añadió un módulo de mecanismos

de lógica, así como una interfaz para poder interactuar con el conocimiento. Además este último es compatible con estándares como Ontolingua.

**Figura 6. Lenguajes de ontologías tradicionales.**



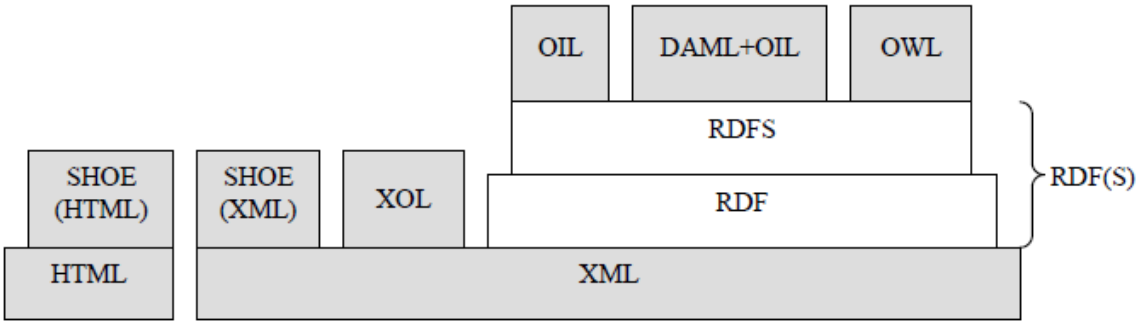

Fuente: Sistema Multi-Agente para la construcción semiautomática de Ontologías a partir de la Web.

Posteriormente, y debido a la consagración de internet, surgió una nueva etapa en la que se crearon lenguajes específicos para el desarrollo de ontologías y para poder ser utilizados en la web. Todos ellos se basan en el lenguaje XML, que es un metalenguaje que permite definir y validar los lenguajes de etiquetado que se usan en la web. A continuación hablaremos brevemente de los que se consideran más destacados.

El lenguaje **SHOE** fue el primer lenguaje para diseñar ontologías en la web. En este proyecto las ontologías y las etiquetas se incrustaban en archivos HTML. Se trata de un lenguaje que permite definir clases y reglas de inferencia, pero no es posible definir negaciones o disyunciones. Este proyecto fue abandonado, aunque también existe una serialización de este lenguaje en XML.

Por otro lado, el **RDF** (Resource Description Framework), es un lenguaje que está siendo desarrollado por el World Wide Web Consortiun (W3C) y que se utiliza para modelar datos. Se trata de un lenguaje abstracto que fue creado por la necesidad de poseer una mayor expresividad en el procesamiento semántico que la ofrecida hasta entonces por lenguajes como el XML, que sólo proporcionaba una estructura sintáctica. Mediante recursos, propiedades y declaraciones, permite una representación explícita de la semántica de los datos. Los recursos (sujetos) son todos aquellos de lo que se puede decir algo y son referenciados mediante un identificador único de recursos, URI. Las propiedades (predicados) son atributos o relaciones que se usan para definir un recurso. Por último, las declaraciones (objeto) sirven para dar valores a las propiedades de un recurso específico. Los tres, constituyen una tripleta sujeto-predicado-objeto que es la construcción básica en RDF.

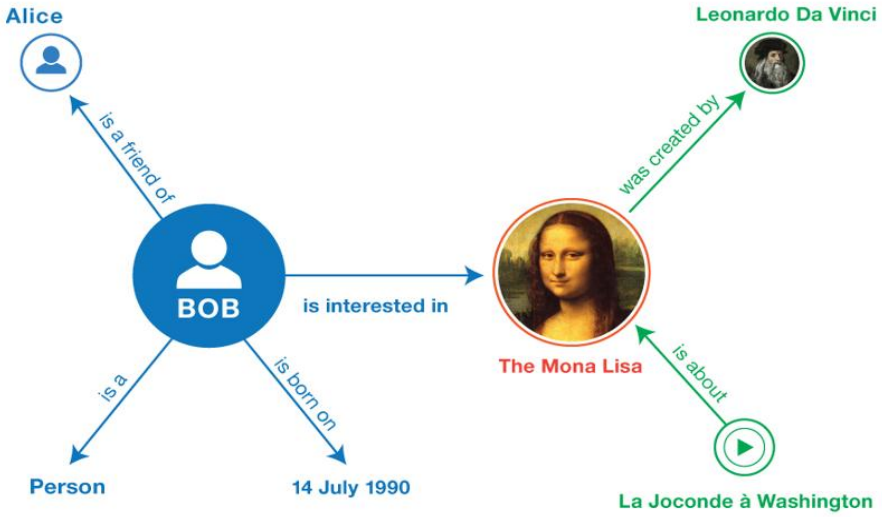
**Figura 7. Lenguajes de ontologías basados en Web.**




Fuente: Sistema Multi-Agente para la construcción semiautomática de Ontologías a partir de la Web.

Sin embargo, este lenguaje también presenta algunos inconvenientes, como que no pueden expresarse restricciones de cardinalidad o que algunas expresiones no pueden declararse mediante la lógica de primer orden, causando por tanto sentencias indecidibles de las que no se puede afirmar ni negar nada.

**Figura 8. Ejemplo de tripleta RDF.**




Fuente: W3C (http http://www.w3.org/TR/rdf11-primer/)

El lenguaje **OIL** (Ontology Interchange Language), se desarrolló como parte del proyecto OntoKnowledge y utiliza la sintaxis del lenguaje XML. Está definido como una extensión RDFS y trata de conseguir la interoperabilidad semántica entre recursos web. Posee varias capas de sub-lenguajes, cada una de las capas subsiguientes añade alguna funcionalidad y mayor complejidad. Su principal carencia es la falta de expresividad para declarar axiomas.

A parte, se encuentra el lenguaje **DAML +OIL**. El programa DAML, DARPA Agent Markup Language, fue en 1999 una iniciativa del Defense Advance Research Projects Agency con el objetivo de proveer los fundamentos de la Web semántica, al igual que el OIL.

La principal diferencia de éste con el lenguaje OIL es que se trata de un lenguaje que se aleja de los ideales de frames para acercase más a una base de lenguajes de lógica descriptiva. Se trata de un lenguaje que tiene una semántica sencilla y que en su última revisión, ofrece ya un rico conjunto de elementos con los que se pueden crear ontologías y marcar la información para que sea legible y comprensible por una máquina.

Por último y no por ello menos importante, se va a hablar del lenguaje **OWL**, pero como se trata del lenguaje elegido en la realización del proyecto, se le dedicará un apartado especial.

#### 1.2.4.1. Lenguaje utilizado: OWL.

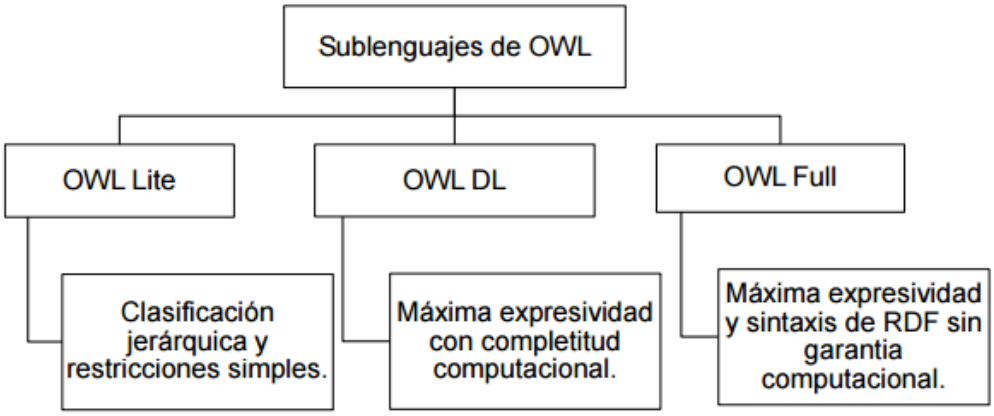
El lenguaje OWL (Ontology Web Language), está basado en el RDF pero posee una mayor capacidad expresiva y surge como revisión al lenguaje DAML-OIL para que sirva como estándar para los investigadores de la Web semántica. El OWL es el resultado del trabajo del Web Ontology Working Group (W3C), que se formó en noviembre de 2001. Uno de sus principales usos es cuando la información almacenada en los documentos necesita ser procesada o bien por programas o bien por aplicaciones.

Este lenguaje también utiliza el modelo de tripletas de RDF, con un mayor poder expresivo, puesto que se puede utilizar tanto para representación de los términos como de las relaciones entre los mismos, fundamental a la hora de realizar una ontología. Además incluye propiedades como restricciones entre clases, inclusiones o equivalencia, así como alguna funcionalidad para el razonamiento que amplían el lenguaje RDF y sobre todo, sus usos. También permite expresar la cardinalidad, es decir, el número de elementos que puede componer una clase, o un objeto.

Este lenguaje además permite la interoperabilidad y posee propiedades que facilitan la importación y exportación de clases, algo que resulta sumamente útil cuando se integran o se mezclan diversas ontologías. Además las ontologías que se crean con OWL son documentos web y por tanto, se referencian gracias a un identificador de recursos uniforme (URI), aspecto fundamental para este proyecto, puesto que esto permite la distribución de la ontología que se desarrolle.

OWL como se puede apreciar en la figura anterior se separó en tres versiones distintas en función de su expresividad semántica, siendo éstas OWL Lite, la versión más simple que extiende RDF y reúne las características más comunes de OWL, por lo que está destinada a quienes quieren crear clases y utilizar las relaciones más sencillas; OWL DL, que incluye el vocabulario completo de OWL  y por último OWL Full que incluye todo el vocabulario y carece de limitaciones para explorar su potencial.

**Figura 9. Sublenguajes de OWL.**




Por tanto, dado que se ha visto el poder expresivo y la interoperabilidad que ofrece este lenguaje es suficiente para la creación de la ontología que pretende este proyecto, este ha sido el lenguaje seleccionado.

### 1.2.5. Herramientas para ontologías

Se pueden encontrar diversas herramientas para el tratamiento de las ontologías, por un lado están las herramientas de desarrollo de ontologías, que son aquellas que sirven o bien para la creación de nuevas ontologías o bien para la reutilización de aquellas que ya existen. Por otro lado, las herramientas de fusión e integración de ontologías, útiles cuando se pretende obtener una ontología a partir de las ya existentes. En este proyecto se abordará la descripción de las primeras, puesto que se quiere crear una ontología y a partir de la misma, establecer un aprendizaje automático.

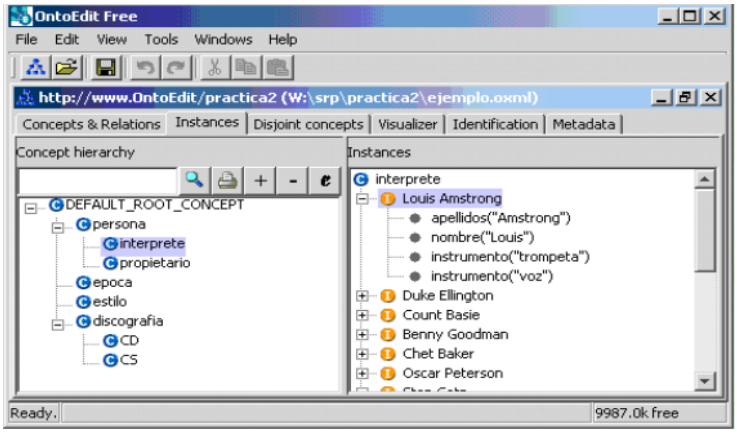
#### 1.2.5.1. Evolución de las herramientas para la creación de ontologías.

A la hora de construir una ontología es necesario poseer un editor que sea capaz de soportar la definición de los conceptos, relaciones y atributos que se quieren crear. Algunos de estos editores se han creado específicamente para un determinado lenguaje, como es el caso de Ontolingua Server que fue creado en 1990 en la Universidad de Stanford y que está orientado al lenguaje Ontolingua, aunque después se habilitó para otra serie de lenguajes. Otras sin embargo, son capaces de soportar diversos lenguajes.

Una herramienta importante es el OilEd, que fue desarrollado por la Universidad de Manchester. Se trata de un editor basado en Java y que ofrece la posibilidad de crear ontologías usando OIL o lenguajes de alta expresividad como puede ser el lenguaje DAML+OIL. Esta herramienta, a pesar de ser de libre acceso, no resulta de utilidad en este proyecto puesto que carece de la capacidad de migración o integración de ontologías.

Por otro lado se encuentra la herramienta OntoEdit, que se trata de una herramienta gráfica para el desarrollo de las ontologías en RDF y en DAML+OIL y para el mantenimiento de las mismas. Este editor permite crear ontologías compuestas de términos que representan los conceptos de un dominio determinado; propiedades, que se asocian a los diferentes términos y relaciones entre términos.

**Figura 10. Entorno de trabajo de OntoEdit**




Fuente: UPV

También existe una infraestructura de gestión ontológica específica para aplicaciones de negocio que se denomina Kaon y fue desarrollada por la universidad de Karlsruhe.

El NeOn toolkit es una plataforma desarrollada por el ECFundedNeon Project basada en el editor de ontologías OntoStudio. Trabaja con los lenguajes F-logic y OWL. Es una herramienta de modelado para la creación y mantenimiento de ontologías compuesto por un navegador ontológico, un panel de individuos ( donde muestra todas las instancias) y un panel de propiedades y entidades.

Por último, es importante destacar el WebODE, una plataforma de desarrollo de ontologías que se empezó a desarrollar en 1999 en la Universidad Politécnica de Madrid dentro del grupo de ingeniería Ontológica de esta misma universidad. Se construyó para la creación de ontologías y permite la interoperación con otros sistemas de información, basádose en los formularios HTML y applets Java. Este editor ofrece servicios como la edición de ontologías, la mezcla, el razonamiento, y la importación y exportación de lenguajes de ontologías para generar documentación, entre otros. En el editor se combinan tres interfaces de usuario, la primera permite editar los términos de una ontología, la segunda permite editar axiomas y reglas, y la última es una interfaz gráfica que permite editar taxonomías de conceptos y relaciones. Sin embargo, desde 2006 ya no está siendo utilizado ni mantenido por el Ontology Engineering Group, por lo que su uso para este proyecto fue descartado.

En función al lenguaje que se requería para la ontología y de las características que se requerían para la creación de la misma, el editor que se ha seleccionado se muestra a continuación.

#### 1.2.5.2. Herramienta utilizada: Protegé

Protegé es una herramienta gratuita creada en la Universidad de Stanford y probablemente es la herramienta de construcción de ontologías más utilizada. Se trata de una herramienta de software integrado, de código abierto, que permite la creación de sistemas basados en el conocimiento. Sus primeras versiones datan de 1998.

Con esta herramienta se puede crear una ontología, insertar datos en la misma y hasta optimizar los datos de entrada. Además del editor posee una biblioteca de extensiones que permite aumentar sus funcionalidades. Contiene un conjunto de estructuras para el modelado del conocimiento y diversos formatos para la representación y visualización de los contenidos. Con este editor es posible manejar ontologías con más de 100.000 conceptos.

Además, su código Java es libre y ofrece el acceso a la API utilizada en su desarrollo con la que poder crear aplicaciones Java que manipulen ontologías. Esto es algo fundamental de cara a la aplicación del diseño de bases de conocimiento para agentes artificiales a través de su representación mediante ontologías.

Una de las utilidades de esta herramienta que se puede destacar es que permite la visualización de las ontologías, lo que facilita la interacción con los seres humanos, ya sea en las labores de inspección o en la comprensión por parte de personas que no estén familiarizadas con este tema.

Este editor ofrece la posibilidad de tener dos formas de modelar el conocimiento, el editor Protegé-Frames, que permite la construcción de ontologías basadas en marcos, o el editor Protegé-OWL que permite construir ontologías para la web semántica en lenguaje OWL y que es la funcionalidad escogida para la realización del proyecto.

Protegé-OWL permite abrir y guardar las ontologías creadas en ficheros con extensión *.owl o bien desde una dirección de internet. Este tipo de ontologías también posee clases, propiedades e instancias y facilita la inferencia de conocimiento deducible a través de las relaciones semánticas. Se ha seleccionado este entorno ya que el poder crear un archivo *.owl, facilita el que pueda ser utilizado por diversos sistemas y porque, como se verá más adelante en la configuración del entorno, existe una librería Java que nos permite la lectura y edición de este tipo de archivos a través del programa Eclipse, necesario para crear el sistema de aprendizaje automático.

Por todos los motivos y características señalados, Protégé es la herramienta seleccionada para permitir la interacción humana con la base de conocimiento que se quiere crear en este proyecto, y que además permitirá la creación de los ficheros en los cuales se almacenará la información de la misma.

### 1.2.6. Aplicaciones de las ontologías.

Como se ha comentado anteriormente, el uso de las ontologías posee grandes ventajas, puesto que proporciona un vocabulario común para representar y compartir el conocimiento, unifica la forma de intercambio de dicho conocimiento, permite la reutilización del mismo y proporciona un protocolo de comunicación. La aplicabilidad de las ontologías con el paso del tiempo es cada vez mayor, no sólo estando presente en la filosofía, la biblioteconomía y la documentación, sino que ha sufrido un gran impulso con el desarrollo de la web semántica, para transformar la red en un espacio de conocimiento, además de estar presente en el campo de la inteligencia artificial.

Un ejemplo de uso de las ontologías puede ser el de la Web semántica o en los sistemas de gestión empresarial, puesto que permiten que las aplicaciones estén de acuerdo en los términos que usan cuando se comunican. Mediante las ontologías es más sencillo recuperar información relacionada temáticamente en la web, aun cuando no existan enlaces

directos entre las páginas web. Además, a través de las ontologías se favorece la gestión de contenidos, la integración de la cadena de suministro y de la cadena de valor, así como la estandarización de la información de los mercados electrónicos (e-marketplaces).

También se utilizan para otros propósitos como puede ser el procesamiento del lenguaje natural o en la integración inteligente de los sistemas, estando presente en ámbitos como la ingeniería del conocimiento o la ingeniería de bases de datos o software.

En el campo de la inteligencia artificial, las ontologías son muy útiles para facilitar el aprendizaje automático, es decir, sin intervención humana. Hace posible dotar a los robots de cierta autonomía para realizar las tareas que se espera de los mismos sin una continua orientación humana. Esta autonomía tendrá un impacto considerable en cómo los robots interactúan con los seres humanos y es la aplicación de la ontología que se va a realizar en este proyecto.

## 1.3. ROBOTS SOCIALES.

Como este proyecto trata sobre el aprendizaje en los robots sociales, es necesario realizar una breve mención sobre la historia de la robótica y sobre qué es un robot social, así como de los antecedentes a este trabajo.

### 1.3.1. Historia de la robótica.

Si nos remontamos a lo largo de la historia, los seres humanos siempre han tratado de imitar las funciones y movimientos de los seres vivos mediante máquinas, a las cuales se denominó autómatas. Los primeros mecanismos animados que se conocen datan del 270 a.C. y se le atribuyen a Ctesibius, quien aplicó sus conocimientos de ingeniería para producir los primeros relojes de agua y órganos con figuras en movimiento. Posteriormente, surgieron los manipuladores teleoperados. Sin embargo, a pesar de los numerosos autómatas que se desarrollaron a lo largo de los siglos, no es hasta 1921 cuando se introduce por primera vez el término robot. Fue utilizado por el escritor checo Karel Capek en el estreno de su obra Rossum´s Universal Robot en el teatro nacional de Praga. El origen del término robot es la palabra eslava robota que se refiere al trabajo realizado de manera forzada. Los robots en esta obra de teatro eran máquinas androides fabricadas por el científico Rossum que servían a sus jefes humanos desarrollando todos los trabajos físicos, hasta que llegado el momento se revelaban contra ellos.

Posteriormente, fue un término muy utilizado en la literatura, pero es el escritor americano Isaac Asimov el máximo impulsor de esta palabra, además del creador de la palabra robotics (robótica), utilizada por primera vez en su obra Runaround en 1942. Asimov publicó en 1942 en la revista Galaxy Science Fiction una historia titulada <<The Caves of Steel>> en las que enunció por primera vez sus tres leyes de la robótica:

1. Un robot no puede perjudicar a un ser humano, ni con su inanición permitir que un ser humano sufra daño.

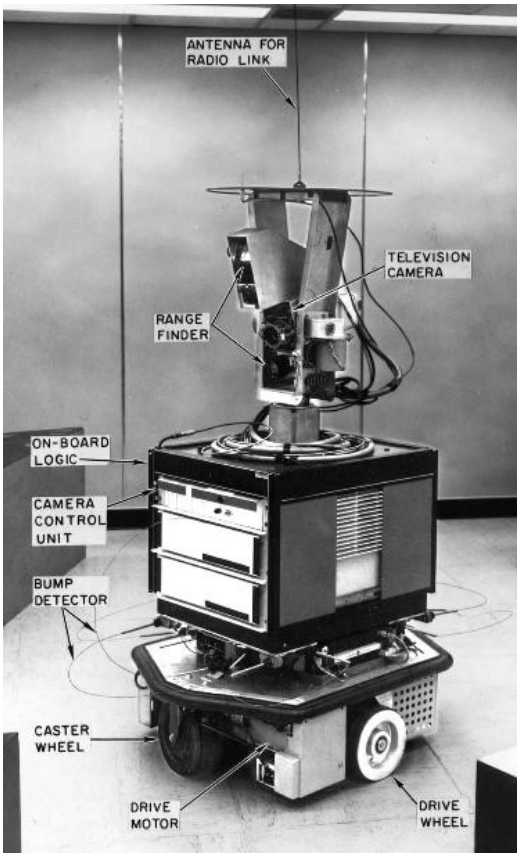
2. Un robot ha de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entran en conflicto con la primera ley.

3. Un robot debe proteger su propia existencia mientras tal protección no entre en conflicto con la primera o segunda ley.

En los años 50 y 60 empiezan a aparecer las primeras patentes de artilugios robóticos, y ya en 1972 surge en Japón la primera asociación robótica del mundo, aventajando así a Estados Unidos y por supuesto a Europa, algo más tardía en este campo. Es durante estas décadas cuando empiezan a sentarse las bases de la investigación acerca de este campo en diversas universidades y se desarrollan los primeros robots móviles, los cuales poseían cierto grado de autonomía. El primero de este tipo de robots fue Shakey, el cual se muestra a continuación y que fue desarrollado por el Stanford Research Institute.

En esta época empiezan a desarrollarse también brazos manipuladores y poco a poco a introducirse los robots en otros campos además de la investigación, como en el espacial o el submarino.

**Figura 11. Robot Shakey.**




Fuente: Robot, Moravec, Oxford 1998. Capítulo 2.

Durante los 40 últimos años, la robótica ha evolucionado a pasos agigantados, creándose robots humanoides y robots de entretenimiento, que son capaces de realizar funciones como tocar un piano, caminar, subir escaleras, poseer una cierta capacidad de aprendizaje o tener un comportamiento autónomo. Además, se han introducido robots en multitud de hogares para realizar tareas domésticas como el robot aspiradora Roomba.

Actualmente, podemos encontrar robots en casi cualquier área de producción y en cualquier tipo de industria, como sustituto de los humanos en aquellas tareas repetitivas y hostiles, así como en aquellas actividades de riesgo para las personas. El desarrollo de la robótica, se centra de cara al futuro en el aumento de la movilidad, destreza y autonomía de sus acciones. Además cada vez se busca una mayor interacción con los seres humanos, con elementos que simplifiquen su vida, como es el caso de las sillas robóticas para las personas discapacitadas, o simplemente para tareas como la educación o el acompañamiento a las personas mediante los robots sociales, en los cuales esta interacción se manifiesta a través de la voz, el estudio de las expresiones faciales, el lenguaje o los gestos y movimientos.

Como se ha podido comprobar, el campo de la robótica avanza rápidamente y por tanto es necesario ir revisando y ampliando el concepto de robot frecuentemente. Por ello, se suele añadir algún adjetivo a la palabra robot, de modo que así se haga referencia a un campo más reducido y sea posibles especificar más sus características o su aplicación. La primera definición que se estableció de robot se le atribuye al Robot Institute of America en 1979 según la cual, un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables,

programadas para realizar tareas diversas. Esta definición ha sido la base para las sucesivas definiciones y especificaciones de robot que han ido surgiendo a lo largo de estas décadas.

En lo que a este proyecto respecta, no resulta de interés establecer las diferentes definiciones ni clasificaciones de robots que existen en la actualidad, puesto que se centrará en un tipo de robot específico, los robots sociales.

### 1.3.2. Robots sociales.

Un robot social es un robot autónomo que interactúa y se comunica con las personas, de una manera sencilla y agradable siguiendo una serie de comportamientos y normas sociales. Los robots autónomos perciben su mundo, son capaces de tomar sus propias decisiones y realizar acciones para desarrollar sus tareas. Para que esta interacción robot-humano sea posible, es necesario que además de poseer sistemas perceptivos robustos, posea una serie de habilidades que se agrupan dentro de lo que se conoce como inteligencia social. Para esta interacción social, el robot debe poseer un modelo cognitivo-afectivo, es decir, capacidades como la comunicación, la comprensión y el aprendizaje.

Como podemos ver, se consideran sólo robots sociales aquellos en los que la interacción persona-máquina adquiera un nivel relevante. Actualmente, este tipo de robot se encuentra aún en fase de investigación, aunque empiezan a aparecer este tipo de robots en determinadas circunstancias, como pueden ser los robots para la rehabilitación o acompañamiento de enfermos o los robots guía en los museos. Dentro de esta última funcionalidad encontramos a Urbano, el robot social desarrollado por el grupo de investigación de Control Inteligente de la Escuela Técnica Superior de Ingenieros Industriales, y para el cual fue pensado este proyecto.

**Figura 12. Robot Urbano.**


Los robots sociales, como establece Breazeal en uno de sus artículos, se pueden dividir en cuatro grupos atendiendo a la complejidad del escenario en que se produce la interacción,

siendo éstos socialmente evocativos, robots de interfaz social, socialmente receptivos y sociables.

Los robots socialmente evocativos son aquellos que se antropomorfizan para animar a las personas a interactuar con la tecnología, como pueden ser las "mascotas" robóticas o en los videojuegos en que los participantes ingenian criaturas animadas para interactuar con ellas. Es decir, cuando el humano atribuye la capacidad de respuesta social al robot, pero su comportamiento no se corresponde con la realidad.

Los robots de interfaz social sin embargo utilizan un modo de comunicación de tipo humano para facilitar la interacción con las personas. Este es el caso de los robots con cierta inteligencia social que se ha comentado anteriormente para poder transmitir los mensajes de manera adecuada. Para ello se apoya en gestos, expresiones, etc.

Los robots receptivos, al igual que los anteriores también se benefician de las interacciones con las personas. Entrarían dentro de este tipo los robots que aprenden a través del entrenamiento ya sea del movimiento o del habla. Son aquellos que las personas pueden modelar el comportamiento del robot a través de gestos o etiquetas simbólicas. Este tipo de robots son socialmente pasivos, es decir, responden a los esfuerzos que las personas realizan para interactuar con ellos pero no comprometen activamente a las personas para satisfacer objetivos sociales internos.

Por último, los robots sociables son aquellos con sus propias metas y motivaciones internas. Tratan de involucrar a las personas de una manera social no sólo para el beneficio de las mismas, sino para beneficiarse ellos mismos, por ejemplo para aprender. Estos robots tratan por tanto de modelar a las personas en términos sociales y cognitivos para interactuar con ellos.

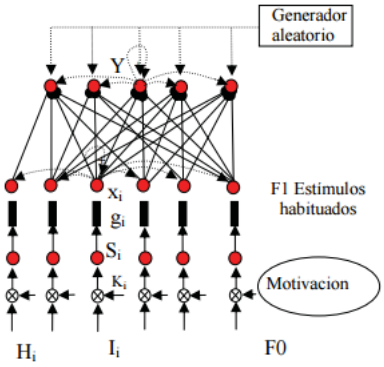
Este proyecto, al intentar desarrollar un aprendizaje en los robots sociales, está tratando también de convertirlos un poco más en **robots sociables**.

### 1.3.3. Antecedentes de robots con conocimiento.

El aprendizaje es uno de los grandes retos dentro de la robótica social y uno de los más complejos, debido a la dificultad de representar el conocimiento y relacionarlo con la experiencia.

Se encuentra el caso de Arisco, un robot social en el que el aprendizaje que emplea realiza asociaciones estímulo-respuesta, utilizando un modelo de red neuronal como el que se muestra en la figura. En este tipo de aprendizaje se relacionan dos estímulos, uno de ellos asociado a una determinada respuesta (estímulo condicionado). Tras producirse este estímulo próximo a otro en el tiempo, el individuo termina por asociar el estímulo independiente con la respuesta. El modelo básico de memoria a corto plazo que utiliza se basa en el modelo multiplicativo de Grossberg que a su vez se basa en la ecuación de Hodgking-Huxley que describe el comportamiento de las membranas de las neuronas. Por otro lado, la ecuación que modela el aprendizaje a largo plazo es la ecuación de aprendizaje outstar enunciada por Grossberg en 1980.

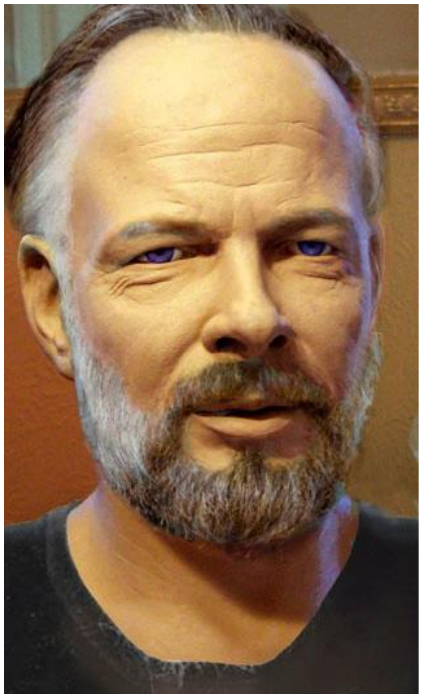
**Figura 13. Arquitectura neuronal de Arisco.**




Fuente: "Arisco" un Robot Social con Capacidad de Interacción, Motivación y Aprendizaje

Otro ejemplo es el androide Dick, que ha recibido diversos premios por su avance en conversaciones robóticas y en la interacción robot-humano. Su primera versión fue construida en 2005 por Hanson Robotics. Este androide tiene en su software el trabajo de distintos autores, principalmente el de Philip K.Dick, de forma que si se le pregunta algo que no está almacenado en su sistema, es capaz de generar una respuesta similar a lo que diría este autor gracias a lo que su creador llama un análisis semántico latente. Este robot es capaz de incorporar en tiempo real palabras nuevas que va aprendiendo en cada conversación y modifica su tono según el ánimo de su interlocutor gracias a su software de reconocimiento facial.

**Figura 14. Imagen del androide Dick.**



Fuente: Hanson Robotics

Aunque no dentro del ámbito de los robots sociales, también podemos destacar a Amelia, un agente cognitivo desarrollado por IPsoft. Es un sistema basado en inteligencia

artificial que habla más de 20 idiomas y es capaz de leer y entender textos, resolver problemas y aprender de la experiencia. Según sus creadores, es capaz de entender el contexto, aplicar la lógica y deducir implicaciones. Está enfocado a la automatización de los procesos dentro de un negocio.

Por último, para referirnos a algo más similar a lo que se realiza en este proyecto debemos citar el proyecto YAGO. Se trata de una base de conocimiento derivada de Wikipedia, WordNet y GeoNames desarrollada conjuntamente por el Max Plank Institute for Informatics y la universidad de telecomunicaciones ParisTech. YAGO tiene conocimiento de más de 10 millones de entidades tales como personas, ciudades, organizaciones, etc. y más de 120 millones de hechos sobre dichas entidades. Se ha demostrado una precisión de un 95% y cada relación va acompañada de un factor de confianza. Además se trata de una ontología anclada en el espacio y el tiempo, puesto que atribuye una dimensión temporal y espacial a muchos de sus hechos y entidades.

### 1.3.4. Trabajos anteriores.

El Grupo de Control Inteligente de la Universidad Politécnica de Madrid viene trabajando en el tratamiento de esta parte de la inteligencia artificial desde hace tiempo. El primer trabajo se desarrolló en el año 2008 como un proyecto fin de carrera por Carlos Florit. En este trabajo se pretendía demostrar la viabilidad del uso de ontologías informáticas por robots sociales. Además se estudió la posibilidad de utilizar Protégé para representar el conocimiento.

Más adelante, en 2013 en el Simposio de Control Inteligente celebrado en Tenerife, se presentó la integración del uso de una ontología informática en el diálogo del robot Urbano con el usuario. Es decir, se estudió la posibilidad de consultar una ontología desde otro programa. Estos trabajos demostraban por tanto la viabilidad y el uso de estas ontologías, pero la definición del conocimiento tenía que ser realizada por el usuario.

# 2. OBJETIVOS

Después de haber contextualizado el proyecto es el momento de definir los objetivos que se han establecido para su realización.

En este proyecto se va a crear un programa que permita realizar un aprendizaje de manera automática para poder crear así una ontología que almacene el conocimiento. Es importante destacar que con este proyecto se pretende resolver el problema de la memoria a medio plazo, permitiendo la búsqueda de contenidos en la ontología y fundamentalmente la memoria a largo plazo, puesto que dota al agente de la capacidad de aprender de una forma autónoma y almacenar los contenidos en una base de conocimiento para que pueda usarlos cuando necesite.

Este trabajo constará por tanto de un programa principal, que se realizará en lenguaje JAVA y en la plataforma software Eclipse. Este programa constará de una interfaz gráfica que permitirá la comunicación con el programa, de modo que el usuario pueda visualizar por pantalla la evolución del agente. Además, en Eclipse se instalará una interfaz de programación de aplicaciones para poder realizar la gestión de las ontologías, que se crearán gracias al editor de ontologías Protégé, y que ha sido elegido atendiendo a las razones expresadas en la introducción. Desde Eclipse se realizará tanto la lectura de la ontología como la creación de nuevas clases, instancias o relaciones. Más adelante se explicará con más detenimiento cómo se debe llevar a cabo la integración de los programas, además de las librerías necesarias.

Además, en este programa principal se desarrollará un bucle de aprendizaje, de manera que el agente pueda modificar su base de conocimiento, la ontología, según la información que vaya obteniendo. En este caso, la obtención de la información será posible o bien mediante el procesamiento de un archivo de texto, o bien mediante búsquedas que el agente podrá realizar en la enciclopedia Wikipedia, dotando al agente gracias a esta última funcionalidad de un aprendizaje automático. Se pretende que este aprendizaje automático sea para ampliar algún conocimiento sobre algo que ya se haya almacenado en la ontología, o para añadir conocimiento nuevo mediante una búsqueda de contenido aleatoria en la citada enciclopedia.

También se incluirá una función que será la que permita realizar una comunicación TCP entre el programa y el robot Urbano, en quien se pretende instalar esta aplicación.

Para asegurar la fiabilidad del conocimiento que se almacene, se creará un factor de confianza que acompañará a los términos que integren la ontología de forma que si los conceptos aparecen en más ocasiones se reforzará la veracidad de estos contenidos.

En última instancia, lo que se obtendrá será un fichero con la extensión *.owl que contendrá la base de conocimiento y que será posible distribuir para integrar en robots sociales o en otros agentes.

# 3. METODOLOGÍA

Como se ha mencionado en el capítulo anterior, el objetivo es crear un programa de aprendizaje automático. Para su desarrollo son necesarias varias herramientas. A continuación, se van a ver las diversas herramientas que se utilizarán y cómo se lleva a cabo la integración de todas ellas para obtener un programa que se adapte a las necesidades del proyecto.

## 3.1. HERRAMIENTAS UTILIZADAS.

Para desarrollar el programa principal de este proyecto, se ha utilizado el programa Eclipse, un software de código abierto desarrollado inicialmente por IBM que se utiliza para crear entornos de desarrollo integrado para casi cualquier lenguaje de programación, siendo más utilizado con los lenguajes Java y C/C++.

En el contexto de este proyecto, esta plataforma, concretamente la versión de Eclipse Mars, ha sido seleccionada para crear el programa desde el cual se gestiona todo. A través de este programa se creará una interfaz de usuario con la ayuda de la aplicación WindowBuilder que se verá más adelante como obtenerla. Esta interfaz de usuario es la que permitirá manejar de una forma más intuitiva todo el proceso de aprendizaje.

Además, se ha seleccionado este entorno de programación porque Protégé, el editor de ontologías que se ha comentado en el primer capítulo, posee una interfaz de programación de aplicaciones (API), es decir, un conjunto de subrutinas, funciones y métodos para poder ser utilizado a través de Eclipse. Por lo tanto, desde este código que se desarrolla en Java se pueden gestionar la lectura, edición y la población de la ontología que se va a crear. En otras palabras, se crearán una serie de clases y funciones que manejarán la ontología a través de la API Protégé-OWL. A continuación se describirá cómo se debe llevar a cabo la integración de estos dos programas.

Para que esto sea posible, es necesario también instalar en Eclipse la librería Jena, un framework desarrollado en lenguaje Java para la construcción de aplicaciones que funcionan con las tecnologías de la web semántica, aportando un entorno de programación para el desarrollo de RDF, OWL y RDF Schema.

Por otro lado, también en Eclipse se crearán las funciones necesarias para realizar las búsquedas de conocimiento en la página Web de Wikipedia, más concretamente de los Wikidatas. Wikidata es una base de conocimiento libre y abierta que puede ser leída y editada tanto por seres humanos como por máquinas. Actúa por lo tanto como almacenamiento central para los datos estructurados de sus proyectos hermanos de Wikimedia, en el que se incluye Wikipedia. Presenta un aspecto como el mostrado en la figura.

**Figura 15. Elemento Wikidata del término Robot.**

| Language | Label | Description | Also known as |
|---|---|---|---|
| English | robot | mechanical or virtual artificial agent | bot |
| Spanish | robot | entidad virtual o mecánica artificial | |

Fuente: https://www.wikidata.org/wiki/Q11012

Igualmente estas oraciones que se obtienen de internet se tratarán con un analizador sintáctico, en este caso es Freeling el que se va a utilizar. Freeling es una librería de código abierto que ofrece herramientas para el análisis lingüístico (análisis morfológico, análisis sintáctico, etiquetado de roles semánticos, desambigüación de palabras..). Fue desarrollado por el grupo de investigación de procesamiento de lenguaje natural de la Universidad Politécnica de Cataluña. A continuación se muestra una imagen de la demo que ofrecen en su página web y acto seguido otra imagen sobre la salida que se obtendría del análisis de la oración que se introduzca.

**Figura 16. Analizador sintáctico Freeling.**

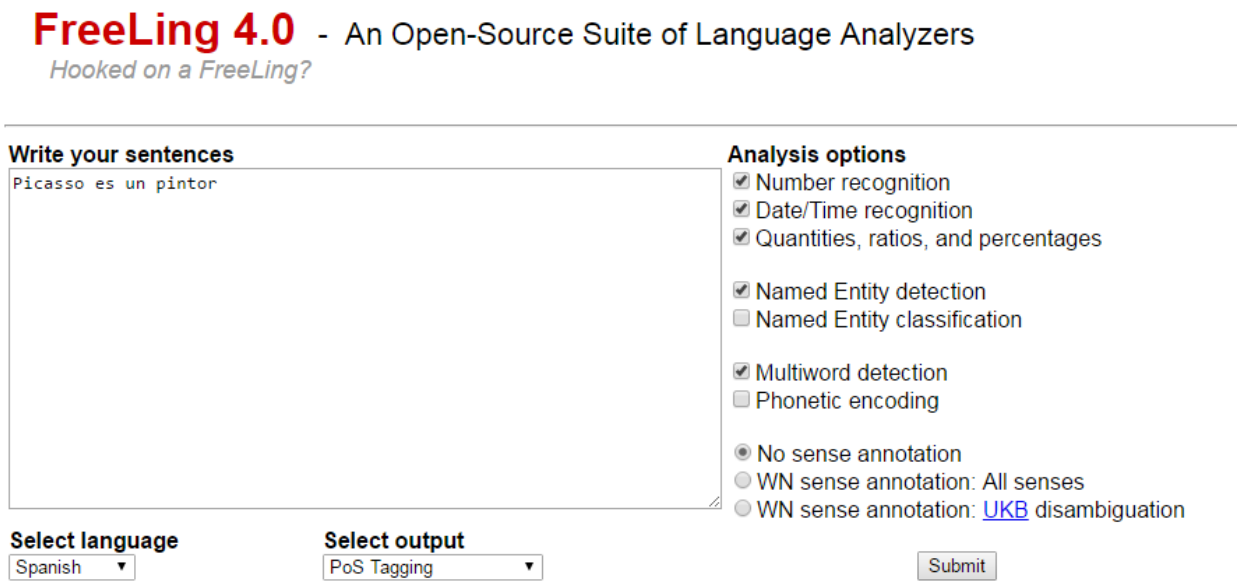
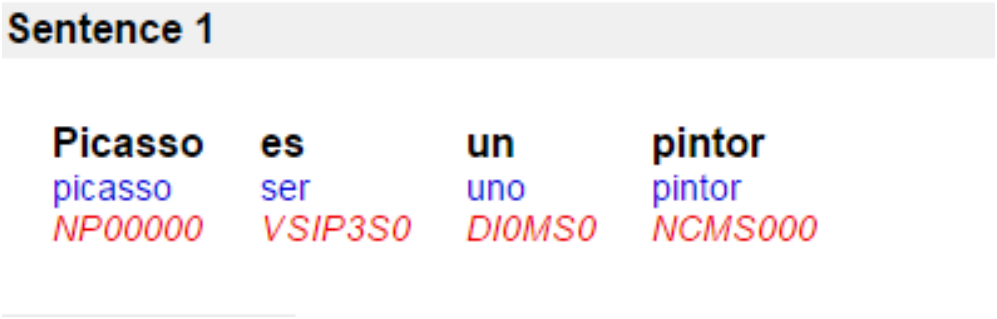



**Figura 17. Salida del analizador sintáctico.**




Fuente: nlp.lsi.upc.edu/freeling/demo

Estas son las herramientas que se utilizan en el proyecto, a continuación se va a ver cómo integrar todos estos elementos en un único programa y posteriormente los resultados obtenidos en la realización del trabajo.

## 3.2. PREPARACIÓN DEL ENTORNO DE TRABAJO.

Lo primero que se necesita, como ya se ha mencionado, es realizar la instalación del programa Eclipse, que se puede obtener directamente en su página web. En el caso de este proyecto, se ha realizado en la versión de Eclipse Mars, puesto que ya se contaba con la instalación del mismo.

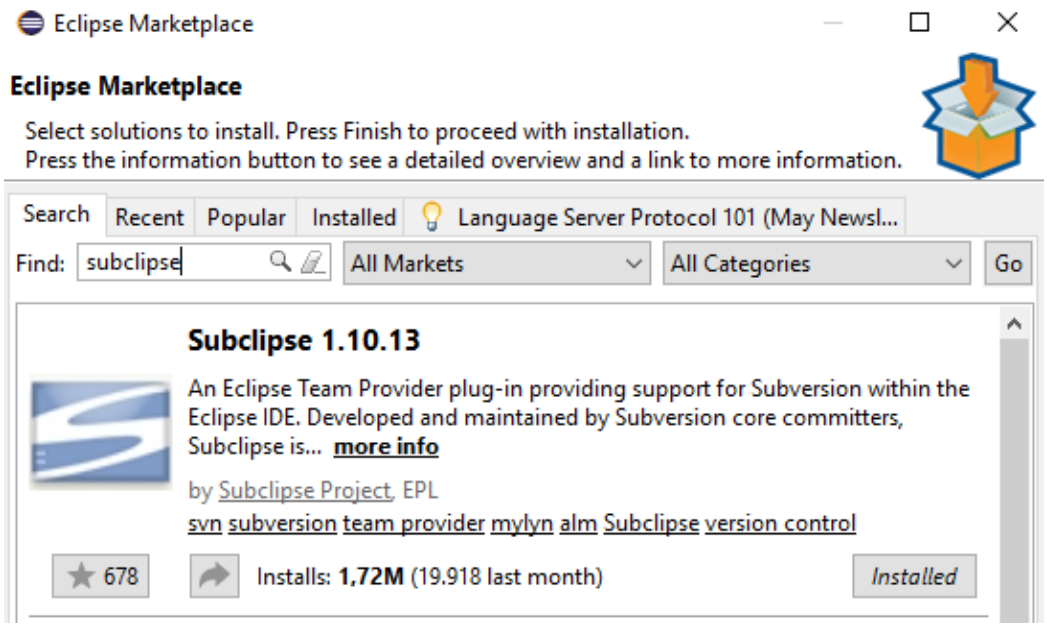
Una vez instalado Eclipse, para poder trabajar con las ontologías se debe instalar el API Protégé-OWL. Se trata de un proceso algo costoso porque cuesta encontrar información sobre ello, por lo que aquí se va a realizar un resumen de cómo llevar a cabo la instalación.

Es necesario instalar dos grupos de archivos. El primero es el Protégé-Core para el entorno de desarrollo de Eclipse, que es el que permite ejecutar Protégé desde Eclipse. Hay varias formas de llevar a cabo su instalación, describiéndose a continuación la que se ha considerado más sencilla, pero teniendo en cuenta que esto es válido para la versión 3.x de Protégé y no es aplicable para la versión 4.0. El segundo grupo de archivos necesario es "Protégé-OWL", que es el que permite la interacción entre Eclipse y Protégé para la generación de la ontología y que posee la misma limitación de versiones que el anterior. A continuación se detalla la integración de ambos.

### 3.2.1. Instalación de Protégé-Core.

Lo primero que hay que hacer es abrir Eclipse e ir al repositorio SVN, una herramienta que posee para el control de versiones de código abierto basado en un repositorio cuyo funcionamiento se asemeja a un sistema de ficheros. Para ello, es necesario tener instalado el complemento Subclipse que añade la subversión al entorno de desarrollo integrado de eclipse. Lo podemos instalar desde el Marketplace de Eclipse, al que se accede desde la pestaña de ayuda, e introduciendo en el buscador el nombre del complemento, tal y como se muestra en la figura.

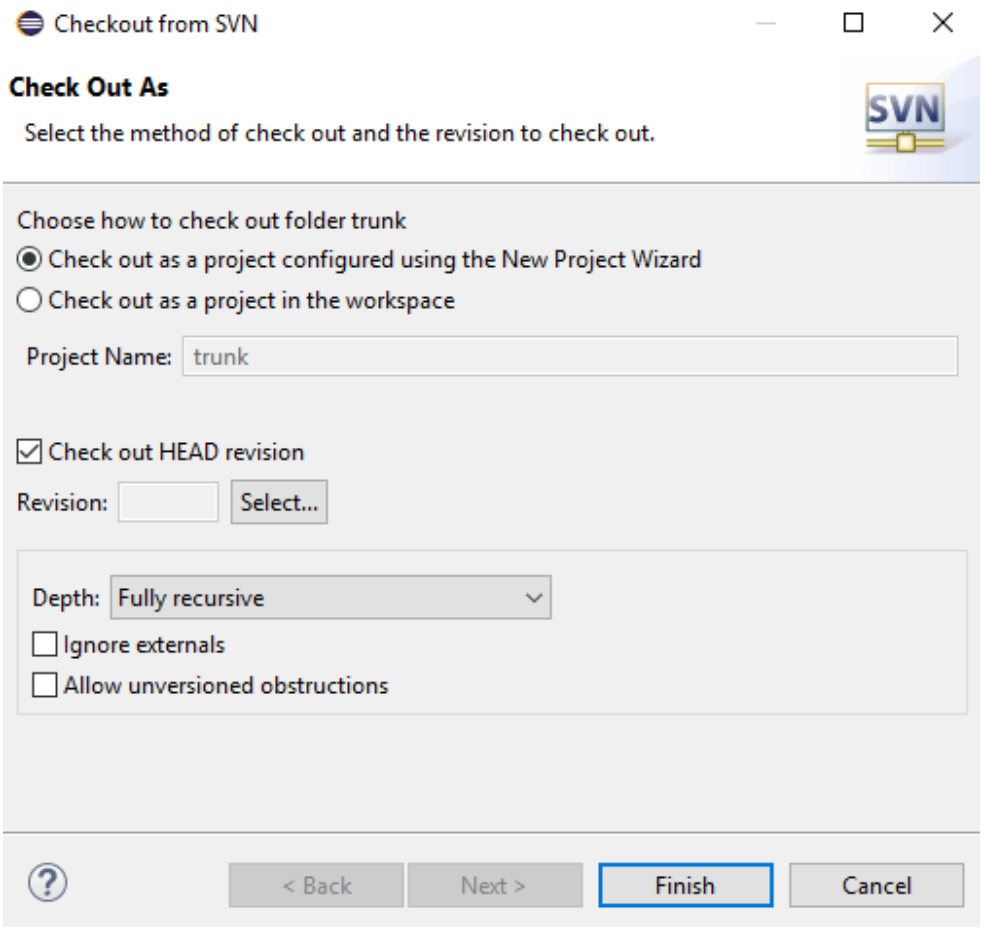
**Figura 18. Instalación de Subclipse.**



Una vez instalado, debemos seleccionar la opción añadir un nuevo repositorio SVN e introducir en el cuadro de texto el siguiente localizador uniforme de recursos (URL) y darle al botón finalizar: http://smi-protege.stanford.edu/repos/protege/

Aparecerá en Eclipse una pestaña en la que pone repositorio SVN y en la que aparecerán diversas carpetas con una serie de archivos. A continuación se buscará la carpeta denominada "protege-core/trunk". En ella se clicará con el botón derecho del ratón para desplegar el menú de opciones, seleccionando la opción Checkout. Una vez se ha abierto esta ventana, se deben seleccionar las opciones que se muestran en la figura 19.

**Figura 19. Selección de parámetros de la ventana Checkout.**

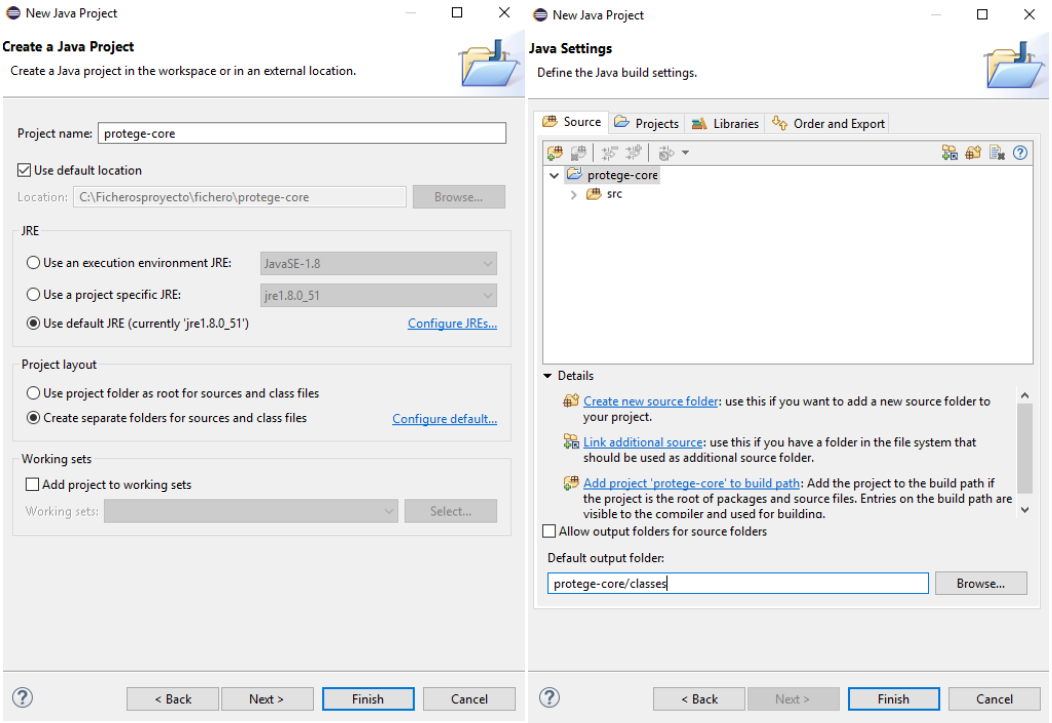



En el asistente de nuevo proyecto que aparece hay que seleccionar la opción "Java Project" y pulsar el botón siguiente. Esto llevará a la aparición de un cuadro de diálogo y se deben seleccionar las opciones que aparecen marcadas en la figura en la parte izquierda de la imagen 20, pulsando después el botón siguiente de nuevo. En la siguiente ventana que aparece, que es la que se muestra a la derecha de esa misma figura, se debe marcar lo que en ella aparece y pulsar finalizar.

En este momento, Eclipse comenzará a comprobar el código fuente de Protégé-Core. Cuando el proceso concluye, se puede observar que aparece en el explorador de paquetes una carpeta con el nombre protege-core[protege-core/trunk], pero que muestra errores, porque todavía falta su configuración. Para ello, se clica en ella con el botón derecho del ratón y se

selecciona la opción propiedades. Aparecerá una ventana emergente en la que se debe seleccionar "Java Build Path" en el margen izquierdo y hacer click en la pestaña librerías. Una vez realizado esto, se pulsará en el botón "Add Jars" y se seleccionará la carpeta protege-core/lib. En esa carpeta se seleccionarán todos los archivos JAR que aparecen y se pulsará OK. Estos ficheros aparecen ahora incorporados en la pestaña librerías y por tanto sólo queda pulsar el botón OK para finalizar este proceso.

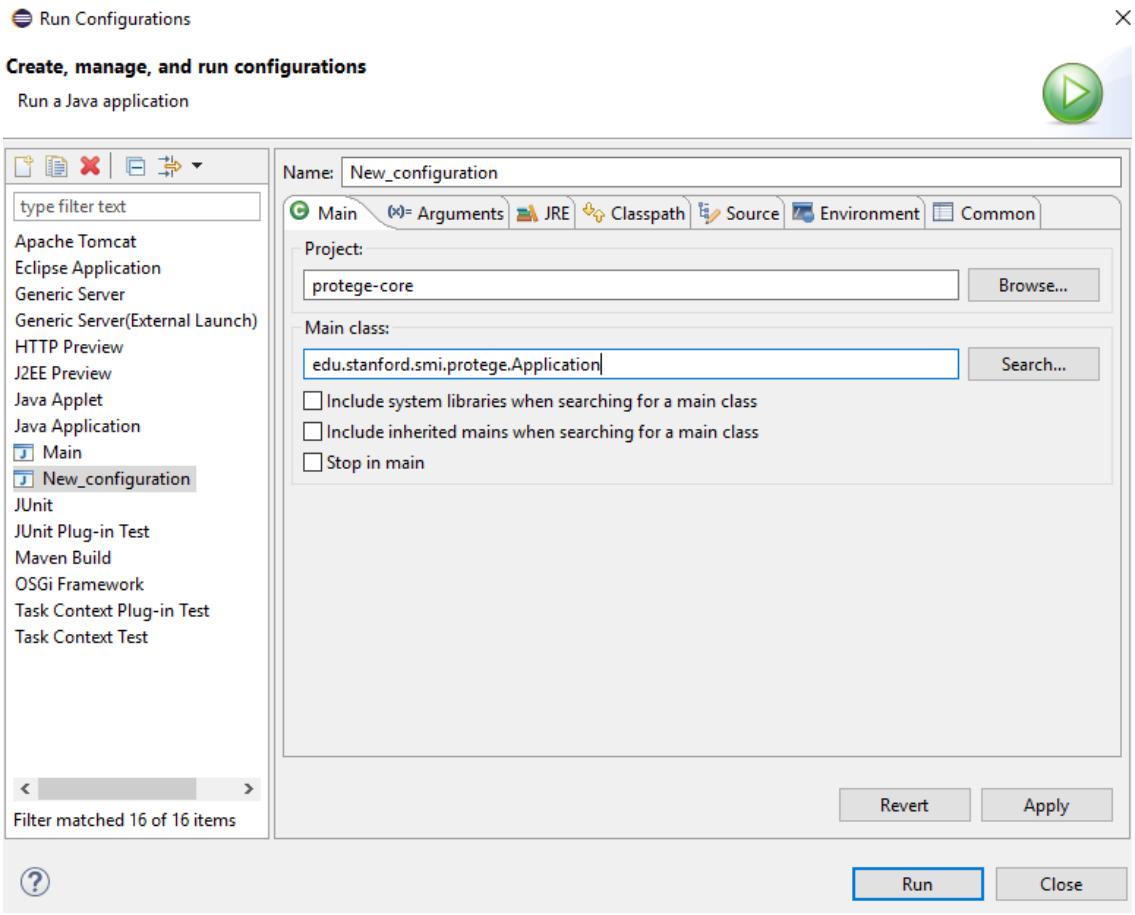
**Figura 20. Selección de parámetros del nuevo proyecto.**




En este momento, Eclipse comenzará a comprobar el código fuente de Protégé-Core. Cuando el proceso concluye, se puede observar que aparece en el explorador de paquetes una carpeta con el nombre protege-core[protege-core/trunk], pero que muestra errores, porque todavía falta su configuración. Para ello, se clica en ella con el botón derecho del ratón y se selecciona la opción propiedades. Aparecerá una ventana emergente en la que se debe seleccionar "Java Build Path" en el margen izquierdo y hacer click en la pestaña librerías. Una vez realizado esto, se pulsará en el botón "Add Jars" y se seleccionará la carpeta protege-core/lib. En esa carpeta se seleccionarán todos los archivos JAR que aparecen y se pulsará OK. Estos ficheros aparecen ahora incorporados en la pestaña librerías y por tanto sólo queda pulsar el botón OK para finalizar este proceso.

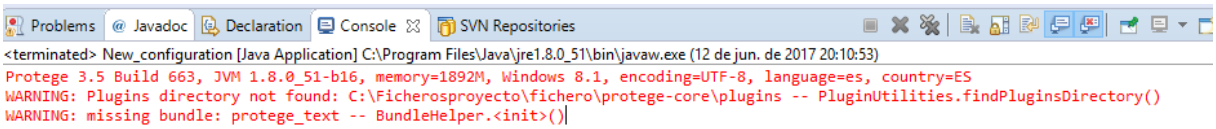
Como se puede observar, ahora ya es posible compilar este código fuente, y para ello, es necesario crear una configuración de ejecución que permita ejecutar Protégé a través de Eclipse. Esto se lleva a cabo pulsando en la pequeña flecha que aparece al lado del botón de ejecución (triángulo verde), y seleccionando la opción "Run configurations". Aparecerá una ventana en la que hay que seleccionar en el menú de la izquierda la opción de aplicaciones Java y pulsar en el botón de nueva configuración de lanzamiento, rellenando los campos tal y como se muestra en la figura 21.

**Figura 21. Configuración de lanzamiento de protege-core.**




Por último se pulsa en aplicar y ya está listo para ejecutarlo seleccionando esta nueva configuración de ejecución. Una vez se ha ejecutado se puede observar que aparecen algunas advertencias en la consola.

**Figura 22. Advertencias en la ejecución de protege-core**




El primero de ellos aparece porque Protégé espera que exista un subdirectorio bajo el directorio de ejecución denominado "plugins". Por tanto, una manera de solucionarlo es creando un directorio vacío con este nombre en la carpeta protege-core del proyecto de Eclipse que se ha realizado y que permitirá que Eclipse lo reconozca en el momento de arranque.

El segundo de los avisos aparece porque Protégé espera un fichero de propiedades Java en el directorio donde se ejecuta. Para solucionar esta advertencia, hay que copiar el fichero protege-core/src/edu/stanford/smi/protege/resource/files/protege_text.properties en el directorio raíz del proyecto protege-core. Este es el directorio desde el cual es lanzado el ejecutable de Protégé.

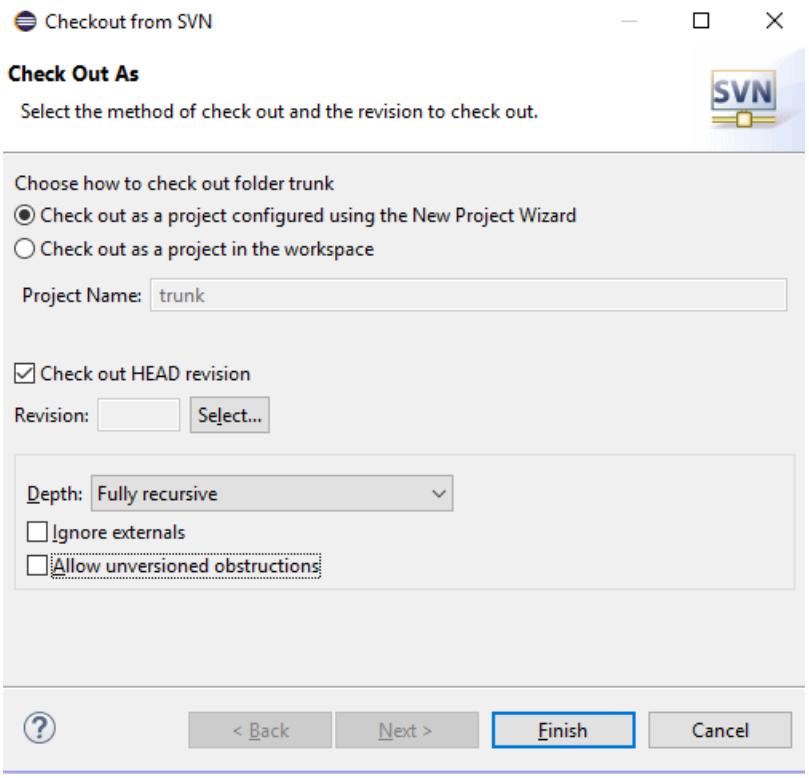
### 3.2.2. Instalación de Protégé-OWL

Protégé-OWL depende de Protégé-Core, por lo tanto no se puede comenzar su instalación si no compila este último correctamente. Para instalarlo, hay que volverse a conectar al repositorio de subversión que se ha indicado en el apartado anterior:

http://smi-protege.stanford.edu/repos/protege/

Una vez que se ha establecido esta conexión, hay que navegar hasta la carpeta owl/trunk y hacer click con el botón derecho sobre dicha carpeta para abrir el menú de opciones. En dicho menú se seleccionará Checkout para poder visualizar la ventana que se muestra en la figura, en la cual se deben marcar los parámetros que se observan y se pulsa finalizar.

**Figura 23. Parámetros en la ventana Checkout.**

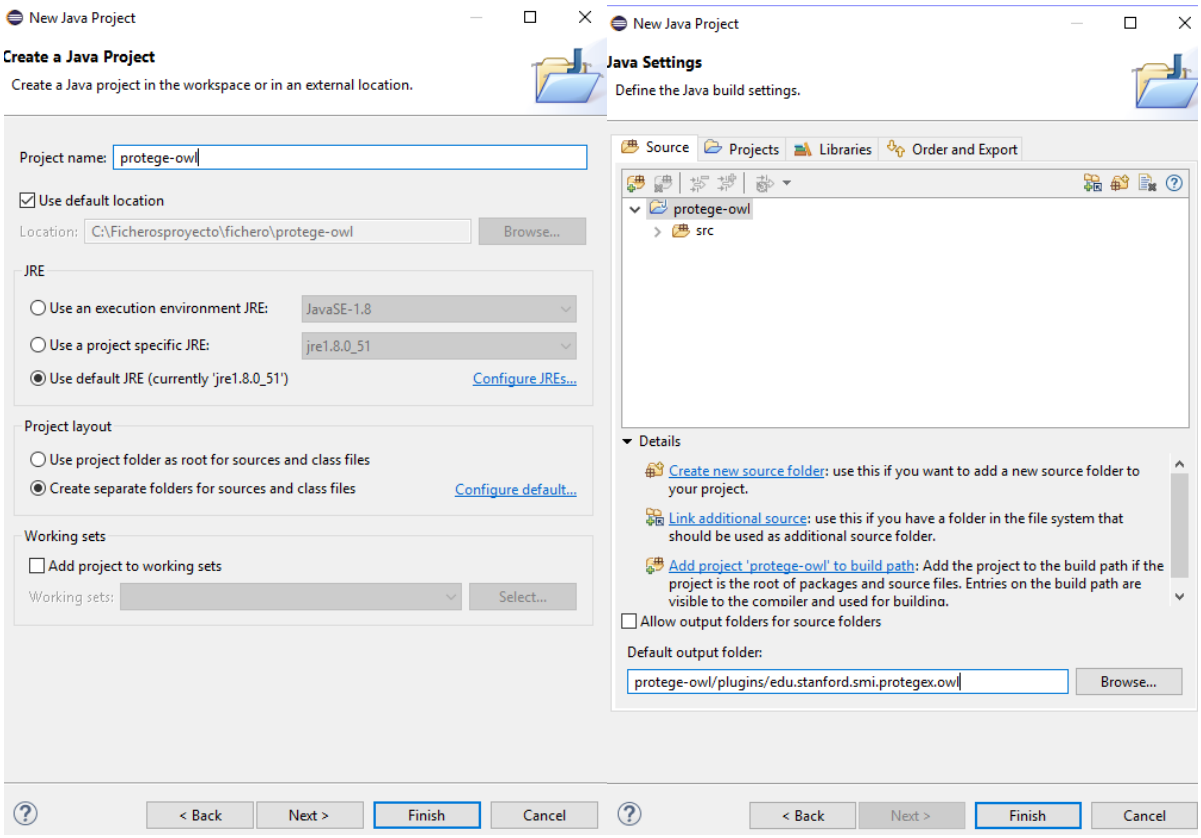



Al igual que ocurría en el apartado anterior, emergerá una ventana de nuevo proyecto, en la que hay que seleccionar la opción de proyecto Java y darle a siguiente. La creación del nuevo proyecto es análoga a la descrita en la instalación de Protégé-Core, pero completando la información que se muestra en la Figura 24.

Al terminar este proceso, se incluye en el explorador de paquetes una carpeta con el nombre protege-owl[owl/trunk], pero que presenta una serie de errores que indican que aún está pendiente su configuración. Para ello se pulsa en dicha carpeta con el botón derecho del ratón y se selecciona la opción propiedades. Aparecerá una ventana emergente en la que se debe seleccionar "Java Build Path" en el margen izquierdo y hacer click en la pestaña librerías. Una vez realizado esto, se pulsará en el botón "Add Jars" y se seleccionará la carpeta

protege-owl/lib. En esa carpeta se seleccionarán todos los archivos JAR que aparecen y se pulsará OK. Estos ficheros aparecen ahora incorporados en la pestaña librerías y por tanto sólo queda pulsar el botón OK para finalizar este proceso.

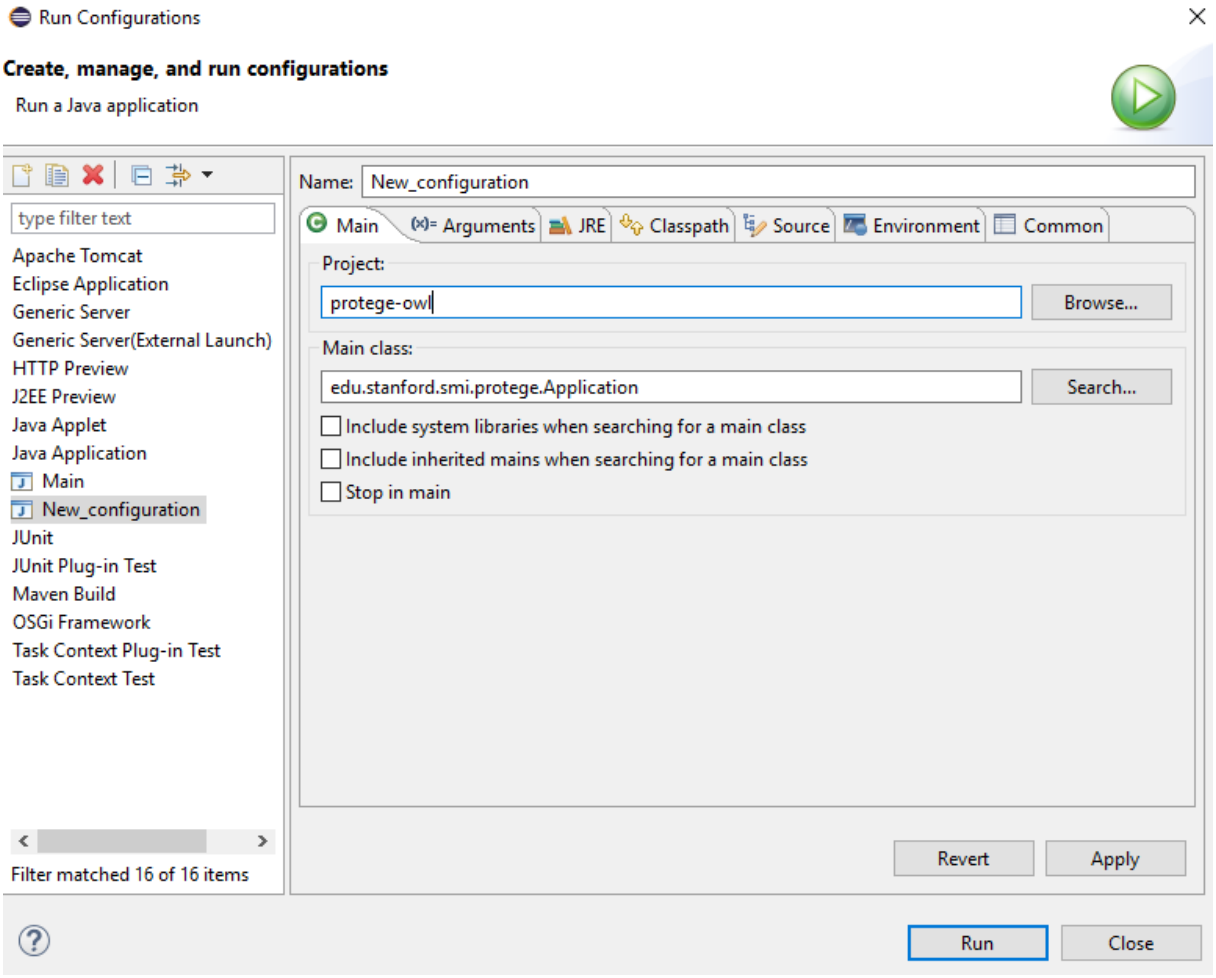
**Figura 24. Selección de parámetros del nuevo proyecto.**




Hay dos pasos más pendientes de realizar antes de poder ejecutar este proyecto desde eclipse. El primero de ellos se debe a que Protégé-OWL espera que ciertos archivos estén presentes en el directorio de salida en tiempo de ejecución. Para ello, hay que copiar todo el contenido del directorio protege-owl/etc al directorio de salida protege-owl/plugins/edu.stanford.smi.protegex.owl. El segundo paso tiene su origen en que Protégé espera que esté presente en el directorio en el cual se ejecuta un fichero de propiedades de registro (logging.properties). Por ello, hay que realizar una copia de dicho archivo que se encuentra en el directorio protege-core en el directorio protege-owl.

Ahora ya es posible compilar este código fuente, pero es necesario crear una configuración de ejecución. Esto se lleva a cabo pulsando en la pequeña flecha que aparece al lado del botón de ejecución (triángulo verde), y seleccionando la opción "Run configurations". Aparecerá una ventana en la que hay que seleccionar en el menú de la izquierda la opción de aplicaciones Java y pulsar en el botón de nueva configuración de lanzamiento, rellenando los campos tal y como se muestra en la siguiente figura. Posteriormente, se pulsa el botón aplicar y se procede a su ejecución.

**Figura 25. Configuración de ejecución de Protégé-OWL.**




### 3.2.3. Jena

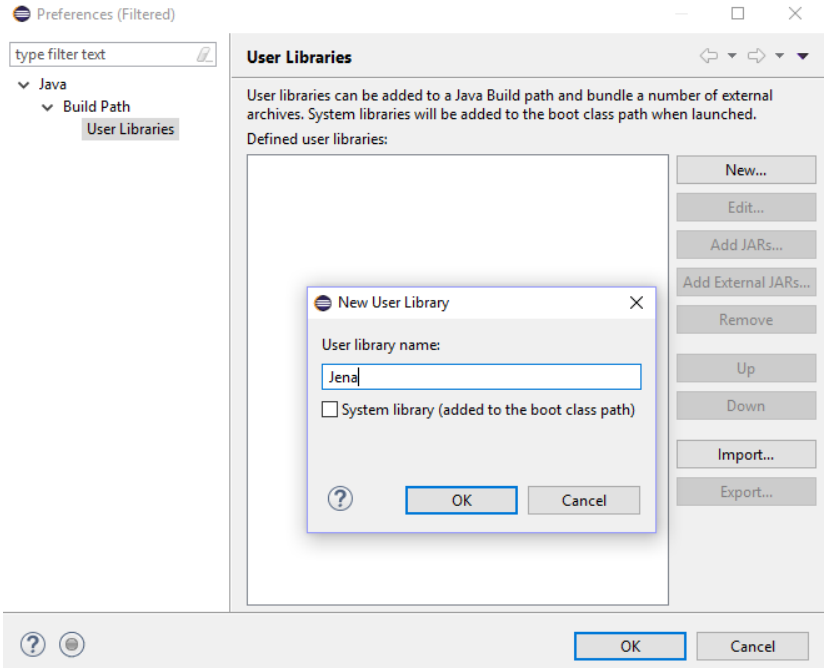
Jena es un programa de código abierto que se creó en el Programa de Web Semántica de laboratorio de HP. Como se ha mencionado anteriormente, es un entorno de trabajo que ha sido desarrollado en Java y que permite construir aplicaciones que funcionan con las tecnologías de la web semántica, suministrando un entorno de programación para el desarrollo de RDF y OWL, e incluyendo además un motor de reglas de inferencia.

Para la integración de Jena en Eclipse hay que descargar inicialmente Jena de la página web https://jena.apache.org/downloads .Posteriormente, se creará un nuevo proyecto Java en el workspace en el que se está trabajando. Dicho proyecto será el que vaya a contener el conjunto de archivos .jar que se han descargado anteriormente.

En este proyecto se pulsará con el botón derecho del ratón y se seleccionará la opción propiedades. En el menú que se muestra a la izquierda de la ventana se debe seleccionar la opción "Java Build Path", y dentro de esta opción hay que pulsar en la pestaña que se denomina librerías. Aparecerá a la derecha otro menú con botones, pues bien, se pulsa el botón añadir librería. Se creará una nueva ventana en la que se puede apreciar a la derecha un botón en el que se muestra librerías del usuario, presionando dicho botón y haciendo click en

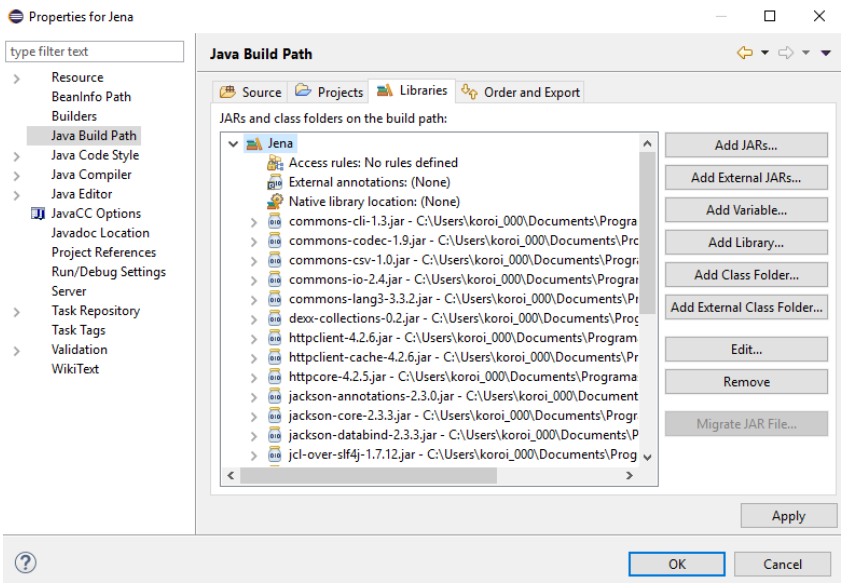
nuevo. Se mostrará una ventana para introducir el nombre la librería que se quiere crear, en este caso se le denominará Jena, tal y como se muestra en la figura.

**Figura 26. Creación de la librería del usuario.**




Una vez que se ha presionado el botón OK y aparece la librería ya creada, hay que presionar el botón "Add JARs…" y se navegará hasta la carpeta donde se habían almacenado los .jar, copiándose todos ellos a esta nueva librería. De esta forma, ya se posee la librería Jena en el Build Path, tal como se aprecia en la figura.

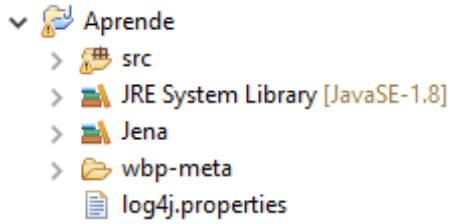
**Figura 27. Apariencia de la librería de usuario Jena.**




Una vez que la librería ha sido creada, lo único que queda es incorporarla al proyecto en el que se vaya a trabajar con las ontologías. Para ello, hay que desplegar con el botón derecho en el proyecto el menú de opciones y se selecciona de nuevo la opción propiedades, Java

Build Path y posteriormente la opción añadir librería. Dentro de esta opción se selecciona librería del usuario y se seleccionará la librería Jena que acaba de ser creada, pulsando por último el botón finalizar. Si se mira el explorador de paquetes, el proyecto en el que se va a trabajar tendrá un aspecto similar al mostrado en la figura.

**Figura 28. Proyecto en Java con la librería Jena.**




Con la instalación de estos tres elementos ya se puede comenzar a crear una ontología y ya se dispone de un entorno de programación adecuado para la creación del programa que va a realizar el aprendizaje de una máquina a través de una base de conocimiento como es una ontología.

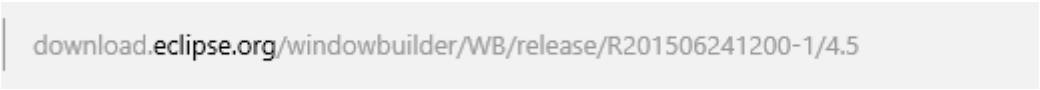
### 3.2.4. Instalación de WindowBuilder.

El proyecto WindowBuilder es un proyecto de código abierto propuesto entre las herramientas del programa Eclipse. Se ha seleccionado esta herramienta para facilitar la creación de la interfaz de usuario que se pide en los objetivos del trabajo de fin de grado. Es una herramienta de programación visual que permite al desarrollador crear una interfaz gráfica de usuario arrastrando y soltando elementos de una paleta en una superficie de diseño y que se encarga de la generación del código Java de la misma. Esto permite dedicar menos tiempo y dinero a los desarrolladores puesto que permite centrarse en crear una serie de funcionalidades específicas de la aplicación en vez de codificar la lógica de bajo nivel para que las interfaces gráficas se ejecuten.

Este constructor fue creado en 2003 por Instantiations y está considerado como el mejor constructor de interfaces gráficas en el mundo Java, de ahí que se haya seleccionado para su utilización en este proyecto. Además, el código Java que genera no requiere ninguna biblioteca personalizada adicional para su compilación y ejecución.

Se puede descargar el constructor en www.eclipse.org/windowbuilder/download.php. Ahí se debe seleccionar el constructor de acuerdo a la versión de Eclipse que se esté utilizando, en el caso de este proyecto la versión Mars. Una vez hemos pinchado en una determinada versión, se cargará una página con una URL que se debe copiar.

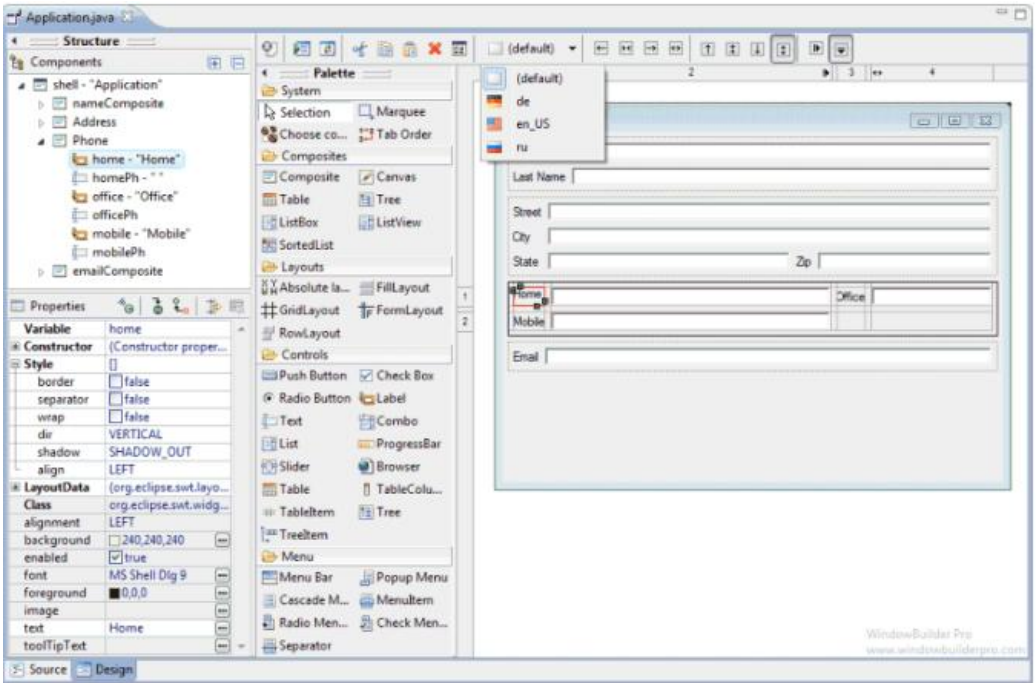
**Figura 29. URL de WindowBuilder.**




Posteriormente, habrá que ir a Eclipse y pulsar en la pestaña ayuda y dentro de ella en instalar un nuevo Software. En la ventana que se abre se pegará la URL anteriormente copiada y se pulsará el botón añadir para añadir el nuevo repositorio, renombrándolo si se desea.

Cuando se finalice, aparecerá una ventana en la que se deben marcar todas las opciones disponibles para instalar y pulsar siguiente. A continuación se mostrarán todos los paquetes a instalar, hay que aceptar la licencia de términos de uso y esperar a que finalice. Cuando esto ocurra, habrá que reiniciar eclipse y ya se dispondrá del plugin instalado. Al abrirlo (New-Other-WindowBuilder-Swing Designer- Application Window), se mostrará el constructor, que posee un aspecto como el de la figura.

**Figura 30. WindowBuilder.**




Fuente: www.eclipse.org/windowbuilder

# 4. RESULTADOS Y DISCUSIÓN

En este capítulo se va a explicar cómo se han ido desarrollando las clases que forman parte del programa en Java que se ha creado para realizar este aprendizaje automático. Se ha estructurado de la forma en la que se han ido creando poco a poco, partiendo de las clases más básicas con las que se comenzó el programa y que son las que permiten la creación de la ontología, así como poblar de conocimiento la misma; hasta llegar a las clases que permiten la búsqueda de información o que proporcionan los elementos para manejar de manera sencilla este programa.

## 4.1. CREACIÓN DE LA ONTOLOGÍA.

Antes de comenzar a desarrollar cualquier tipo de aprendizaje, ha sido necesario invertir un cierto tiempo en analizar cómo se lleva a cabo la creación de una ontología en lenguaje Java desde Eclipse en Protégé. Por tanto, cuando ya se cuenta con el entorno de trabajo adecuado, cuya instalación se ha descrito en el apartado anterior, es el momento de empezar a probar cómo es su funcionamiento.

Es fundamental comenzar creando ontologías sencillas de ejemplo con algunas clases e instancias, aunque no estén relacionadas con el fin del trabajo. Esto facilita la posterior creación de funciones para el almacenamiento de los conceptos y ayuda a familiarizarse con el lenguaje Jena.

La primera ontología que se ha creado dentro de este proyecto, para practicar estos comandos cuando se instaló el entorno de trabajo, ha sido una ontología sobre los animales siguiendo un ejemplo que se encontró explicado. El aspecto de la misma es el que se muestra en la figura. Posteriormente se realizaron algunas ontologías algo más complejas y poco a poco se fueron enfocando a lo que se quería obtener tras un aprendizaje. Se van a ver a continuación algunas funciones interesantes a la hora de su creación.

**Figura 31. Primera ontología creada con el API Protégé-OWL para Eclipse**


Lo primero es ser capaz de crear un archivo que vaya a contener dicha ontología. Para ello, hay que tener en cuenta que lo que se debe hacer es crear un modelo de la ontología que se quiere realizar en el proyecto en el que se está trabajando. Los comandos que se utilizan para ello en este caso se muestran en la figura. También se observa que es necesario introducir un namespace, es decir, una especie de contenedor abstracto en el que un grupo de identificadores únicos pueden existir.

**Figura 32. Código para crear un modelo de una ontología.**

```
//Creamos un modelo de la ontología (es una extensión del modelo de Jena rdf)
OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
//Establecemos el NameSpace por defecto para nuestra ontología
String NS = "relaciones";
model.setNsPrefix (NS, "http://www.owl-ontologies.com/ontologia.owl");
```

Al crear este modelo de ontología, que se trata de una extensión del modelo de la librería Jena para RDF, se heredan una serie de funciones que son las que van a permitir manejar la ontología. A continuación se va a hablar de las funciones que se consideran más interesantes y que se han utilizado en este trabajo.

La función createClass es una función que permite crear una clase en la ontología, que será aquella que represente una entidad o un concepto. En lo que al lenguaje se refiere, una clase a la hora de representar un conocimiento será un sustantivo, puesto que se trata de una entidad fija y no contextual como ocurre con otro tipo de términos como pueden ser los pronombres. Por otro lado, la función addSubClass permite establecer una relación de dependencia entre dos clases, una a la que se considerará súper-clase, que englobará a todas aquellas clases que se considere que dependan de ella. En el caso de la ontología creada sobre los animales, Animales sería la súper-clase y Vertebrados e Invertebrados serían subclases de ésta.

CreateIndividual es una función que permite crear una instancia o elemento de una clase. Este tipo de elementos pueden contener una serie de atributos o propiedades a las que se les pueden asignar distintos valores. Esto se gestiona a través de las funciones createDatatypeProperty y setPropertyValue. Hay más funciones que se pueden utilizar para esto y que se encuentra fácilmente su descripción de cómo utilizarlas en internet.

Además, una vez se creen las clases, instancias o cualquier otro contenido que se quiera almacenar en la ontología, hay que guardarlo en la misma. Acto seguido se muestra en una figura los comandos que se deben utilizar para guardar la ontología en un fichero .owl, que es el formato que se ha elegido en este caso. Se ha decidido almacenar en este tipo de fichero puesto que es un formato típico de almacenamiento de ontologías y que puede ser procesado por cualquier editor de ontologías y por diversos programas. Esta característica es fundamental dentro de los objetivos de este proyecto puesto que se quiere obtener un fichero que pueda ser distribuido, de manera que cualquier persona pueda descargarse esta base de conocimiento para integrarla en sus robots o en sus programas.

**Figura 33. Comandos para el almacenamiento de la ontología.**

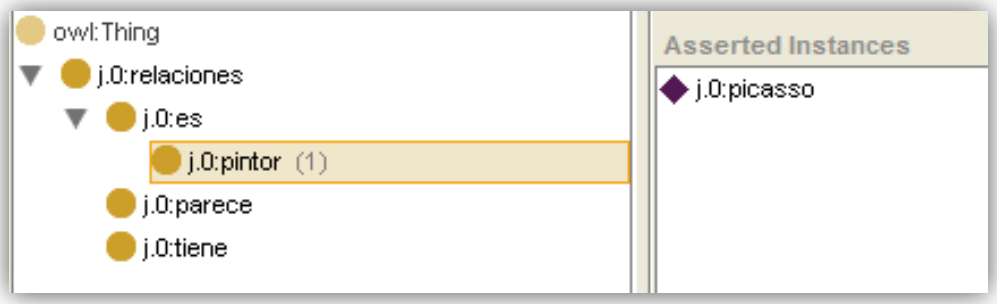
```
//Almacenamos la ontología en un fichero OWL
File file = new File("C:/progproyecto/protege/OntosOWL/ontologia.owl");
//Hay que capturar las Excepciones
if (!file.exists()){
     file.createNewFile();
}
model.write(new PrintWriter(file));
```

## 4.2. CONSTRUCCIÓN DEL MODELO DE TRIPLETAS.

Una vez se conoce el funcionamiento del API Protégé-OWL y se es capaz de crear ontologías, independientemente del contenido de las mismas, es el momento de definir cómo debe ser el formato de la ontología que se quiere crear.

El objetivo es realizar un aprendizaje automático orientado al diálogo con el usuario, por lo tanto, para el tratamiento de la información que se reciba se ha seleccionado una simplificación en forma de tripletas. Las tripletas constan de tres elementos: un sujeto, un predicado y un objeto como se comentó en la introducción. Por lo tanto, como de la información que se reciba se hará una simplificación a esta estructura (sujeto, predicado, objeto), se ha llevado a cabo la decisión de que esta información se almacene en una ontología tal y como se muestra en la figura 34. Las clases superiores serán las relaciones y las subclases los sustantivos que actúan como objetos y que aportan información sobre un determinado sujeto. Asimismo, los sujetos se almacenarán como instancias de los objetos a los hacen referencia. Se pueden añadir propiedades o atributos a dichas instancias, aunque el único atributo que actualmente se plantea añadir es un índice de confianza, que será el que indique cómo de cierta puede considerarse esa información.

**Figura 34. Forma de almacenar el conocimiento**




Actualmente este índice de confianza se ha establecido por repetición. Es decir, se considera que este índice debe aumentar cada vez que se proporciona una información, de este modo, si repiten tres veces que Picasso es pintor, tiene más probabilidad de ser veraz esta información y por tanto tendrá un índice de confianza mayor que si esta información sólo la has oído una vez. En un futuro, se quiere poder contrastar diversas fuentes para conocer la veracidad de la información que se extrae. Sin embargo, en este trabajo sólo se tiene como motor de búsqueda la enciclopedia Wikipedia y otra fuente sería la información que proporcionan los usuarios al robot.

Después de definir la estructura de la ontología, se ha creado la clase AddOnto, que será la que, una vez se posean estas tripletas, las añada a la ontología. En esta función lo primero que se hace es leer la ontología para poder conocer el contenido. Una vez que se carga la ontología se comprueba lo primero si la relación, (a lo que se ha llamado anteriormente predicado), existe o no existe. Si no existe la relación se añade ésta a la ontología y se procede a añadir el objeto como clase y el sujeto como instancia de esta clase. En el caso de que la relación ya exista en la ontología, lo que se hace es comprobar si ya existen el objeto y el sujeto para añadirlas correctamente y evitar duplicar informaciones. Si ya existiesen los tres elementos lo único que queda sería aumentar el índice de confianza.

## 4.3. APRENDIZAJE AUTOMÁTICO.

Uno de los principales objetivos de este proyecto era la creación de un aprendizaje automático. Hasta ahora en los trabajos que se habían realizado previamente, como se ha mencionado en la introducción, las ontologías estaban previamente diseñadas por el usuario. En este trabajo por el contrario, lo que se quería crear era una ontología que se diseñase directamente desde el aprendizaje.

Para ello, era fundamental además de que el usuario pudiese introducir información en un momento dado, que el robot en un momento de inactividad pudiese dedicarse a buscar más información en función de su estado de ánimo o de algo que le motive a realizar algún aprendizaje. Por ello, se estudió la posibilidad de la búsqueda de conocimiento a través de internet.

Inicialmente se barajó la posibilidad de obtener la información de la Real Academia de la Lengua Española, página que se consideraba idónea por la veracidad de toda la información que contiene. Sin embargo, tras la realización de varias pruebas para acceder a esta información esta enciclopedia tuvo que ser descartada. El motivo principal de esto fue que la estructura del contenido de esta página variaba para las diferentes definiciones que se daban, por lo tanto, se hacía muy complejo el tratamiento de las definiciones, sinónimos, ejemplos, etc. que en ella se contenía.

Tras realizar una búsqueda de otro tipo de enciclopedias de las que poder extraer contenido, la que presentaba una estructura más homogénea y que contenía más información sobre temas muy diversos fue Wikipedia. Por tanto se seleccionó finalmente Wikipedia como el motor de búsqueda de contenido para poder poblar la ontología que se iba a crear.

### 4.3.1. Consultas a Wikipedia.

Wikipedia es una enciclopedia libre administrada por la Fundación Wikimedia. Fue creada en el año 2001 por Jimmy Wales y Jerry Sanger y es la enciclopedia más popular y más consultada de internet. Está disponible en varios idiomas, pero en lo que a este trabajo respecta, en español cuenta con más de 300.000 artículos disponibles.

Las primeras pruebas de búsqueda de conocimiento en esta enciclopedia se realizaban en los artículos a los que cualquier usuario accede normalmente cuando quiere consultar algo en Wikipedia. El problema que se encontró fue que los artículos contenían muchísima información, con oraciones muy complejas, en muchos casos con oraciones coordinadas y subordinadas, muy difíciles de tratar. Por tanto era muy compleja la creación de una clase que generase tripletas de todas estas oraciones, aun contando con la ayuda de un analizador sintáctico. De toda la extensión de los artículos, en numerosos casos apenas era posible extraer un par de tripletas, y estas tripletas muchas veces carecían del tipo de información que se quería buscar, proporcionando información que no era de ninguna utilidad para la ontología.

A continuación se muestra un extracto de la información obtenida de Wikipedia al buscar quién es Barack Obama. En él se pueden observar las longitudes de las frases, así como las distintas estructuras que presentan las mismas.

**Figura 35.Ejemplo de la información obtenida de Wikipedia.**

**Barack Hussein Obama II**[1] (🔊 [bəˈrɑːk huːˈseɪn oʊˈbɑːmə] (?·i) en inglés americano; Honolulu, Hawái, 4 de agosto de 1961) es un político estadounidense que fue el 44° presidente de los Estados Unidos de América desde el 20 de enero de 2009 hasta el 20 de enero de 2017. Fue senador por el estado de Illinois desde el 3 de enero de 2005 hasta su renuncia el 16 de noviembre de 2008. Además, es el quinto legislador afroamericano en el Senado de los Estados Unidos, tercero desde la era de reconstrucción. También fue el primer candidato afroamericano nominado a la presidencia por el Partido Demócrata y es el primero en ejercer el cargo presidencial.

Fuente: Wikipedia.

Buscando una forma más simple de obtener el conocimiento de esta enciclopedia, se encontraron los Wikidatas. Wikidata es como una base de datos del almacenamiento estructurado de los proyectos hermanos del grupo Wikimedia. En esta parte de la enciclopedia se obtienen definiciones mucho más sencillas de tratar, tal y como se mostraba en el tercer capítulo  al presentar las herramientas que se iban a utilizar. Además cuenta con un apartado en el que se indican elementos relacionados con el término que se está buscando proporcionando ideas sobre de qué puede ser ese término una subclase o una instancia, son lo que Wikidata denomina declaraciones. Actualmente no se están utilizando estos elementos en el proyecto, sólo se realiza una búsqueda de la definición que proporciona Wikidata, pero puede ser interesante en un futuro para ampliar aún más la información que se puede extraer sobre un determinado término o concepto. En este trabajo no se ha utilizado porque normalmente esta información está en inglés y también habría que realizar una labor de traducción, por lo que añadiría una mayor complejidad al programa.

**Figura 36 Ejemplo de las declaraciones de Wikidata.**


Fuente: Wikidata

Para acceder a esta información se ha creado una clase, denominada CWeb. Esta clase está formada por varias funciones con las que se puede extraer la información de diversas formas. A continuación se van a ir comentando las diversas funciones.

- Wiki(). Es la función que se encarga de establecer la conexión con la página de Wikidata y que permite la lectura de la página correspondiente en cada caso. También se encarga de almacenar la información que se extrae en un fichero de texto, al que se ha denominado input.txt, para que posteriormente pueda ser tratada y almacenada en la enciclopedia.

- RandomLearn(int tope). Esta función permite la búsqueda aleatoria de contenido. Se puede introducir un tope para limitar la cantidad de información aleatoria que se quiere obtener, para que no esté buscando de manera indefinida.

- ModelLearn(String what, int tope). Se trata de una función que permite buscar artículos con una temática determinada. En ella se debe introducir qué se desea buscar y un tope como ocurría en el caso anterior. Esta función podría utilizarse para añadir instancias a una determinada clase.

- UnoLearn(String Numb). Permite buscar un único término. Lo característico de esta función es que la búsqueda se realiza mediante un número. Esto ocurre porque cada información que se almacena en Wikidata se referencia con un identificador formado por una Q seguido de un número. Por ejemplo, el identificador "Q1028181" hace referencia a la palabra "pintor", como se puede ver en https://www.wikidata.org/wiki/Q1028181. Por tanto, introduciendo diferentes números se van obteniendo nuevos términos. Esta función resulta útil cuando se quiere realizar una búsqueda aleatoria de un solo término, ya que introduciendo números al azar se van generando contenidos nuevos.

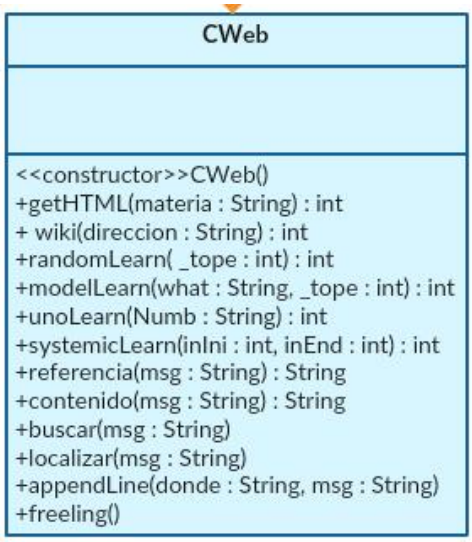
- SystemicLearn(int inIni, int inEnd). Se basa en la idea anterior de los identificadores y permite realizar una búsqueda sistemática en la enciclopedia, de forma que se pueden ir programando intervalos para poco a poco realizar un barrido de todo el contenido que almacena Wikipedia.

Esta clase cuenta con alguna función más necesaria para que se pueda buscar en español la información o para localizar la definición dentro de la página de Wikidata. A modo de resumen de todas las funciones que contiene, se va a mostrar su diagrama UML a continuación.

Más adelante, se mostrará cómo se relaciona esta clase con las demás clases del programa y se verá cómo se utiliza y cómo se accede a ella. Es una clase fundamental en este programa puesto que es la que va a proporcionar ese bucle de aprendizaje automático, y que como ya se ha mencionado anteriormente es la característica distintiva de este proyecto respecto a los que se habían estado realizando anteriormente en los proyectos citados en la introducción.

Tal y como se muestra en la figura, esta clase también contiene la función freeling(). Esta función es la encargada de llamar al analizador sintáctico, cuyo funcionamiento se va a exponer en el siguiente apartado. Esta función es la que proporciona el paso necesario para luego tratar las frases y añadir nuevos términos a la ontología.

**Figura 37. Diagrama UML de la clase CWeb.**




### 4.3.2. Tratamiento de la información con Freeling.

Como se mencionaba en las herramientas utilizadas, se ha trabajado con el analizador sintáctico Freeling, desarrollado por la Universidad Politécnica de Cataluña.

Freeling es una biblioteca orientada al desarrollador que ofrece servicios de análisis del lenguaje y disponible en numerosos idiomas. En este proyecto únicamente se está realizando una ontología en Español, luego en su configuración hay que seleccionar éste como idioma. Por otro lado, Freeling cuenta con varios modos de funcionamiento, entre los cuales hay que seleccionar el que proporcione la salida del análisis más adecuada a la aplicación que se le vaya a dar.

**Figura 38. Servicio de análisis disponible para cada lengua**

| | as | ca | cy | en | es | gl | it | pt | ru |
|---|---|---|---|---|---|---|---|---|---|
| Tokenization | X | X | X | X | X | X | X | X | X |
| Sentence splitting | X | X | X | X | X | X | X | X | X |
| Number detection | | X | | X | X | X | X | X | X |
| Date detection | | X | | X | X | X | | X | X |
| Morphological dictionary | X | X | X | X | X | X | X | X | X |
| Affix rules | X | X | X | X | X | X | X | X | |
| Multiword detection | X | X | X | X | X | X | X | X | |
| Basic named entity detection | X | X | X | X | X | X | X | X | X |
| B-I-O named entity detection | | | | X | X | X | | | |
| Named Entity Classification | | | | X | X | | | | |
| Quantity detection | | X | | X | X | X | | X | X |
| PoS tagging | X | X | X | X | X | X | X | X | X |
| WN sense annotation | | X | | X | X | | | | |
| UKB sense disambiguation | | X | | X | X | | | | |
| Shallow parsing | X | X | | X | X | X | | X | |
| Full/dependency parsing | X | X | | X | X | X | | | |
| Coreference resolution | | | | | X | | | | |

Fuente: Manual de usuario de Freeling

En la figura anterior se observa que la librería ofrece clases para transformar los datos. En este caso interesaba utilizar la clase Morfo, que recibe una lista de objetos marcada como una frase completa y analiza morfológicamente cada palabra de cada una de esas frases. Esta clase es, sencillamente, un meta-analizador que aplica una cascada de analizadores especializados, cada uno de los cuales es a su vez una clase de procesamiento que puede ser llamada independientemente si se desea.

Dentro de los modos de funcionamiento dentro de esta clase se barajaron fundamentalmente dos opciones. Utilizar el modo Shallow o el modo DEP. El modo Shallow proporciona una salida fiable con una estructura de sintagmas plana. Por otro lado, el modo DEP proporciona una salida más profunda y que sirve para el tratamiento de casos más complejos. Optándose finalmente por este último.

Para poder ejecutar esta librería en el proyecto, hay que modificar el archivo analyzer.bat. Para ello es necesario tener instalado el Visual Studio puesto que Freeling está desarrollado en lenguaje C++. En el fichero analyzer.bat hay que establecer un directorio y hay que llamar al archivo analyzer.exe indicándole el idioma en que se desea realizar el análisis, en este caso español (es), e indicar los documentos de texto de entrada donde se contienen las frases a almacenar y de salida donde se desea obtener el análisis. El directorio del fichero de entrada se escribe precedido por el caracter "<". Sin embargo, el directorio del fichero de salida lo hace precedido por ">".

**Figura 39. Modificación del archivo analyzer.bat**

analyzer.bat

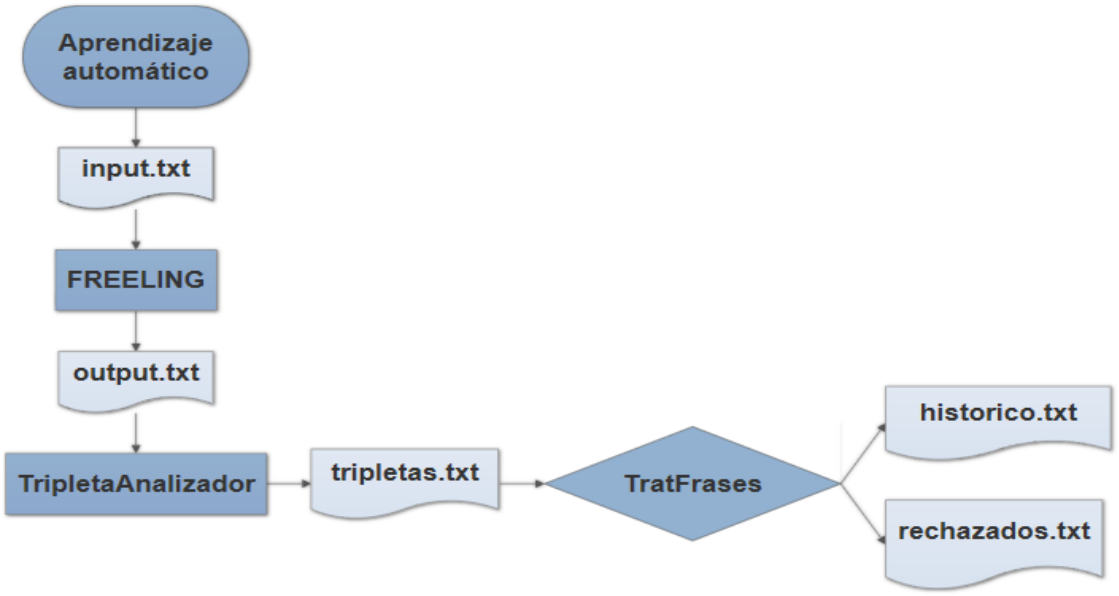
```
@Echo OFF

set BASEDIR=C:\progproyecto\Freeling\bin\

set OLDPATH=%PATH%
set PATH=%PATH%;%BASEDIR%\..\dependencies\boost\lib\x86;%BASEDIR%\..\dependencies\boost\lib\x64;%BASEDIR%\..\dependencies\zlib\lib\x86;%BASEDIR%\

set FREELINGSHARE=%BASEDIR%..\data
rem //--outlvarg Output analysis level (ident,token,splitted,morfo,tagged,shallow,parsed,dep)

analyzer.exe -f C:\progproyecto\Freeling\data\config\es.cfg  <C:\progproyecto\protege\input.txt >C:\progproyecto\protege\output.txt
```

Se ha observado en algún ejemplo que Freeling puede, previa modificación del código del programa, extraer tripletas directamente y proporcionarlas como salida al análisis realizado. Esto sería interesante para unificar de esta forma el analizador con la función TripletasAnalizador, de la que se hablará más adelante, pudiéndose extraer así un mayor número de tripletas. Puesto que el objetivo de este trabajo no es el procesamiento del lenguaje natural, en trabajos futuros es una mejora que se podría implementar. De este modo se estaría ganando en eficacia y en precisión.

### 4.3.3. Sistemas de ficheros.

Para toda la gestión del aprendizaje es necesario ir almacenando la información en todos los pasos, para que sea un aprendizaje eficiente. Es por esto que a lo largo del programa se van utilizando diversos ficheros en los que se guardan los resultados de las diversas clases por las que pasan las oraciones. A continuación se muestra un gráfico que resume este sistema de ficheros.

**Figura 40. Ficheros utilizados.**




Para todo el tratamiento de los diversos ficheros, se ha creado una clase con este mismo nombre, Fichero, que se encarga de la apertura, lectura, escritura o cierre de los mismos. Así, cada una de las clases que llevan a cabo algún tipo de tratamiento de estos ficheros, podrá realizar una llamada a esta clase y se evita el estar repitiendo código en cada uno de los procesos. Las funciones con las que cuenta esta clase son las siguientes:

- escribeFichero(String nombreFichero,String cadena,boolean append): Se utiliza en la escritura de las oraciones en los diversos documentos de texto.

- limpiaFichero(String nombreFichero): Se trata de una función que realiza la apertura del fichero y al cerrarlo borra el contenido. Se utiliza fundamentalmente para ir borrando los ficheros que sirven como intermediarios en el proceso de aprendizaje.

- openFichRead(String nombreFichero): Mediante un buffer de entrada esta función sirve para abrir un fichero para su lectura.

- leeFichRead(BufferedReader buffEntrada): Permite la lectura de los archivos.

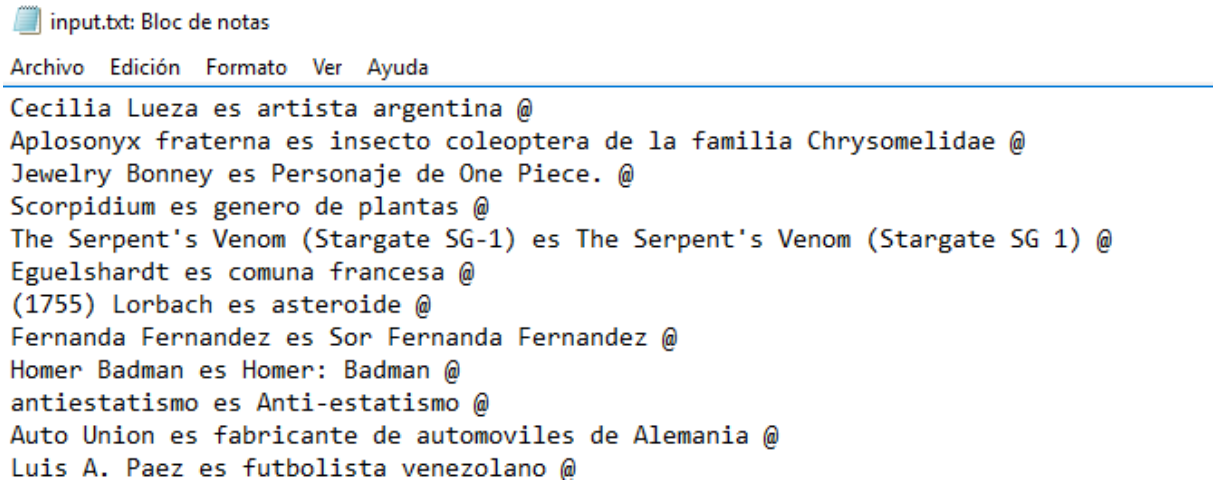
- closeFichRead(BufferedReader buffEntrada) : Cierra los ficheros.

Todas estas funciones son las que permiten el tratamiento de los ficheros que se han utilizado en el programa. Es el momento por tanto de conocer el contenido de cada uno de esos ficheros y el porqué de su creación. Algunos de ellos son meros intermediarios y se va borrando su contenido según van siendo procesados. Otros sin embargo son permanentes.

En primer lugar, de la búsqueda de contenido en Wikipedia, se extrae un fichero que contiene las sentencias con la información de los términos buscados. A este fichero se le ha denominado input.txt y tiene el aspecto que se proporciona en la imagen siguiente. Las "@"

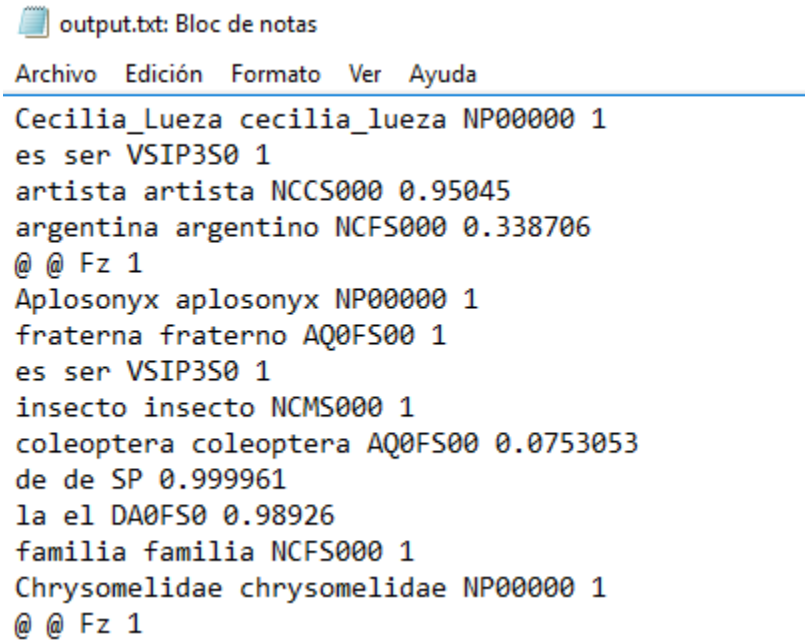
que aparecen al final de la frase son un recurso para conocer dónde termina la información de cada término. Esto facilita la creación de las tripletas, puesto que así se es capaz de saber dónde comienza y dónde termina cada frase.

**Figura 41. Fichero input.txt**




Por otro lado, una vez que este fichero pasa por el analizador sintáctico, se obtiene otro fichero de salida, al que se ha denominado output.txt. Este fichero de salida contiene el análisis sintáctico de toda esta información. A continuación se muestra una figura en la que se ve el contenido del mismo. Como se observa en cada línea, la primera palabra es la palabra que toma del fichero input.txt, la segunda es la palabra genérica que hace referencia a la anterior; por ejemplo si se tiene la palabra es, la segunda palabra que aparece será el verbo ser. Por último, en tercer lugar aparece el tipo de palabra de la que se trata: verbo, sintagma nominal, adjetivo, etc. proporcionando también información sobre el género y número si procede; la cuarta palabra es el índice de confianza del análisis de dicho término.

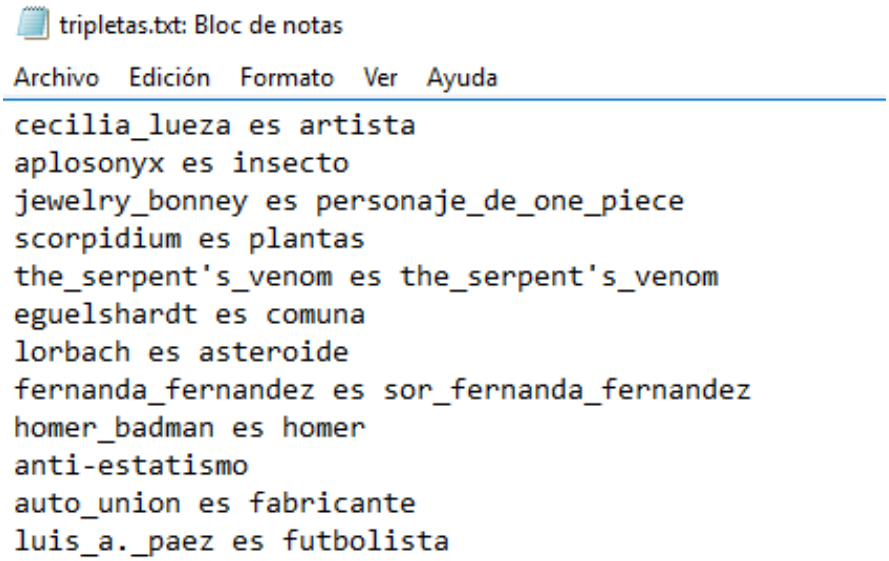
**Figura 42. Fichero output.txt**



Este fichero se pasa a la clase TripletaAnalizador. En esta clase lo que se hace es leer este fichero de salida del analizador y extraer una tripleta en base a esta salida. De esta forma lo que se hace con esta función es mediante un bucle de lectura ir recorriendo los elementos de la frase. En primer lugar se almacenará el sintagma nominal que aparezca en primer lugar. Posteriormente se almacenará el verbo y por último otro sintagma nominal que será el que proporcione la información del primer nombre almacenado. Actualmente no se están teniendo en cuenta adjetivos y otros elementos puesto que el análisis del lenguaje no es objetivo de este proyecto. Se ha creado esta clase para poder almacenar el contenido pero se han visto ejemplos de este analizador que directamente puedes extraer la tripleta como salida del analizador sintáctico. En un futuro se quieren unificar estos dos procesos y trabajar más con el analizador para que se extraigan las tripletas, de esta forma de cada frase podrían obtenerse un mayor número de tripletas y con mayor fiabilidad. Aun así, en la información que se ha ido procesando con esta clase se han obtenido buenas tripletas y se ha creado una base de conocimiento inicial con bastantes definiciones muy interesantes.

El fichero de texto que se obtiene tras pasar el fichero output.txt por esta clase se ha denominado tripletas.txt y presenta el aspecto que se muestra en la figura. Es importante remarcar que este fichero procede solamente del aprendizaje automático. El aprendizaje a través de la información que proporciona el usuario en forma de tripletas es añadido a otro fichero que se ha denominado frases.txt y que es añadido en otro proceso a la ontología. Por tanto en tripletas.txt sólo aparecen las tripletas de la búsqueda de contenido en Wikipedia.

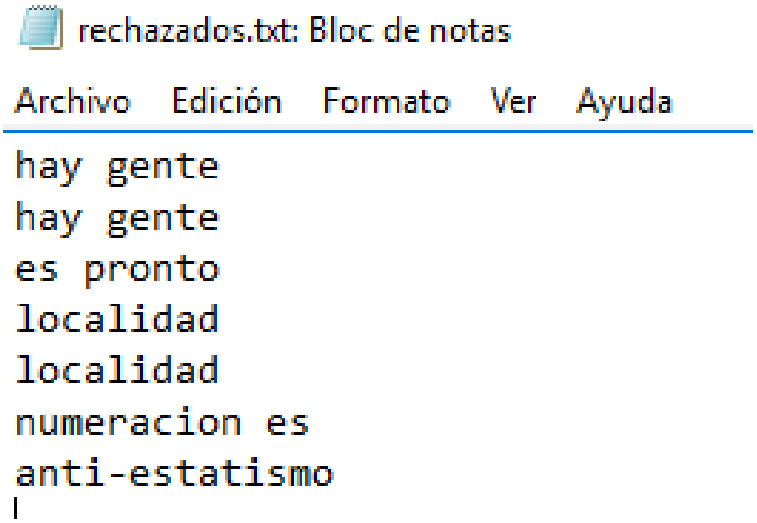
**Figura 43. Fichero tripletas.txt**




Una vez que se tienen creadas las tripletas, este archivo tiene que ser añadido a la ontología. Para ello lo que se ha hecho es crear una clase, TratFrases, en la que se realiza esta acción. En esta clase lo primero que se mira es que la tripleta esté formada por tres palabras, un sujeto, un predicado y un objeto. Esto es fundamental puesto que si no se poseen las tres palabras el programa no va a saber el elemento que está ausente y hay una gran probabilidad de que no se almacenen correctamente los términos. Si el número de palabras es superior a tres, se vuelve a pasar la frase por el analizador para ver si es posible reducirla a una tripleta. Si no estuviesen las tres palabras, lo que se hace es copiar esta frase a un fichero denominado rechazados.txt y no se almacena aún en la ontología. De este modo, en un futuro se puede buscar más información sobre las frases que vayan siendo rechazadas y de esta forma generar

tripletas correctamente para ser añadidas a la ontología. Este sería un proceso que podría ser llevado a cabo en el robot en momentos de inactividad, fomentando la capacidad de analizar por qué no ha entendido antes esa información y realizando búsquedas en internet que le permitan comprender esos conceptos. Esto sería una parte muy importante en un proceso de aprendizaje autónomo. A continuación se muestra un extracto de este fichero, donde se puede observar el tipo de oraciones que actualmente no pueden ser procesadas por el programa.

**Figura 44. Fichero rechazados.txt**




Como se observa en la imagen, son oraciones impersonales o palabras que han quedado sueltas. Las palabras sueltas pueden ser fácilmente consultadas en internet y se podría obtener una definición de las mismas. En el caso de las oraciones impersonales su tratamiento sería bastante más complejo, sin embargo se podría obtener información sobre qué es gente o qué es pronto y así se estaría obteniendo un conocimiento de las oraciones, aunque no se puedan tratar directamente.

Por otro lado, si se observa que las tripletas contienen la estructura deseada y cumplen los requisitos, son añadidas a la ontología tal y como se explicaba al inicio de este capítulo. Una vez que son añadidas a la ontología, esta tripleta se copia en un fichero denominado historicos.txt. Este fichero contendrá todas las tripletas que se han añadido a la ontología, para tener así un fichero que recoja todos los datos que han sido almacenados en la base de conocimiento. El aspecto de este fichero, como se puede observar en la figura, es similar al fichero tripletas.txt pero de mayor extensión, puesto que los ficheros historicos.txt y rechazados.txt son los únicos en los que nunca se elimina contenido. Los demás ficheros, son meros intermediarios, de modo que cuando la información que contienen pasa de una etapa a otra del aprendizaje se borran para evitar repeticiones innecesarias en el tratamiento.

Además, esta estructura de ficheros permite que en un momento dado cualquier tipo de fichero pueda ser introducido al analizador sintáctico y realizar este mismo proceso. Esto es interesante de cara a un futuro, puesto que con esta consideración sería posible el tratamiento de cualquier fichero independientemente de su procedencia o de su extensión. Es cierto que para que esto sea posible habría que cambiar algunas configuraciones del analizador sintáctico, pero se ha visto algún ejemplo y es posible realizar este cambio. Por ello este será un tema susceptible a tratar en proyectos futuros.

**Figura 45. Fichero historico.txt**

historico.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

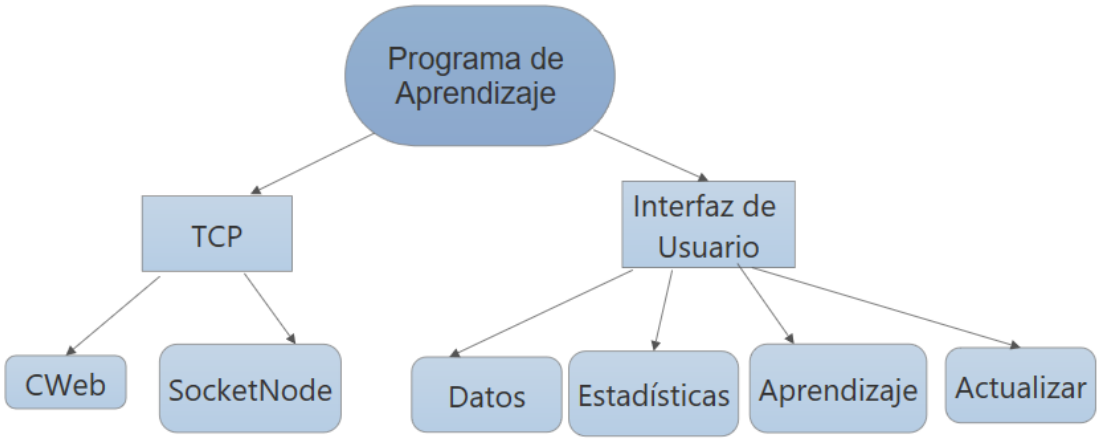
```
picasso es pintor
monet es pintor
Semino_Rossi es cantante
Chicacao es Municipio
Eridanosaurus es mamiferos
Forest es Vorst
Rodrigo_Mannara es futbolista
Hrappsson es vikingo
persona compone piezas
Semino_Rossi es cantante
Chicacao es Municipio
Eridanosaurus es mamiferos
```

## 4.4. MANEJO DEL PROGRAMA.

El programa que se ha creado, es una aplicación en Java que funciona en ordenador y que pretende ser implantada en un robot social. Por ello, a la hora de realizar el trabajo, se ha tenido en cuenta que el programa tenga dos posibles modos de funcionamiento. El primero de ellos es a través de una clase que permite la comunicación vía TCP. Esta conexión está pensada para que el programa pueda ser implantado en el robot Urbano, del que se hacía una breve mención en la introducción. El segundo modo de funcionamiento es a través de una interfaz de usuario que se ha creado con la ayuda de la aplicación WindowBuilder, de la cual se ha comentado en el tercer capítulo cómo se ha llevado a cabo su integración en el programa.

De este modo, se muestra en la siguiente figura como quedaría la estructura del programa creado en forma de esquema. En este capítulo se va a comentar cómo funciona el programa y qué debe ir haciendo el usuario para que se produzca el aprendizaje.

**Figura 46. Esquema del programa principal.**

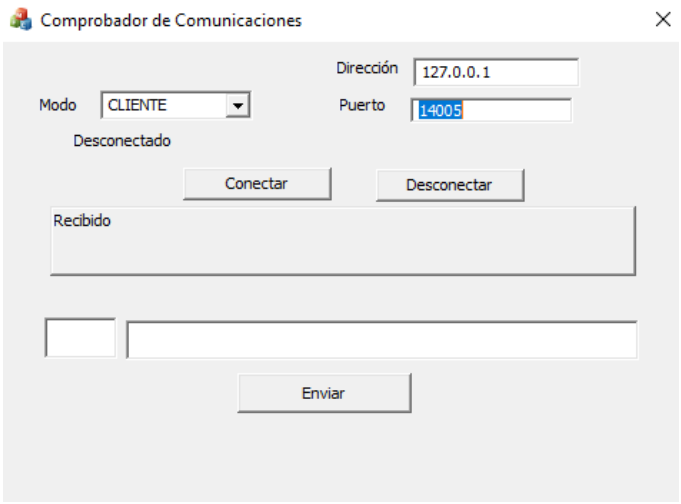


### 4.4.1. Comunicación TCP.

La comunicación TCP (Protocolo de control de transmisión) es uno de los protocolos fundamentales en internet. Muchos programas dentro de una red de datos compuesta por una red de computadoras, pueden usar este tipo de comunicación para establecer conexiones entre sí a través de las que pueden enviarse flujos de datos. Este protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden que se transmitieron. También proporciona un mecanismo para distinguir las distintas aplicaciones dentro de una misma máquina a través del concepto puerto, una interfaz a través de la cual se pueden enviar y recibir datos.

Para comprobar el funcionamiento de esta comunicación se ha utilizado un comprobador de comunicaciones, que simulaba la conexión desde el robot. Para ello, en el comprobador de comunicaciones hay que seleccionar el modo cliente, puesto que el programa actúa como servidor, siendo el que proporciona la información. Además como dirección se introduce la IP: 127.0.0.1 o localhost. Esa es la IP del ordenador en el que se trabaja, ya que todos los ordenadores personales para consultar sus recursos utilizan esta IP.

**Figura 47. Interfaz del comprobador de comunicaciones.**




Para establecer la comunicación se pulsa el botón conectar y se ejecuta el programa en Eclipse. Una vez que se ha establecido la comunicación se envía un número en el cuadro pequeño, que corresponde a la información que se pretende obtener. En el cuadro grande se introduce si fuese necesario algún término o parámetro que necesite conocer el programa a la hora de responder a la demanda realizada.

Dentro del programa Comunicación se ha realizado una función denominada TcpMessage(int code, String msg) que es la que recibe el código para la operación y el mensaje necesario. Esta función se ha realizado mediante distintos case. A continuación se muestra la codificación utilizada.

1. Sirve para pedir estadísticas. Se obtiene una respuesta con el siguiente formato: respuesta=2-número de clases-número de instancias-número de relaciones;

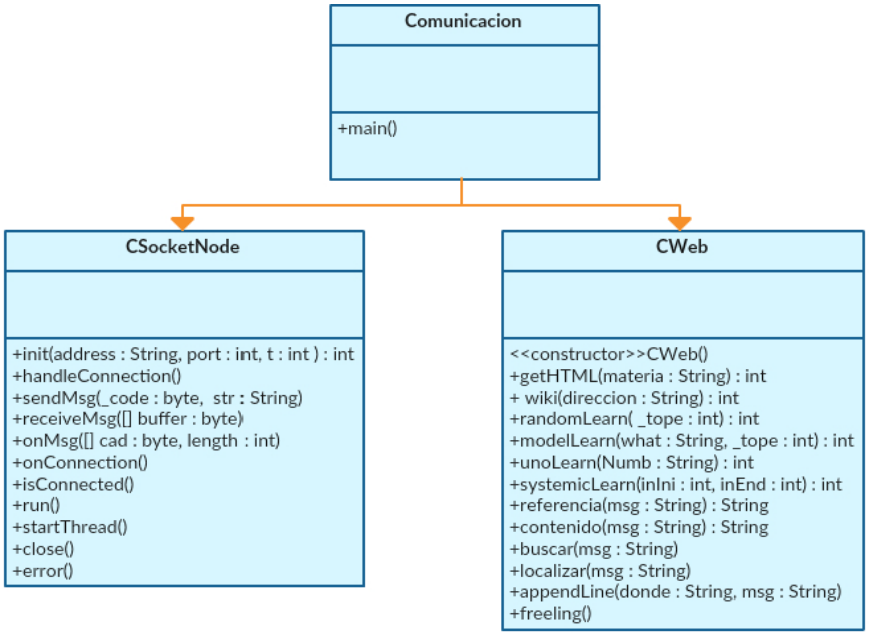
2. A través del mensaje el usuario proporciona un término y el programa te responde si ese término está contenido o no en la ontología. Así mismo proporciona

información sobre si se trata de una clase, una instancia o una relación en caso de que el término se encuentre en la ontología.

3. Dada una clase por el usuario, te proporciona una respuesta sobre los individuos que contiene esa clase en la ontología.
4. Dada una relación por el usuario, el programa muestra las clases que contiene diche relación en la ontología.
5. Define una clase en la ontología cuando se proporciona el nombre de esa clase y de la relación de la que depende.
6. Define una relación.
7. Procesa un fichero de texto en forma de tripletas.
8. Realiza un aprendizaje automático aleatorio en la enciclopedia Wikipedia.
9. Realiza un aprendizaje mediante la búsqueda en la enciclopedia de un término proporcionado por el usuario.
10. Realiza un aprendizaje sistemático de la enciclopedia Wikipedia de un intervalo de páginas.
11. Actualizar la ontología del aprendizaje automático.

En el diagrama UML de la figura se muestra cómo es la estructura de esta parte del programa. Las clases que se utilizan fundamentalmente son CWeb para la gestión del aprendizaje automático y CShocketNode, la clase encargada de gestionar la comunicación TCP. El resto de clases son indirectamente utilizadas para los procesos, tanto la clase Buscador a la hora de pedir estadísticas como la de Fichero para el tratamiento de los documentos de texto. No se han incluido en el diagrama por simplificar su contenido y por remarcar las dos clases fundamentales.

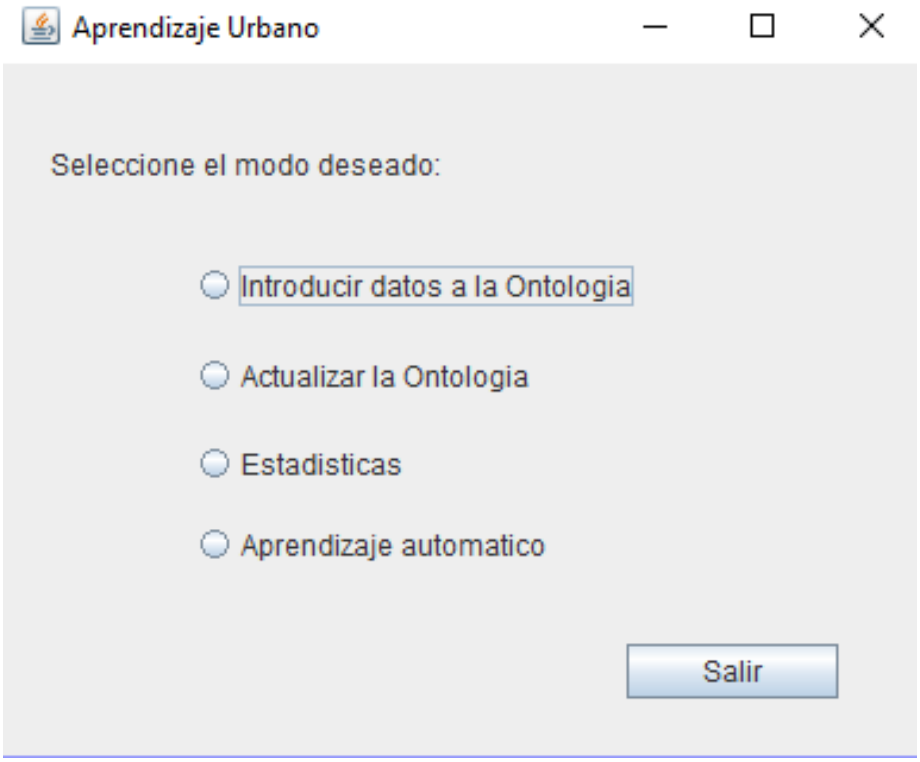
**Figura 48. Diagrama UML de la comunicación TCP.**



### 4.4.2. Interfaz de usuario.

En este proyecto se ha construido una interfaz de usuario que permite la gestión del programa de aprendizaje automático. Cuando se arranca el programa la primera ventana que emerge es un menú de opciones con las tareas que puede realizar el programa, como se muestra en la figura. Para ello, lo único que hay que ejecutar es el Main de esta parte del programa que está contenido en la clase Aplication. Esta clase es la encargada de llamar a la clase VentanaPrincipal que será la que proporcione este menú de opciones.

**Figura 49. Menú de la interfaz de usuario.**



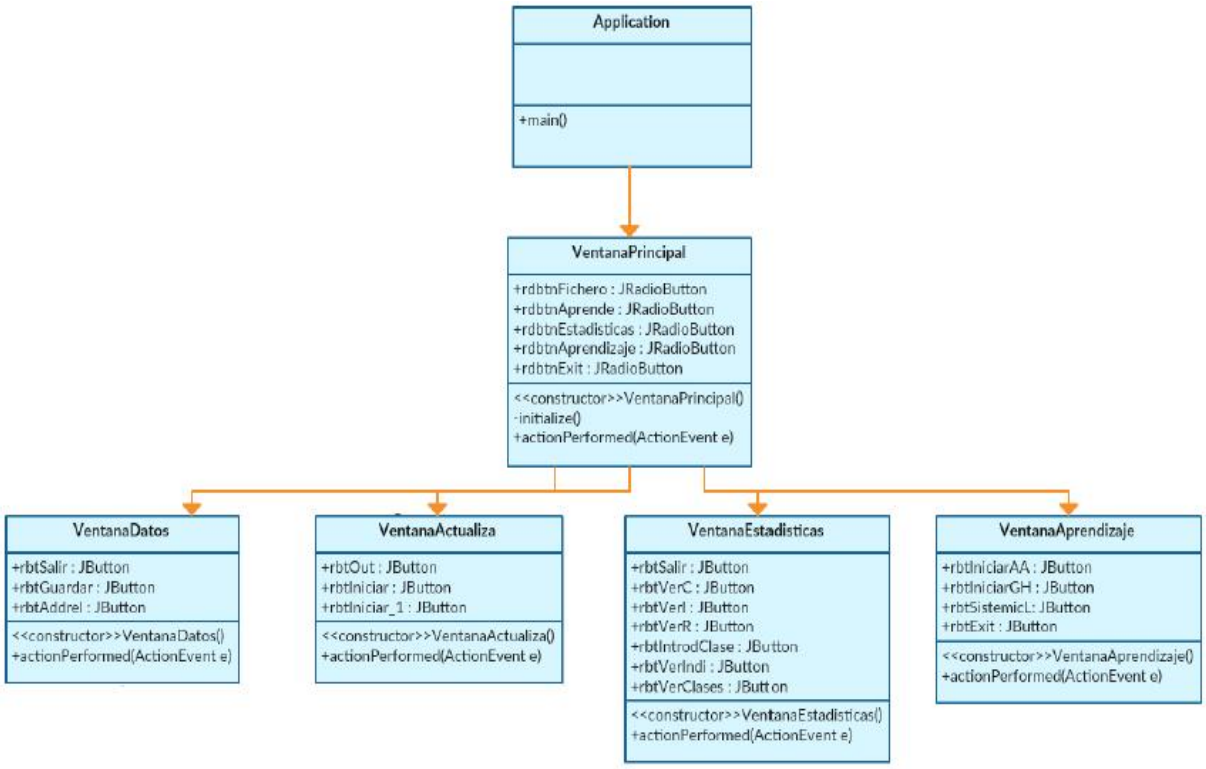

Se van a ir detallando a continuación las tareas que se realizan en cada uno de los elementos del menú de opciones. Además, se irán mostrando en cada caso los diagramas UML. El primero de ellos es el de esta interfaz de usuario.

En este primer diagrama se observa que de las distintas opciones del menú van emergiendo nuevas ventanas. Todas ellas han sido creadas con WindowBuilder, lo que ha simplificado enormemente la creación del código Java de cada una de ellas. Esto ha supuesto un ahorro en horas de programación, muy importante en el cómputo global de horas dedicadas al proyecto.

Cada una de estas nuevas ventanas se ha construido de una forma muy intuitiva para que pueda ser manejada por cualquier usuario. En algunas de ellas, cuando se realiza alguna tarea que puede durar un tiempo mayor, se ha añadido una barra de estados que orienta al usuario sobre el proceso que se está desarrollando.

A continuación se detalla cada una de estas ventanas secundarias.

**Figura 50. Diagrama UML de la interfaz de usuario.**



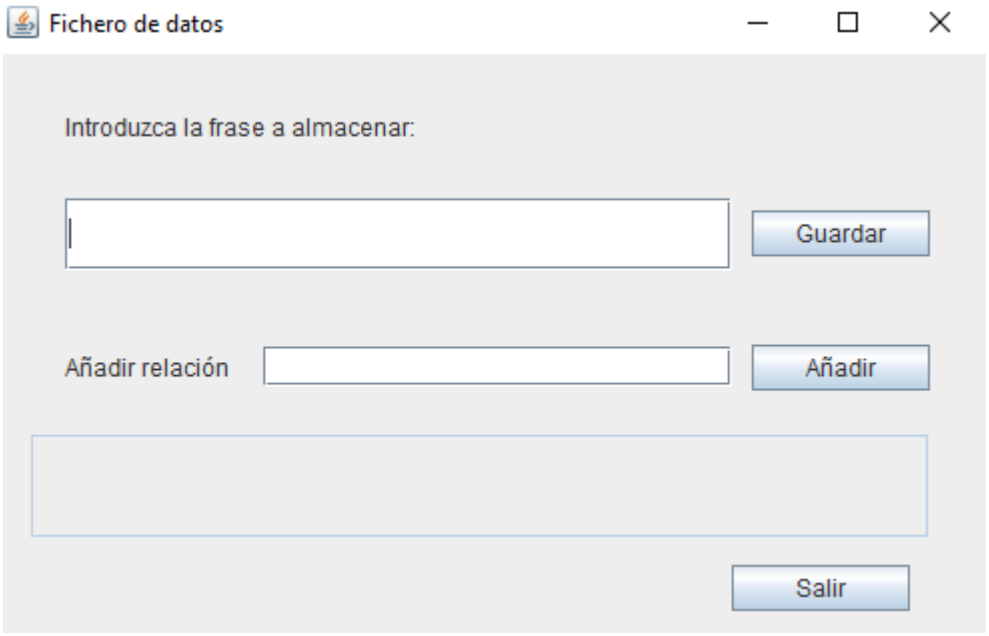

#### 4.4.2.1. Introducir datos a la Ontología.

Al seleccionar esta opción en el menú emerge una ventana como la mostrada en la figura. En ella se puede observar la petición "Introduzca la frase a almacenar", seguida de un cuadro de texto editable. En él, el usuario puede introducir una tripleta para almacenar en la ontología. Estas tripletas se van añadiendo en un fichero de texto que posteriormente será tratado. Si el usuario se equivocase y no introdujese una tripleta, se trataría de la siguiente manera. Si tiene más palabras, como será revisada por el analizador, se extraerá la tripleta pertinente y si no llegase a tres palabras sería rechazada como se explicaba anteriormente y añadida al fichero pertinente.

Además, el usuario también tiene la opción de añadir una relación, es decir, algún verbo que quiera que aprenda el robot.

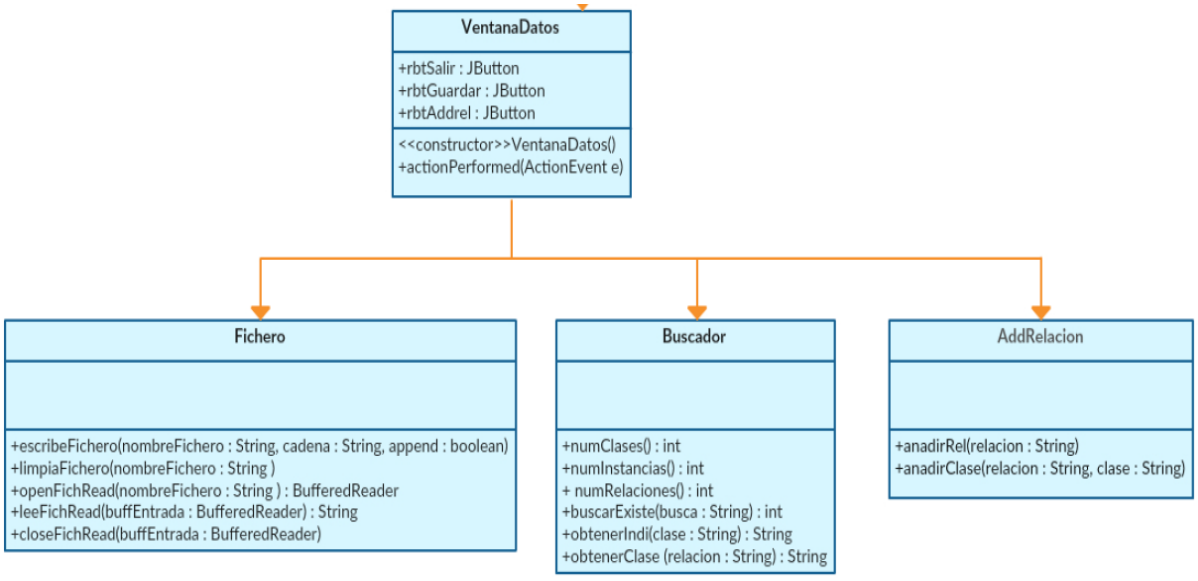
Gracias a esto, el robot no sólo aprende de manera autónoma, sino que también un usuario puede indicarle que quiere que aprenda. Esto es muy útil de cara a empatizar con el robot, puesto que en numerosas ocasiones los usuarios se refieren con una jerga propia a determinados elementos o determinadas personas. Con esta opción se pretende que el robot no sólo pueda aprender definiciones dadas por una enciclopedia, sino que también pueda aprender de mano del usuario estas expresiones.

**Figura 51. Ventana de Introducción de datos.**




A continuación se muestra el diagrama UML de esta opción del menú. Mediante la clase Fichero se produce la adición de la frase que introduce el usuario al fichero de texto correspondiente. Por otro lado las clases Buscador y AddRelación están relacionadas con la opción de añadir una nueva relación a la ontología. Mediante la clase buscador, a través de la función buscarExiste(busca:String) se comprueba si esa relación está o no en la ontología. Si ya existe no se realiza ninguna acción. Si la relación no existe, a través de la clase AddRelación se añade a la ontología, gracias a las funciones que configuran esta clase.

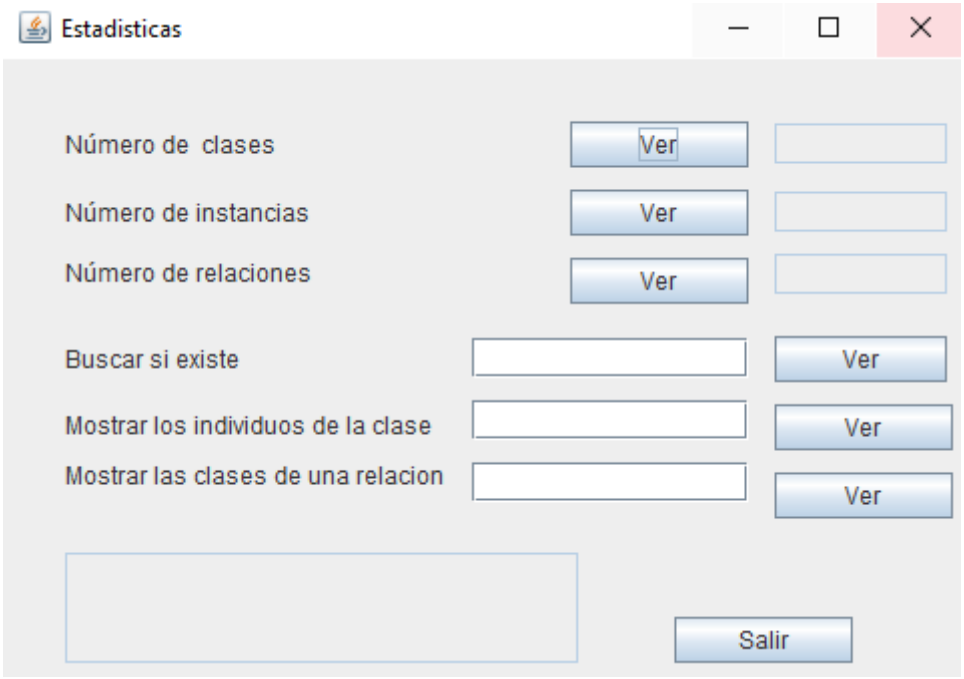
**Figura 52. Diagrama UML de la ventana de datos.**



#### 4.4.2.2. Estadísticas.

La ventana de estadísticas es la que proporciona información sobre el contenido de la ontología.

**Figura 53. Ventana "Estadísticas".**




Como se observa en la figura, a través de esta opción el usuario puede conocer el número de clases, relaciones e instancias que conforman la ontología. Así se puede ir comprobando la cantidad de elementos que pasan a engrosar la base de conocimiento.

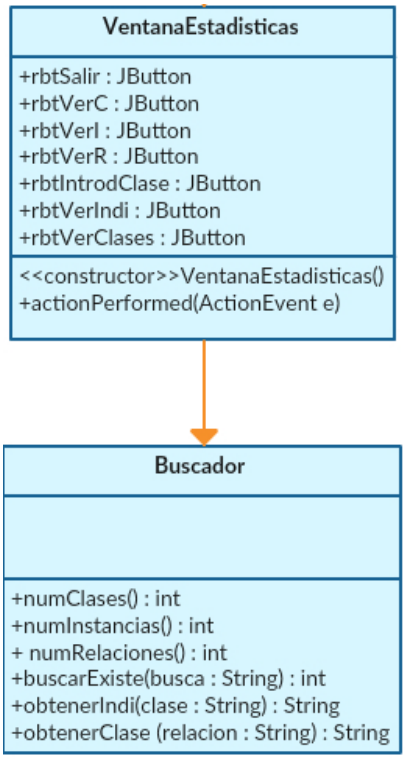
Además, el usuario puede consultar si existe algún término en la ontología y el programa, a través del cuadro de texto no editable que aparece en la parte inferior de la ventana, proporciona información sobre si se trata de una clase, una instancia o una relación. También es posible consultar los individuos que forman parte de una clase o las clases que pertenecen a una determinada relación.

Esta opción es muy útil cuando se quiere comprobar si en los momentos de inactividad aprende conceptos o de cara a comprobar si se posee una información que resulte de interés para el usuario, de modo que si no existiese éste pueda enseñársela.

El diagrama UML de la ventana "Estadísticas" es muy sencillo como se puede apreciar. Lo que se hace es consultar continuamente a la clase buscador. Esta clase está formada por varias funciones a las que se llama en los distintos casos. Las funciones numClases(), numInstancias() y numRelaciones() básicamente lo que hacen es recorrer la ontología con un bucle de lectura y según van encontrando los elementos que buscan contar cada una de ellas va incrementando un contador, lo que permite que las funciones devuelvan un resultado numérico que se muestra en el cuadro de texto que corresponda.

Las funciones obtenerIndi() y obtenerClase(), al igual que buscarExiste(), van realizando una comparación del nombre proporcionado por el usuario con los términos que contiene la ontología. Si hay coincidencia entonces se proporciona la respuesta requerida por cada función.

**Figura 54. Diagrama UML de la ventana "Estadísticas"**



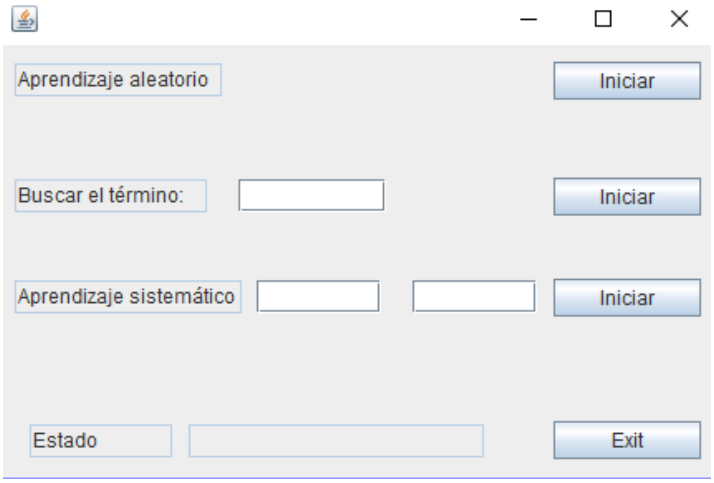

#### 4.4.2.3. Aprendizaje automático.

Esta ventana es fundamental en el proyecto. Desde ella se gestiona el aprendizaje automático, el objetivo fundamental del trabajo. Controla la búsqueda en internet de una determinada información.

Como se muestra en la figura, el usuario puede seleccionar diferentes modos de aprendizaje. Se cuenta con un aprendizaje aleatorio de contenido en Wikipedia, con un aprendizaje sobre un concepto que proporcione el usuario o un aprendizaje sistemático de la enciclopedia. Es importante recordar que para realizar un aprendizaje sistemático, hay que introducir un número precedido por la letra Q, es decir, el usuario pondría un intervalo que podría ser por ejemplo "Q3520"-"Q3650". Cada uno de estos identificadores iría en uno de los cuadros de texto.

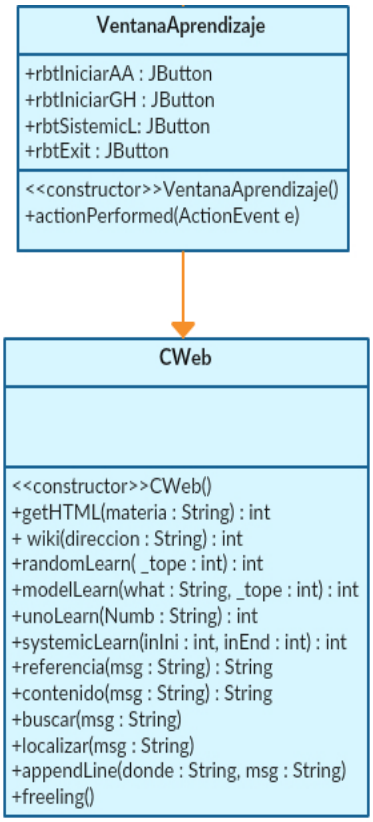
La ventana cuenta con una barra de estado que va proporcionando información al usuario de la etapa en que se encuentra, así como de cuando concluye la búsqueda de información.

**Figura 55. Ventana de aprendizaje automático.**




En el diagrama UML de la ventana se observa que únicamente se trabaja con la clase CWeb, que ha sido explicada con detalle anteriormente. No se hace mención a la clase Fichero porque esa clase es llamada por CWeb, como se explicaba en la exposición del sistema de ficheros.

**Figura 56. Diagrama UML de la ventana de aprendizaje.**



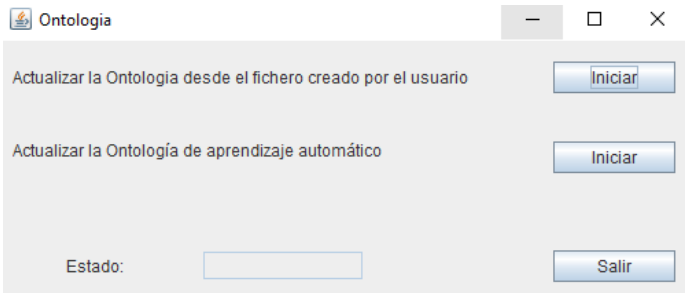

En cuanto a la velocidad de procesamiento, la búsqueda de contenido en 50 páginas (identificadas por su código "Qxxx"), se realiza en unos 90 segundos. Cierto es que algunas de estas páginas se encuentran vacías y carecen de contenido. Esto permite afirmar que el

robot que posea este mecanismo de aprendizaje, aunque posea lapsos de tiempo de inactividad muy breves puede llevar a cabo un aprendizaje extenso de conceptos.

#### 4.4.2.4. Actualización de la ontología.

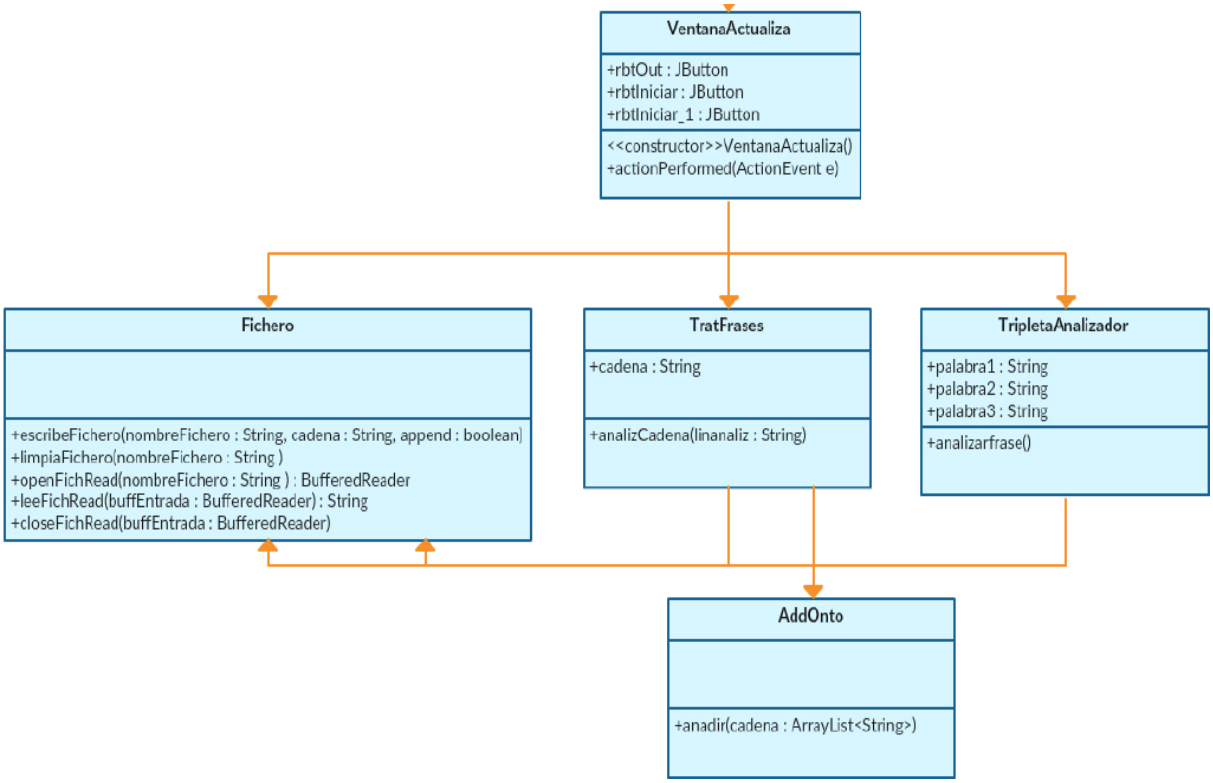
Por último, a través de esta ventana se lleva a cabo el último paso del aprendizaje, la incorporación de los conceptos a la base de conocimiento. Esta incorporación se puede realizar por el tratamiento de dos tipos de ficheros, o bien el fichero creado por el usuario con las tripletas que han ido introduciendo a través de la ventana de datos o bien mediante el fichero que se ha creado del aprendizaje automático.

**Figura 57. Ventana "Actualiza".**




La actualización de la ontología es casi instantánea, las 50 oraciones de las que se hablaba anteriormente pueden ser añadidas a la ontología en apenas 20 segundos. Esto demuestra la eficacia del aprendizaje. Además, de cara a tratar ficheros muy amplios nos da la seguridad de que no se bloquearán otras actividades. Por ello, al ver estos tiempos de procesamiento, es posible plantearse que en un futuro puedan tratarse libros o pdf extensos.

**Figura 58. Diagrama UML de la ventana actualiza**



Como se aprecia en el diagrama UML, las clases que se utilizan son Fichero, TratFrases y TripletaAnalizador, de las que ya se ha hablado en el trabajo. Además se utiliza la clase AddOnto, encargada de añadir los términos a la ontología.

AddOnto lo que hace es abrir la ontología y leerla. Para ello toma el fichero que contenga las tripletas que se quieran añadir y va comprobando si los distintos términos que conforman la tripleta están ya presentes en la ontología. Si no lo están va añadiendo cada uno según el papel que tengan en la base de conocimiento.

### 4.5. RESULTADO.

En última instancia, después de este proceso de aprendizaje, lo que se obtiene es una base de conocimiento. Esta base de conocimiento es generada gracias al API Protégé-OWL, posee una extensión .owl y muestra un aspecto como el de la figura.

**Figura 59. Documento que contiene la Ontología.**

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="relaciones:the_serpent&apos;"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.1="relaciones:"
    xmlns:relaciones="http://www.owl-ontologies.com/ontologia.owl"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Class rdf:about="relaciones:the_serpent&apos;s_venom">
    <rdfs:subClassOf>
      <owl:Class rdf:about="relaciones:es"/>
    </rdfs:subClassOf>
    <rdf:type rdf:resource="relaciones:the_serpent&apos;s_venom"/>
  </owl:Class>
  <owl:Class rdf:about="relaciones:personaje_de_one_piece">
    <rdfs:subClassOf>
      <owl:Class rdf:about="relaciones:es"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="relaciones:insecto">
    <rdfs:subClassOf>
      <owl:Class rdf:about="relaciones:es"/>
    </rdfs:subClassOf>
  </owl:Class>
```

Este tipo de documento es procesado por cualquier editor de ontologías y por numerosas herramientas, lo que hace posible su integración en distintos robots o entidades. Además gracias al programa que se ha creado, esta ontología es actualizable y ampliable. De este modo, puede ser colgada en internet para que esté accesible para cualquier usuario que desee introducir una base de conocimiento en su proyecto. Se espera que esté disponible en la página web del departamento. Su distribución será gratuita aunque si se pretende llevar algún tipo de control sobre las descargas.

Esta base de conocimiento está orientada a su uso en un diálogo humano-robot. Luego los usuarios principales a los que va dirigida es a aquellos que posean un robot social o una entidad para dicho fin.

Además, si fuese necesario, este formato puede ser convertido a cualquier otro que el usuario necesite, puesto que hay numerosas herramientas que son capaces de realizar esta conversión. Sin ir más lejos, Protégé permite la apertura de este archivo y su conversión a otro formato como pueda ser RDF.

Por otro lado, al ser el programa una aplicación programada en Java con una interfaz de usuario es instalable en numerosos dispositivos. Uno de los planteamientos es su instalación en el robot demostrador que ha sido creado en el Grupo de Control Inteligente de la UPM. Este robot cuenta con una Raspberry Pi 3, que permite el procesamiento en Java, luego la

instalación en este robot a priori no debería suponer ningún problema. Así mismo, al contar con una comunicación TCP, este programa también se podría utilizar en el robot Urbano. De este modo, los robots no solo contarían con una base de conocimiento, sino que incluirían el aprendizaje.

# 5. CONCLUSIONES

En este apartado se va a proceder a realizar un análisis de los resultados obtenidos contrastándolos con los objetivos planteados en el segundo capítulo.

- Entorno de trabajo:

  En los objetivos del proyecto se planteaba la utilización de Eclipse con Protégé. Para ello era necesario instalar un API que hiciese posible trabajar conjuntamente con estos dos programas. La integración de ambos ha sido un proceso complicado, debido a la reducida información que se encontraba sobre cómo instalarlo. Además, había que solventar numerosos errores en su instalación. Esto llevó mucho tiempo en el proyecto pero se consiguió tener integrado todo en el trabajo. Al tener esta API instalada, es posible abrir cualquier ontología desde Eclipse.

  Posteriormente se llevó a cabo la instalación de la librería Jena, un proceso más sencillo. Una vez con Jena instalado había que aprender el lenguaje utilizado para crear las ontologías, por lo que fue un proceso lento pero se consiguió realizar lo esperado.

  La elección del lenguaje Java para la programación vino un poco dad por los programas que permitían crear el programa que se requería. Puesto que era un lenguaje desconocido hasta ahora, y que en proyectos anteriores se había utilizado C++ se desconocía a priori si se podrían lograr los objetivos planteados. Sin embargo, es un lenguaje del que hay mucha información y que se ha logrado dominar para poder llevar a cabo el trabajo.

- Comunicación con el programa:

  Para este proyecto se requerían dos modos de comunicación con el programa. El primero de ellos que implementase una comunicación TCP para poder utilizar este programa en el robot Urbano, para el cual fue planteado inicialmente el trabajo. Se ha logrado crear este módulo de comunicación TCP y ha sido probado con un comprobador de comunicaciones, si bien es cierto que no ha dado tiempo a implementar el programa ni a hacer pruebas directamente en el robot Urbano.

  En segundo lugar se pedía una interfaz de usuario que fuese intuitiva. De este modo se permitiría que cualquier usuario pudiese manejar este programa aunque careciese de conocimientos informáticos. Se ha creado por tanto una interfaz de usuario, formada por varias ventanas que cumplen con este requisito. Esto está orientado a poder instalar el programa en dispositivos como el robot demostrador creado por el Grupo de control inteligente de la UPM, que se muestra en la siguiente imagen.

  Además, al ser una aplicación Java es sencilla su implementación en dispositivos como el anteriormente mencionado.

**Figura 60. Robot demostrador.**

- Aprendizaje automático:

  El objetivo más importante sin duda del proyecto era crear un bucle de aprendizaje automático, puesto que era una importante diferencia respecto a los trabajos que se habían creado hasta ahora.

  En este proyecto se ha logrado realizar un aprendizaje automático de la enciclopedia Wikipedia. Este aprendizaje es posible llevarlo a cabo de manera sistemática en la enciclopedia, mediante una búsqueda aleatoria de contenido o mediante términos que se introduzcan por motivación del usuario. Esto da pie a que se pueda programar un aprendizaje en determinados momentos en un robot o que en un momento dado estas búsquedas de términos sean por motivación del robot para aumentar la información de algún concepto que almacene en su base de conocimiento.

  Los resultados del programa creado son muy buenos, en cuanto a velocidad de procesamiento siempre que se cuente con una buena conexión a internet. Además la gestión de todo este aprendizaje es muy fácil de llevar a cabo por cualquier usuario desde la interfaz de usuario creada. Así mismo la comunicación Tcp también permite esta búsqueda de contenido.

  Se han cumplido bastante bien los requisitos que se planteaban por tanto y el resultado obtenido es interesante.

- Ontología:

  El fin último del proyecto era crear una ontología, es decir una base de conocimiento que almacenase toda la información que se pudiese obtener del aprendizaje automático.

  Pues bien, se ha creado un archivo con extensión .owl que contiene todos estos conceptos, que además es actualizable gracias al programa que se ha creado, y que por supuesto es distribuible, de modo que pueda ser utilizada por cualquier usuario que trabaje en el campo de los robots sociales.

  Así mismo, es una ontología orientada al diálogo humano-robot, de modo que se ha dispuesto la información de forma que sea fácilmente recuperable. Además por la estructura seguida sería sencillo recomponer las frases que han dado lugar a ese almacenamiento de contenido.

- Publicaciones:

  Este trabajo fin de grado ha dado lugar a una publicación en el XIII Simposio de Control Inteligente celebrado en junio de 2017 en la ciudad de Vitoria.

# 6. LÍNEAS FUTURAS

En este apartado se plantean las posibles líneas futuras de investigación y desarrollo que pueden ser realizadas como continuación de este trabajo.

- Mejoras en el aprendizaje:

  Como mejora al mecanismo de aprendizaje sería interesante dedicar un tiempo al estudio del manejo de la librería Freeling. De este modo sería posible modificar el código de la misma para obtener como salida del analizador directamente las tripletas. Esto permitiría la generación de un número superior de tripletas, lo que llevaría a la obtención de una ontología con mucho más contenido. Así mismo podrían tenerse en cuenta los adjetivos y de este modo se podría comenzar a añadir propiedades a las instancias gracias a éstos.

  Además, en esta misma línea de trabajo con el analizador, se ha visto algún ejemplo que permite el tratamiento de ficheros de mayor extensión, por ejemplo un PDF, con oraciones más complejas y relacionadas unas con otras. Por tanto sería interesante trabajar en esta línea para ver si es posible el tratamiento de este tipo de ficheros.

- Implementación:

  Se quiere realizar la instalación de este programa en el robot demostrador que acaba de ser desarrollado. Para ello, es preciso dividir el programa en dos aplicaciones. La primera para la realización de búsqueda de contenido y extracción de información de la ontología. También permitiría esta primera aplicación la consulta de las estadísticas del contenido de la base de conocimiento.

  En segundo lugar, sería necesaria otra aplicación que permitiese el aprendizaje. Esto facilitaría que en momentos de inactividad se pueda programar el bucle de aprendizaje atendiendo a diversos motivos.

  En teoría esta división es sencilla de realizar a partir del programa que se ha creado, puesto que tal y como están estructuradas las clases no debe presentar ningún problema. El mayor conflicto puede ser el reconocimiento de Protégé por parte de la Raspberri Pi 3en la que habría que instalar la aplicación en el Robot.

- Filtros de Wikipedia:

  Una de las principales características de Wikipedia es la gran cantidad de conceptos que alberga. Esto a priori es una ventaja, puesto que se cuenta con mucha y muy variada información. Sin embargo, esto ha llevado a plantearse si toda esta información es de utilidad. La respuesta sin lugar a duda es que no, no toda la información es de utilidad para generar las ontologías y por tanto es necesario limitar esta información. Para ello, se pretende implantar una serie de filtros que permitan determinar la información que es de utilidad para cada aplicación.

Algunas propuestas para estos filtros pueden ser limitar la información en función del país. De este modo se puede evitar guardar información de el Banco popular de Marruecos o de el Polo Democrático Alternativo, que es un partido político Colombiano y cuya información no es relevante para el uso que se pretende dar a la ontología.

Otra propuesta es limitar la cantidad de instancias que se puedan almacenar en la base de conocimiento sobre los conceptos. Sin embargo, esto resulta útil por ejemplo si no se quiere obtener todas las especies de hormigas que existen y simplemente se almacenan un par de ellas; pero puede ser un problema a la hora de añadir por ejemplo pintores, puesto que sólo conocer a dos de ellos sería insuficiente.

Por tanto, la opción que se considera más viable para la realización de estos filtros es diseñar un fichero que contenga información de lo que es útil o no para almacenar en la ontología. Mediante este método, sería posible modificar los parámetros y adaptarlos en función del uso o del tipo de destinatario en el que se vaya a implementar esta base de conocimiento.

# 7. BIBLIOGRAFÍA


Alcina, A., Valero, E. y Rambla. E. 2009. Terminología y sociedad del conocimiento. Bern: Peter Lang.

Barrientos, A. Peñín, L. Balaguer, C. Aracil, R. 2007. *Fundamentos de robótica*. McGraw-Hill.

Benjamin, Perakath C.; Menzel, Christopher P.; Mayer, Richard J.; Fillion, Florence; Futrell, Michael T.; deWitte, Paula S.; Lingineni, Madhavi. 1994. «IDEF5 Method Report». Knowledge Based Systems, Inc.

Breazeal, C. 2003. Toward sociable robots. *Robotics and Autonomous Systems* 42,167-175.

Domínguez, S. , Zalama,E., García-Bermejo, J.G. 2008. Arisco, un robot social con capacidad de interacción, motivación y aprendizaje. RIAI, Vol. 5, Abril 2008.

Duffy, B.R. 2003. Anthropomorphism and the social robot. *Robotics and Autonomous Systems* 42, (3-4):177-190.

Feil-Seiter, D. y Mataric, M. 2011. Socially Assistive Robotics. *IEEE Robotics & Automation Magazine*, 18(1):24-31.

Florit, C. 2008 . Metodología de aprendizaje para sistemas cognitivos en robótica. Proyecto fin de carrera. ETSII, UPM 2008.

Gramajo, E., García-Martínez, R., Rossi, B., Claverie, E., Britos, P. y Totongi, A. 1999. Una visión global del aprendizaje automático. *Rev. Inst. Tecnológico B. Aires*, Volumen 22, p. Páginas 67–75.

Gruber, T. R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* Vol. 5:199-220.

Gruber, T. R. 1995. Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology, March 1993, later published in *International Journal of Human-Computer Studies*, Vol. 43, Issues 4-5, pp. 907-928

Luna, J.A.G., Bonilla, M.L., y Torres, I.D. 2012. Metodología y métodos para la construcción de ontologías. *Scientia et technica*.

Morales, L. Ruiz, F. 2008. *Sistema Multi-Agente para la construcción semiautomática de Ontologías a partir de la web*. PFC Universidad Rovira i Virgili 2008.

Neches, R., Fikes, R.E. Finin, T., Gruber, T.R., Senator, T., Swartout, W.R. 1991 "Enabling technology for knowledge sharing". *AI Magazine*. 12(3):36-56.(2)

Noy, N y Deborah L. 2005. *Desarrollo de ontologías-101: Guía Para Crear Tu Primera Ontología*. Stanford University, Stanford,CA.

Padró, L.Analizadores Multilingües en Freeling. 2011. *Linguamática*.

Pedraza-Jiménez, R., Codina,L. y Rovira,C. 2007. Web semántica y ontologías en el procesamiento de la información documental. *El profesional de la información*, 16(6): 569-578.

Russell, S.J. y Norvig, P. 1995. *Artificial Intelligence: A modern approach*. Prentice-Hall, Inc.

Standford University. *Protégé: Open-source platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies*.

http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

http://nlp.lsi.upc.edu/freeling/node/1

https://www.gitbook.com/book/talp-upc/freeling-user-manual/details

http://www.w3.org

https://www.eclipse.org/

# 8. PRESUPUESTO

En este apartado se presenta el presupuesto del proyecto. Para su realización se han considerado dos fuentes principales de gasto. La primera es el gasto del material utilizado en él. La segunda está referida al coste de personal.

En lo que a material se refiere, fundamentalmente es de carácter informático, tanto hardware como software.

Para el coste de personal se ha tomado como modelo el salario de un becario de ingeniería industrial. Además se ha hecho una estimación por la duración en meses del proyecto.

Finalmente, se muestra el coste total del proyecto.

En la Tabla 1 se muestra el desglose por partida (material y personal) y el gasto global del proyecto.

## PRESUPUESTO DEL PROYECTO

| | UNIDAD DE MEDIDA | PRECIO UNITARIO / € | CANTIDAD TOTAL | PRECIO TOTAL / € |
|---|---|---|---|---|
| **GASTO** | | | | |
| **MATERIAL** | | | | |
| ORDENADOR PORTATIL TOSHIBA | 1 | **1.000** | 1 | **1.000** |
| DISCO DURO 1T TOSHIBA | 1 | **60** | 1 | **60** |
| ECLIPSE | 1 | 0 | 1 | 0 |
| FREELING | 1 | 0 | 1 | 0 |
| VISUAL STUDIO | 1 | 0 | 1 | 0 |
| OFFICE 2007 | 1 | 0 | 1 | 0 |
| GANTT PROJECT | 1 | 0 | 1 | 0 |
| IMPRESIÓN Y ENCUADERNACIÓN | 1 | 50 | 1 | 50 |
| **TOTAL MATERIAL** | **8** | **1.110** | **8** | **1.110** |

| | UNIDAD DE MEDIDA / MES | PRECIO UNITARIO / € | CANTIDAD TOTAL | PRECIO TOTAL |
|---|---|---|---|---|
| **COSTE DE PERSONAL** | | | | |
| COSTE DE PERSONAL | 15 | **450** | 15 | **6.750** |
| **TOTAL COSTE DE PERSONAL** | **15** | **450** | **15** | **6.750** |

| | | | | |
|---|---|---|---|---|
| **TOTAL MATERIAL** | **8** | **1.110** | **8** | **1.110** |
| **TOTAL COSTE DE PERSONAL** | **15** | **450** | **15** | **6.750** |
| **GASTO DEL PROYECTO** | **23** | **1.560** | **23** | **7.860** |

**Tabla 1. Presupuesto del proyecto.**

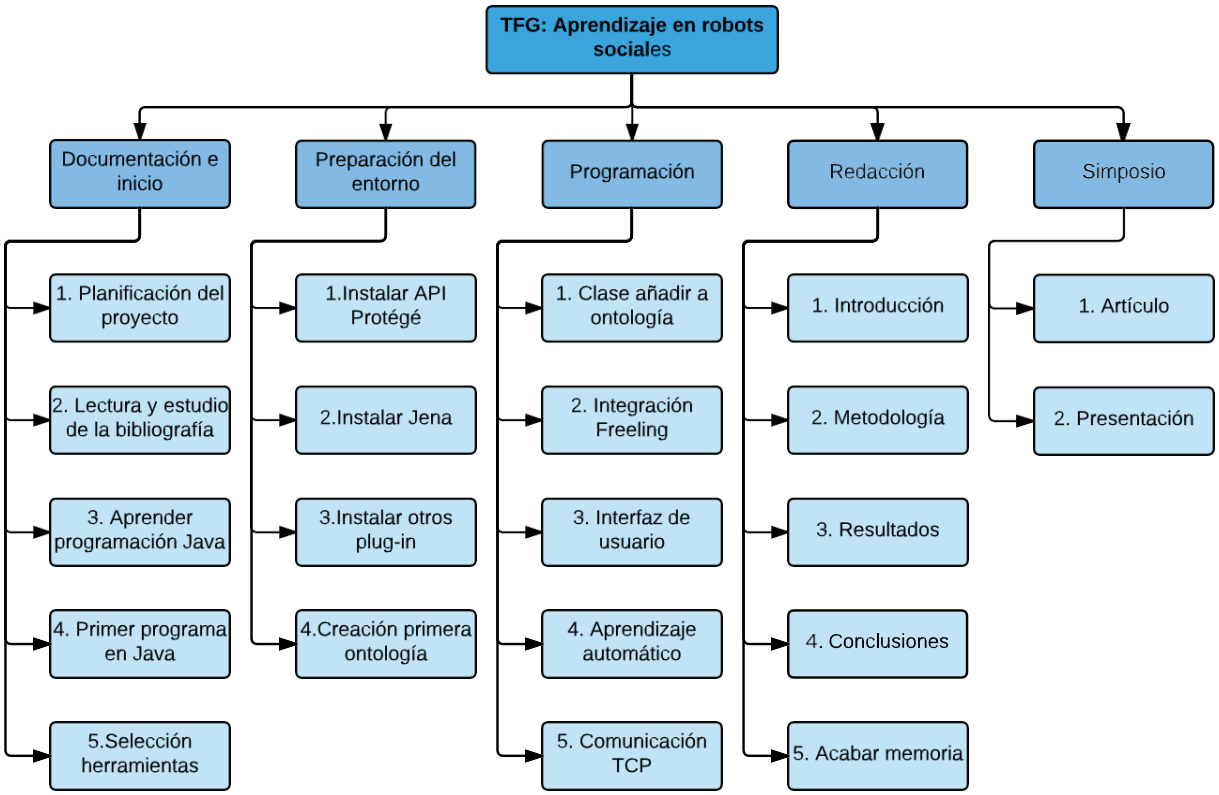
# 9. PLANIFICACIÓN TEMPORAL

En este apartado se muestra la realización de la planificación temporal del proyecto. Esta ha sido llevada a cabo mediante la elaboración del Diagrama de Gantt y de la estructura de descomposición del proyecto.

## 9.1. EDP

En este apartado se muestra la estructura de descomposición de proyecto. Como se puede observar se han agrupado las tareas a realizar en cinco bloques fundamentales. Este diagrama se ha construido con la herramienta Lucidchart.

**Figura 61. EDP**



## 9.2. DIAGRAMA DE GANTT

A partir de la estructura de descomposición del proyecto realizada en la página anterior, se desarrolló una planificación temporal para el cumplimiento de las tareas que aparecen en dicha EDP. Para la realización de este diagrama se ha recurrido a la herramienta de gestión de proyectos Gantt Project.

Se ha tenido en cuenta para la planificación del proyecto la fecha de inicio, cuando fue asignado el TFG (15 de octubre de 2015) y la fecha de finalización del mismo, cuando se realiza la entrega en Indusnet del trabajo (21 de Julio de 2017).

El calendario de trabajo es el estándar que proporciona la herramienta. Se han añadido los periodos no laborales coincidiendo con los periodos de exámenes de los cuatrimestres o épocas que no ha sido posible la realización del proyecto por diversos motivos (periodos en color gris en el diagrama).

En los últimos meses de realización del proyecto ha habido reuniones semanales los miércoles. No se han introducido en el diagrama para evitar la repetición y que no se extendiese en demasía el diagrama.

A continuación se muestra el Diagrama de Gantt correspondiente a la planificación temporal de este proyecto. En él se puede observar la división de tareas, así como la duración y la dependencia de cada una de ellas.

**Figura 62. Diagrama de Gantt.**

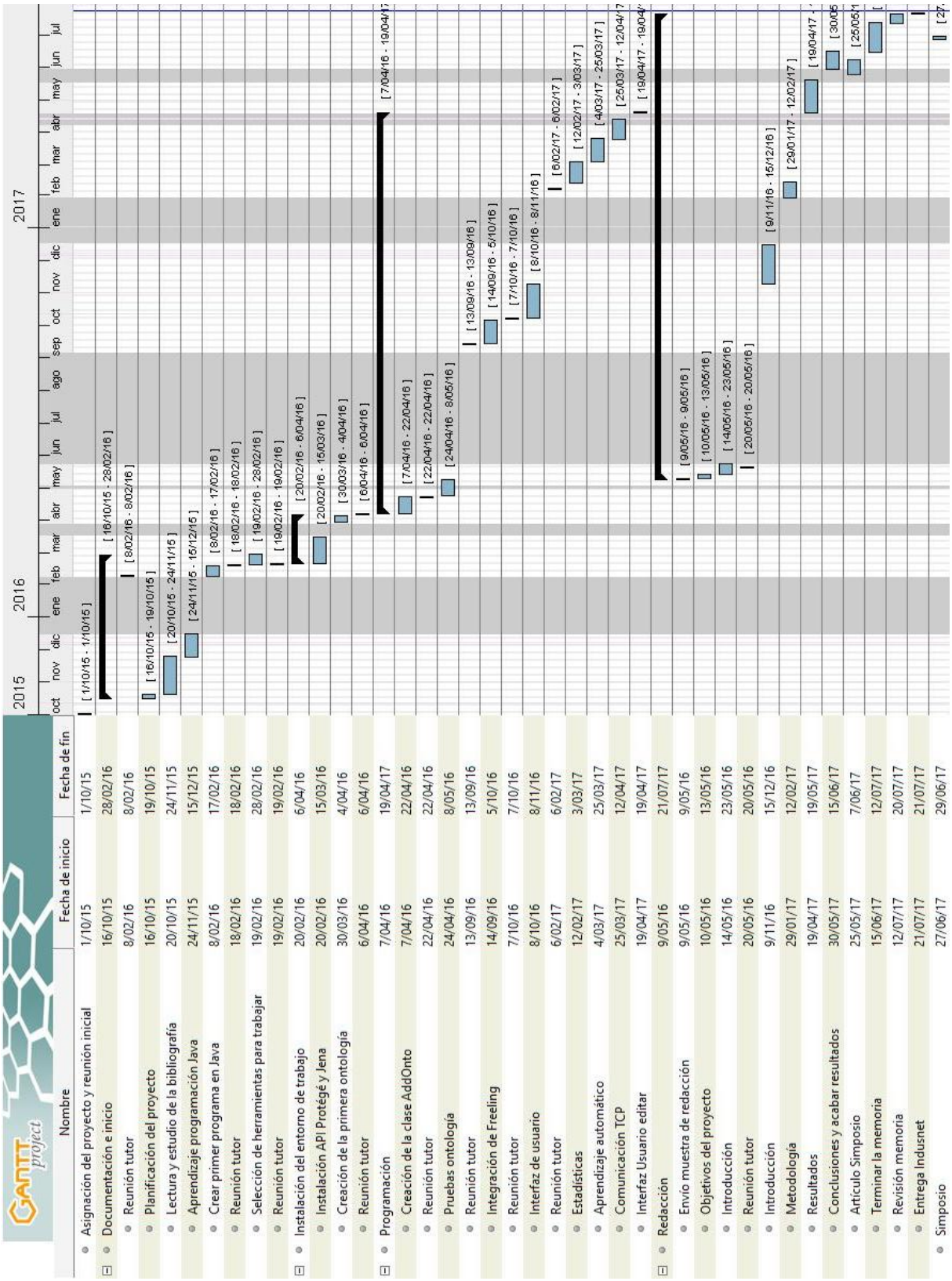


**Tabla 2. Tabla sobre el diagrama de Gantt creada por Gantt Project.**

| Nombre | Fecha de inicio | Fecha de fin |
|---|---|---|
| Asignación del proyecto y reunión inicial | 1/10/15 | 1/10/15 |
| Documentación e inicio | 16/10/15 | 28/02/16 |
| Reunión tutor | 8/02/16 | 8/02/16 |
| Planificación del proyecto | 16/10/15 | 19/10/15 |
| Lectura y estudio de la bibliografía | 20/10/15 | 24/11/15 |
| Aprendizaje programación Java | 24/11/15 | 15/12/15 |
| Crear primer programa en Java | 8/02/16 | 17/02/16 |
| Reunión tutor | 18/02/16 | 18/02/16 |
| Selección de herramientas para trabajar | 19/02/16 | 28/02/16 |
| Reunión tutor | 19/02/16 | 19/02/16 |
| Instalación del entorno de trabajo | 20/02/16 | 6/04/16 |
| Instalación API Protégé y Jena | 20/02/16 | 15/03/16 |
| Creación de la primera ontología | 30/03/16 | 4/04/16 |
| Reunión tutor | 6/04/16 | 6/04/16 |
| Programación | 7/04/16 | 19/04/17 |
| Creación de la clase AddOnto | 7/04/16 | 22/04/16 |
| Reunión tutor | 22/04/16 | 22/04/16 |
| Pruebas ontología | 24/04/16 | 8/05/16 |
| Reunión tutor | 13/09/16 | 13/09/16 |
| Integración de Freeling | 14/09/16 | 5/10/16 |
| Reunión tutor | 7/10/16 | 7/10/16 |
| Interfaz de usuario | 8/10/16 | 8/11/16 |
| Reunión tutor | 6/02/17 | 6/02/17 |
| Estadísticas | 12/02/17 | 3/03/17 |
| Aprendizaje automático | 4/03/17 | 25/03/17 |
| Comunicación TCP | 25/03/17 | 12/04/17 |
| Interfaz Usuario editar | 19/04/17 | 19/04/17 |
| Redacción | 9/05/16 | 21/07/17 |
| Envío muestra de redacción | 9/05/16 | 9/05/16 |
| Objetivos del proyecto | 10/05/16 | 13/05/16 |
| Introducción | 14/05/16 | 23/05/16 |
| Reunión tutor | 20/05/16 | 20/05/16 |
| Introducción | 9/11/16 | 15/12/16 |
| Metodología | 29/01/17 | 12/02/17 |
| Resultados | 19/04/17 | 19/05/17 |
| Conclusiones y acabar resultados | 30/05/17 | 15/06/17 |
| Artículo Simposio | 25/05/17 | 7/06/17 |
| Terminar la memoria | 15/06/17 | 12/07/17 |
| Revisión memoria | 12/07/17 | 20/07/17 |
| Entrega Indusnet | 21/07/17 | 21/07/17 |

# 10. ÍNDICE DE FIGURAS

# 11. ÍNDICE DE TABLAS

# 12. ABREVIATURAS Y ACRÓNIMOS

| | |
|---|---|
| AI | Inteligencia artificial |
| a.C | Antes de Cristo |
| KIF | Knowledge Interchange Format |
| OCML | Operational Conceptual Modeling Language |
| XML | Extensible Markup Language |
| SHOE | Simple HTLM Ontology Extensions |
| RDF | Resource Description Framework |
| W3C | World Wide Web Configuration |
| URI | Identificador Único de Recursos |
| OIL | Ontology Interchange Language |
| DAML | DARPA Agent Markup Language |
| DARPA | Defense Advance Research Projects Agency |
| OWL | Ontology Web Language |
| URL | Localizador Uniforme de Recursos |
| API | Interfaz de programación de aplicaciones |
| IP | Internet Protocol |
| TCP | Protocolo de control de transmisión |

# 13. GLOSARIO

Framework: Arquitectura software que modela las relaciones generales de las entidades del dominio y provee una estructura y metodología de trabajo, la cual utiliza o extiende las aplicaciones del dominio.

Software: Soporte lógico de un sistema informático que hace posible la realización de las tareas específicas, en contraposición a los componentes físicos a los que se denomina hardware.

Visual Studio: Entorno de desarrollo integrado para sistemas operativos Windows. Soporta numerosos lenguajes de programación.

Buffer: Espacio de la memoria en un instrumento reservado para el almacenamiento temporal de información digital mientras que está siendo procesada.

Plug-in: Aplicación o programa informático que se relaciona con otra para agregarle información nueva y generalmente específica.

Dirección IP: Es un número que identifica, de manera lógica y jerárquica, a una interfaz en red de un dispositivo que utilice el protocolo IP (Internet Protocol) que corresponde al nivel de red del modo TCP/IP.

Localhost: Servidor local. Nombre que hace referencia al ordenador o dispositivo que se está utilizando en un momento determinado. Permite utilizar ciertas herramientas TCP/IP apuntando a sí misma, es decir, en modo local, sin necesidad de conectarse a internet.

# 14. ANEXO: CÓDIGO DEL PROYECTO

## 1. Application

```
import java.io.IOException;


public class Aplication {

  public static void main(String[] args) {
        Ventanappal windowppal = new Ventanappal();
        windowppal.setVisible(true);
        //incluir funcion de tratamiento del mensaje
        //switch case
        //init (direccinip, puerto,0 indica si es un servidor o un cliente)
        //startthread crea un thread que esta pendiente de si llega un mensaje


  }

}
```

## 2. Ventanappal

```
import java.awt.*;
import javax.swing.*;


public class Ventanappal extends JFrame implements ActionListener{

  JButton btnExit;
  JRadioButton rdbtnFichero;
  JRadioButton rdbtnAprende;
  JRadioButton rdbtnEstadisticas;
  JRadioButton rdbtnAprendizaje;

  public Ventanappal() {
        initialize();
  }


  private void initialize() {

        setFont(new Font("Arial Black", Font.PLAIN, 11));
        setTitle("Aprendizaje Urbano");
        setBounds(100, 100, 404, 330); //Tamaño ventana
```

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Al salir del frame se
                                                  cierra la app
getContentPane().setLayout(null); //Posiciono yo los controles

rdbtnFichero = new JRadioButton("Introducir datos a la Ontologia");
rdbtnFichero.setFont(new Font("Arial", Font.PLAIN, 12));
rdbtnFichero.setBounds(79, 82, 195, 23);
getContentPane().add(rdbtnFichero); //Añadir al frame el radio button

rdbtnAprende = new JRadioButton("Actualizar la Ontologia");
rdbtnAprende.setFont(new Font("Arial", Font.PLAIN, 12));
rdbtnAprende.setBounds(79, 120, 168, 23);
getContentPane().add(rdbtnAprende);

rdbtnEstadisticas = new JRadioButton("Estadisticas");
rdbtnEstadisticas.setFont(new Font("Arial", Font.PLAIN, 12));
rdbtnEstadisticas.setBounds(79, 157, 109, 23);
getContentPane().add(rdbtnEstadisticas);

rdbtnAprendizaje = new JRadioButton("Aprendizaje automatico");
rdbtnAprendizaje.setFont(new Font("Arial", Font.PLAIN, 12));
rdbtnAprendizaje.setBounds(79, 191, 168, 23);
getContentPane().add(rdbtnAprendizaje);

//Agrupo los radio buttons (solo uno puede estar activo)
ButtonGroup group = new ButtonGroup();
group.add(rdbtnFichero);
group.add(rdbtnAprende);
group.add(rdbtnEstadisticas);
group.add(rdbtnAprendizaje);

 //Activamos los eventos
 rdbtnFichero.addActionListener((ActionListener) this);
 rdbtnAprende.addActionListener((ActionListener) this);
 rdbtnEstadisticas.addActionListener((ActionListener) this);
 rdbtnAprendizaje.addActionListener((ActionListener) this);

//Etiqueta de texto
JLabel lblModo = new JLabel("Seleccione el modo deseado:");
lblModo.setFont(new Font("Arial", Font.PLAIN, 12));
lblModo.setBounds(20, 24, 215, 37);
getContentPane().add(lblModo);

//Botón de salir
btnExit = new JButton("Salir");
btnExit.setFont(new Font("Arial", Font.PLAIN, 12));
btnExit.setBounds(262, 244, 89, 23);
getContentPane().add(btnExit);
```

```
        btnExit.addActionListener((ActionListener)this); //Para escuchar el evento de
                                                          pulsado del boton


    }


    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==btnExit){  //Al pulsarse el boton de salir se cierra la ventana
            System.exit(0);
        }
        else if(e.getSource()==rdbtnFichero){
            VentanaDatos ventana2 = new VentanaDatos();
            ventana2.setVisible(true);

        }
        else if(e.getSource()==rdbtnAprende){
            VentanaActualiza ventana3 = new VentanaActualiza();
            ventana3.setVisible(true);

        }
        else if(e.getSource()==rdbtnEstadisticas){
            VentanaEstadisticas ventana4 = new VentanaEstadisticas();
            ventana4.setVisible(true);

        }
        else if(e.getSource()==rdbtnAprendizaje){
            VentanaAprendizaje ventana5 = new VentanaAprendizaje();
            ventana5.setVisible(true);
        }



    }
}
```

**3. VentanaDatos**

```
import java.awt.*;
import java.io.IOException;
import javax.swing.*;

public class VentanaDatos extends JFrame implements ActionListener{

  private JPanel contentPane;
  private JTextField textField;
  JButton btnSalir;
  JButton btnGuardar;
```

```
	private JLabel lblAadirRelacin;
	JLabel lblIntroduzcaLaFrase;
	private JTextField textrelacion;
	private JButton btnAddrel;
	private JTextField textMuestra;

	public VentanaDatos() {

		setTitle("Fichero de datos");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 513, 328);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		lblIntroduzcaLaFrase = new JLabel("Introduzca la frase a almacenar:");
		lblIntroduzcaLaFrase.setFont(new Font("Arial", Font.PLAIN, 12));
		lblIntroduzcaLaFrase.setBounds(37, 25, 194, 23);
		contentPane.add(lblIntroduzcaLaFrase);

		textField = new JTextField();
		textField.setBounds(37, 72, 333, 36);
		contentPane.add(textField);
		textField.setFont(new Font("Arial", Font.PLAIN, 12));
		textField.setColumns(10);

		btnGuardar = new JButton("Guardar");
		btnGuardar.setFont(new Font("Arial", Font.PLAIN, 12));
		btnGuardar.setBounds(380, 78, 89, 23);
		contentPane.add(btnGuardar);
		btnGuardar.addActionListener((ActionListener)this);

		btnSalir = new JButton("Salir");
		btnSalir.addActionListener((ActionListener)this);
		btnSalir.setFont(new Font("Arial", Font.PLAIN, 12));
		btnSalir.setBounds(370, 255, 89, 23);
		contentPane.add(btnSalir);

		lblAadirRelacin = new JLabel("A\u00F1adir relaci\u00F3n ");
		lblAadirRelacin.setBounds(37, 149, 89, 14);
		lblAadirRelacin.setFont(new Font("Arial", Font.PLAIN, 12));
		contentPane.add(lblAadirRelacin);

		textrelacion = new JTextField();
		textrelacion.setBounds(136, 146, 234, 20);
		contentPane.add(textrelacion);
		textrelacion.setFont(new Font("Arial", Font.PLAIN, 12));
		textrelacion.setColumns(10);
```

```
		btnAddrel = new JButton("A\u00F1adir");
		btnAddrel.setBounds(380, 145, 89, 23);
		btnAddrel.setFont(new Font("Arial", Font.PLAIN, 12));
		contentPane.add(btnAddrel);
		btnAddrel.addActionListener((ActionListener)this);

		textMuestra = new JTextField();
		textMuestra.setEditable(false);
		textMuestra.setBounds(20, 190, 448, 51);
		contentPane.add(textMuestra);
		textMuestra.setFont(new Font("Arial", Font.PLAIN, 12));
		textMuestra.setColumns(10);

	}

	public void actionPerformed(ActionEvent e) {
		if(e.getSource()==btnGuardar){
			//System.out.println("Dentro del boton");
			String cadena;
			cadena=textField.getText();
			System.out.println(cadena);
			Fichero f = new Fichero();

	f.escribefichero("C:/progproyecto/protege/frases.txt",cadena.toLowerCase(), true);
			textField.setText(null);
			textMuestra.setText("Frase almacenada para su tratamiento");
		}
		else if(e.getSource()==btnSalir){
			dispose();                                          //Me cierra solo la
                                                                ventana secundaria
		}
		else if(e.getSource()==btnAddrel){
			String cadena;
			cadena=textrelacion.getText();
			int existe;
			String respuesta;
			Buscador c11 = new Buscador();
			existe= c11.buscarexiste(cadena);
			System.out.println("recibimos de existe :"+existe);
			if(existe!=1){
				addrelacion ad = new addrelacion();
				try {
					ad.anadirrel(cadena);
				} catch (IOException e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}
				respuesta="Hemos añadido la nueva relacion";
				System.out.println("Hemos añadido la nueva relacion");
			}
```

```
                    else{
                            respuesta="Ya existia la relacion";
                            System.out.println("Ya existia anteriormente");
                    }
                    textrelacion.setText(null);
                    textMuestra.setText(respuesta);
            }

    }
}
```

**4. VentanaEstadísticas**

```
import javax.swing.*;
import java.awt.*;

public class VentanaEstadisticas extends JFrame implements ActionListener{

  private JPanel contentPane;
  JButton btnSalir;
  private JTextField textClase;
  private JTextField textInstancias;
  private JTextField textBusca;
  private JTextField textMuestra;
  JLabel lblNmeroDeClases;
  JLabel lblNmeroDeInstancias;
  JLabel lblverinstancias;
  JLabel lblNmeroDeRelaciones;
  JButton btnVerC;
  JButton btnVerI;
  JButton btnIntrodclase;
  JButton btnVerR;
  private JTextField textRelaciones;
  private JLabel lblMostrarLosIndividuos;
  private JTextField textVerIndi;
  private JButton btnVerIndi;
  private JLabel lblMostrarLasClases;
  private JTextField textclases;
  private JButton btnVerClases;


  public VentanaEstadisticas() {
          setTitle("Estadisticas");
          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          setBounds(100, 100, 497, 351);
          contentPane = new JPanel();
          contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
          setContentPane(contentPane);
```

```
contentPane.setLayout(null);

btnSalir = new JButton("Salir");
btnSalir.setFont(new Font("Arial", Font.PLAIN, 12));
btnSalir.setBounds(335, 278, 89, 23);
contentPane.add(btnSalir);
btnSalir.addActionListener((ActionListener)this);

lblNmeroDeClases = new JLabel("N\u00FAmero de  clases");
lblNmeroDeClases.setBounds(31, 31, 203, 23);
lblNmeroDeClases.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(lblNmeroDeClases);

btnVerC = new JButton("Ver");
btnVerC.setBounds(283, 31, 89, 23);
contentPane.add(btnVerC);
btnVerC.setFont(new Font("Arial", Font.PLAIN, 12));
btnVerC.addActionListener((ActionListener)this);

textClase = new JTextField();
textClase.setEditable(false);
textClase.setBounds(385, 32, 86, 20);
contentPane.add(textClase);
textClase.setFont(new Font("Arial", Font.PLAIN, 12));
textClase.setColumns(10);

lblNmeroDeInstancias = new JLabel("N\u00FAmero de instancias");
lblNmeroDeInstancias.setBounds(31, 65, 203, 23);
lblNmeroDeInstancias.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(lblNmeroDeInstancias);

btnVerI = new JButton("Ver");
btnVerI.setBounds(283, 65, 89, 23);
contentPane.add(btnVerI);
btnVerI.setFont(new Font("Arial", Font.PLAIN, 12));
btnVerI.addActionListener((ActionListener)this);

textInstancias = new JTextField();
textInstancias.setEditable(false);
textInstancias.setBounds(385, 66, 86, 20);
textInstancias.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(textInstancias);
textInstancias.setColumns(10);

lblverinstancias = new JLabel("Buscar si existe");
lblverinstancias.setBounds(31, 135, 149, 29);
lblverinstancias.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(lblverinstancias);

textBusca = new JTextField();
```

```
textBusca.setBounds(234, 139, 138, 20);
textBusca.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(textBusca);
textBusca.setColumns(10);

btnIntrodclase = new JButton("Ver");
btnIntrodclase.setBounds(385, 138, 86, 23);
btnIntrodclase.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(btnIntrodclase);
btnIntrodclase.addActionListener((ActionListener)this);

textMuestra = new JTextField();
textMuestra.setEditable(false);
textMuestra.setBounds(31, 246, 256, 55);
textMuestra.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(textMuestra);
textMuestra.setColumns(10);

lblNmeroDeRelaciones = new JLabel("N\u00FAmero de relaciones");
lblNmeroDeRelaciones.setBounds(31, 99, 203, 14);
lblNmeroDeRelaciones.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(lblNmeroDeRelaciones);

btnVerR = new JButton("Ver");
btnVerR.setBounds(283, 99, 89, 23);
btnVerR.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(btnVerR);
btnVerR.addActionListener((ActionListener)this);

textRelaciones = new JTextField();
textRelaciones.setEditable(false);
textRelaciones.setBounds(385, 97, 86, 20);
textRelaciones.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(textRelaciones);
textRelaciones.setColumns(10);

lblMostrarLosIndividuos = new JLabel("Mostrar los individuos de la clase");
lblMostrarLosIndividuos.setBounds(31, 175, 203, 14);
lblMostrarLosIndividuos.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(lblMostrarLosIndividuos);

textVerIndi = new JTextField();
textVerIndi.setBounds(234, 170, 138, 20);
textVerIndi.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(textVerIndi);
textVerIndi.setColumns(10);


btnVerIndi = new JButton("Ver");
btnVerIndi.setBounds(385, 172, 89, 23);
```

```
                contentPane.add(btnVerIndi);
                btnVerIndi.setFont(new Font("Arial", Font.PLAIN, 12));
                btnVerIndi.addActionListener((ActionListener)this);

                lblMostrarLasClases = new JLabel("Mostrar las clases de una relacion");
                lblMostrarLasClases.setBounds(31, 200, 203, 14);
                lblMostrarLasClases.setFont(new Font("Arial", Font.PLAIN, 12));
                contentPane.add(lblMostrarLasClases);

                textclases = new JTextField();
                textclases.setBounds(234, 201, 138, 20);
                textclases.setFont(new Font("Arial", Font.PLAIN, 12));
                contentPane.add(textclases);
                textclases.setColumns(10);

                btnVerClases = new JButton("Ver");
                btnVerClases.setBounds(385, 206, 89, 23);
                btnVerClases.setFont(new Font("Arial", Font.PLAIN, 12));
                contentPane.add(btnVerClases);
                btnVerClases.addActionListener((ActionListener)this);

        }
       /* Buscador c2 = new Buscador();
                                int nclases;
                                int ncla;
                                int ninstancias;
                                int nrelaciones;
                                ncla= c2.numclases();
                                System.out.println("numero clases final: "  +ncla );
                                ninstancias=c2.numinstancias();
                                System.out.println("numero instancias final: "  +ninstancias );
                                nrelaciones=c2.numrelaciones();
                                System.out.println("numero relaciones final: "  +nrelaciones );
                                nclases=ncla-nrelaciones-1;*/
        @Override
        public void actionPerformed(ActionEvent e) {
                if(e.getSource()==btnSalir){
                        dispose();                                              //Me cierra solo la
ventana secundaria
                }
                else if(e.getSource()==btnVerC){
                        Buscador buscarclases = new Buscador();
                        int numC;
                        int numRel;
                        int total;
                        /*System.out.print("Llamamos a la funcion para buscar el numero de
clases");
                        numC= buscarclases.numclases();
                        System.out.print("Numero total " +numC);
                        String strI = "" + numC;
```

```
                        System.out.print("print strI " +strI);
                        textClase.setText(strI); Este código funciona*/

                        numC= buscarclases.numclases();
                        System.out.println("numero clases final: "  +numC );
                        numRel=buscarclases.numrelaciones();
                        System.out.println("numero relaciones final: "  +numRel );
                        total=numC-numRel-1;
                        String strI = "" + total;
                        System.out.print("print strI " +strI);
                        textClase.setText(strI);
                }
                else if(e.getSource()==btnVerI){
                        Buscador buscarinstancias = new Buscador();
                        int numI;
                        System.out.print("Llamamos a la funcion para buscar el numero de
instancias");
                        numI= buscarinstancias.numinstancias();
                        String strIns = "" + numI;
                        System.out.print("print strI " +strIns);
                        textInstancias.setText(strIns);
                }
                else if(e.getSource()==btnVerR){
                        Buscador buscarclases = new Buscador();
                        int numR;
                        System.out.print("Llamamos a la funcion para buscar el numero de
relaciones");
                        numR= buscarclases.numrelaciones();
                        String strRel = "" + numR;
                        System.out.println("numero relaciones final: "  +strRel );
                        textRelaciones.setText(strRel);
                }
                else if (e.getSource()==btnIntrodclase){
                        Buscador c3 = new Buscador();
                        String buscaclase;
                        buscaclase = textBusca.getText();
                        int solucion;
                        String respuesta = null;
                        solucion=c3.buscarexiste(buscaclase);
                        if(solucion==0){
                                respuesta="No existe en la ontologia";
                        }
                        else if(solucion==1){
                                respuesta="Es una relacion";
                        }
                        else if(solucion==2){
                                respuesta="Es una clase";
                        }
                        else if(solucion==3){
                                respuesta="Es una instancia";
```

```
                }
                System.out.println("respuesta: "  +respuesta );
                textMuestra.setText(respuesta);
        }
        else if(e.getSource()==btnVerIndi){
                Buscador c4 = new Buscador();
                String clase = textVerIndi.getText();
                String responde = null;
                int sol;
                sol=c4.buscarexiste(clase);
                if(sol==2){ //tenemos que devolver los individuos
                        responde=c4.obtenerindi(clase);
                        System.out.println("responde: "  +responde );
                }
                else if (sol==0){
                        responde="Clase no almacenada en la ontologia";
                        System.out.println(responde);
                }
                else if (sol==1){
                        responde="Es una relacion no una clase";
                        System.out.println(responde);
                }
                else if (sol==3){
                        responde="Es una instancia no una clase";
                        System.out.println(responde);
                }
                textMuestra.setText(responde);
        }
        else if(e.getSource()==btnVerClases){
                Buscador c5 = new Buscador();
                String relacion= textclases.getText();
                String responde = null;
                int sol;
                sol=c5.buscarexiste(relacion);
                if(sol==1){ //tenemos que devolver las clases
                        responde=c5.obtenerclase(relacion);
                        System.out.println("responde: "  +responde );
                }
                else if (sol==0){
                        responde="Relacion no almacenada en la ontologia";
                        System.out.println(responde);
                }
                else if (sol==2){
                        responde="Es una clase no una relacion";
                        System.out.println(responde);
                }
                else if (sol==3){
                        responde="Es una instancia no una relacion";
                        System.out.println(responde);
                }
```

```
                    textMuestra.setText(responde);
                }


    }
}
```

**5. VentanaAprendizaje**

```
import java.awt.*;
import javax.swing.*;
import java.io.IOException;

public class VentanaAprendizaje extends JFrame implements ActionListener{

  private JPanel contentPane;
  private JTextField txtAprendizajeAleatorio;
  private JTextField txtBuscarElTrmino;
  private JTextField untermino;
  long start = System.currentTimeMillis();
  CWeb web = new CWeb();
  int n=0;
  JButton btnIniciarAA;
  JButton btnexit;
  JButton btnIniciarGH;
  JButton btnSistemicL;
  private JTextField texthasta;
  private JTextField txtSistemiclearn;
  private JTextField textdesde;
  private JTextField txtEstado;
  private JTextField textsituacion;


  public VentanaAprendizaje() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        txtAprendizajeAleatorio = new JTextField();
        txtAprendizajeAleatorio.setEditable(false);
        txtAprendizajeAleatorio.setText("Aprendizaje aleatorio");
        txtAprendizajeAleatorio.setBounds(10, 11, 125, 20);
        txtAprendizajeAleatorio.setFont(new Font("Arial", Font.PLAIN, 12));
        contentPane.add(txtAprendizajeAleatorio);
        txtAprendizajeAleatorio.setColumns(10);
```

```
btnIniciarAA = new JButton("Iniciar");
btnIniciarAA.setBounds(335, 10, 89, 23);
contentPane.add(btnIniciarAA);
btnIniciarAA.setFont(new Font("Arial", Font.PLAIN, 12));
btnIniciarAA.addActionListener((ActionListener)this);

txtBuscarElTrmino = new JTextField();
txtBuscarElTrmino.setEditable(false);
txtBuscarElTrmino.setText("Buscar el t\u00E9rmino:");
txtBuscarElTrmino.setBounds(10, 81, 116, 20);
contentPane.add(txtBuscarElTrmino);
txtBuscarElTrmino.setColumns(10);
txtBuscarElTrmino.setFont(new Font("Arial", Font.PLAIN, 12));

untermino = new JTextField();
untermino.setBounds(145, 81, 89, 20);
untermino.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(untermino);
untermino.setColumns(10);

btnIniciarGH = new JButton("Iniciar");
btnIniciarGH.setBounds(335, 80, 89, 23);
contentPane.add(btnIniciarGH);
btnIniciarGH.setFont(new Font("Arial", Font.PLAIN, 12));
btnIniciarGH.addActionListener((ActionListener)this);

btnexit = new JButton("Exit");
btnexit.setBounds(335, 227, 89, 23);
contentPane.add(btnexit);
btnexit.setFont(new Font("Arial", Font.PLAIN, 12));
btnexit.addActionListener((ActionListener)this);

texthasta = new JTextField();
texthasta.setBounds(250, 142, 75, 20);
texthasta.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(texthasta);
texthasta.setColumns(10);

txtSistemiclearn = new JTextField();
txtSistemiclearn.setEditable(false);
txtSistemiclearn.setText("Aprendizaje sistem\u00E1tico");
txtSistemiclearn.setBounds(10, 142, 137, 20);
txtSistemiclearn.setFont(new Font("Arial", Font.PLAIN, 12));
contentPane.add(txtSistemiclearn);
txtSistemiclearn.setColumns(10);

textdesde = new JTextField();
textdesde.setBounds(156, 142, 75, 20);
textdesde.setFont(new Font("Arial", Font.PLAIN, 12));
```

```
		contentPane.add(textdesde);
		textdesde.setColumns(10);

		btnSistemicL = new JButton("Iniciar");
		btnSistemicL.setBounds(335, 141, 89, 23);
		contentPane.add(btnSistemicL);
		btnSistemicL.setFont(new Font("Arial", Font.PLAIN, 12));
		btnSistemicL.addActionListener((ActionListener)this);

		txtEstado = new JTextField();
		txtEstado.setEditable(false);
		txtEstado.setText("Estado");
		txtEstado.setBounds(19, 229, 86, 20);
		contentPane.add(txtEstado);
		txtEstado.setColumns(10);
		txtEstado.setFont(new Font("Arial", Font.PLAIN, 12));

		textsituacion = new JTextField();
		textsituacion.setEditable(false);
		textsituacion.setBounds(115, 229, 178, 20);
		contentPane.add(textsituacion);
		textsituacion.setColumns(10);
		textsituacion.setFont(new Font("Arial", Font.PLAIN, 12));


	}
	public void actionPerformed(ActionEvent e) {
		String estadoProceso;
		estadoProceso="Iniciado";
		if(e.getSource()==btnIniciarAA){
			textsituacion.setText(estadoProceso);
			try {
				n=web.RandomLearn(50);
			} catch (IOException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
			try {
				web.Freeling();
			} catch (IOException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
			// calcular tiempo transcurrido
			long end = System.currentTimeMillis();
			long res = end - start;
			System.out.println("Segundos: "+res/1000);
			estadoProceso="Terminado";
			textsituacion.setText(estadoProceso);
			}//Fin del if btnIniciarAA
```

```
if(e.getSource()==btnIniciarGH){
	String cadena;
	textsituacion.setText(estadoProceso);
	cadena=untermino.getText();
	System.out.println(cadena);
	try {
		n=web.getHTML(cadena);
	} catch (IOException e1) {
		// TODO Auto-generated catch block
		e1.printStackTrace();
	}
	try {
		web.Freeling();
	} catch (IOException e1) {
		// TODO Auto-generated catch block
		e1.printStackTrace();
	}
	// calcular tiempo transcurrido
	long end = System.currentTimeMillis();
	long res = end - start;
	System.out.println("Segundos: "+res/1000);
	estadoProceso="Terminado";
	textsituacion.setText(estadoProceso);
}//Fin del btnIniciarUL
if(e.getSource()==btnSistemicL){
	int cadenainicio;
	int cadenafin;
	cadenainicio=textdesde.getX();
	cadenafin=texthasta.getX();
	textsituacion.setText(estadoProceso);
	try {
		n=web.SystemicLearn(cadenainicio, cadenafin);
	} catch (IOException e1) {
		// TODO Auto-generated catch block
		e1.printStackTrace();
	}
	try {
		web.Freeling();
	} catch (IOException e1) {
		// TODO Auto-generated catch block
		e1.printStackTrace();
	}
	// calcular tiempo transcurrido
	long end = System.currentTimeMillis();
	long res = end - start;
	System.out.println("Segundos: "+res/1000);
	estadoProceso="Terminado";
	textsituacion.setText(estadoProceso);
	}//Fin del if btnIniciarSystemic
```

```
                if(e.getSource()==btnexit){  //Al pulsarse el boton de salir se cierra la ventana
                        dispose();
                }//Fin del if btnexit
                }
        }
```

**6. VentanaActualiza**

```
        import java.*;
        import javax.swing.*;

        public class VentanaActualiza extends JFrame implements ActionListener {

          private JPanel contentPane;
          private JTextField textEstado;
          JButton btnOut;
          JButton btnIniciar;
          JButton btnIniciar_1;
          JLabel lblActualizarLaOntologia;
          private JLabel lblEstado;
          JLabel lblActualizarLaOntologa;


          public VentanaActualiza() {
                setTitle("Ontologia");
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 519, 224);
                contentPane = new JPanel();
                contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
                setContentPane(contentPane);
                contentPane.setLayout(null);

                lblActualizarLaOntologia = new JLabel("Actualizar la Ontologia desde el
fichero creado por el usuario");
                lblActualizarLaOntologia.setFont(new Font("Arial", Font.PLAIN, 12));
                lblActualizarLaOntologia.setBounds(10, 11, 412, 30);
                contentPane.add(lblActualizarLaOntologia);

                btnIniciar = new JButton("Iniciar");
                btnIniciar.setFont(new Font("Arial", Font.PLAIN, 12));
                btnIniciar.setBounds(404, 15, 89, 23);
                contentPane.add(btnIniciar);
                btnIniciar.addActionListener((ActionListener)this);

                btnOut = new JButton("Salir");
                btnOut.addActionListener((ActionListener)this);
                btnOut.setBounds(404, 152, 89, 23);
                btnOut.setFont(new Font("Arial", Font.PLAIN, 12));
```

```
                contentPane.add(btnOut);

                textEstado = new JTextField();
                textEstado.setFont(new Font("Arial", Font.PLAIN, 12));
                textEstado.setEditable(false);
                textEstado.setBounds(149, 153, 116, 20);
                contentPane.add(textEstado);
                textEstado.setColumns(10);

                lblEstado = new JLabel("Estado:");
                lblEstado.setFont(new Font("Arial", Font.PLAIN, 12));
                lblEstado.setBounds(49, 152, 49, 23);
                contentPane.add(lblEstado);

                lblActualizarLaOntologa = new JLabel("Actualizar la Ontolog\u00EDa de aprendizaje autom\u00E1tico");
                lblActualizarLaOntologa.setBounds(10, 68, 328, 23);
                lblActualizarLaOntologa.setFont(new Font("Arial", Font.PLAIN, 12));
                contentPane.add(lblActualizarLaOntologa);

                btnIniciar_1 = new JButton("Iniciar");
                btnIniciar_1.setBounds(404, 73, 89, 23);
                btnIniciar_1.setFont(new Font("Arial", Font.PLAIN, 12));
                contentPane.add(btnIniciar_1);
                btnIniciar_1.addActionListener((ActionListener)this);
        }

       public void actionPerformed(ActionEvent e) {
         if(e.getSource()==btnOut){  //Al pulsarse el boton de salir se cierra la ventana
                dispose();
         }
         else if(e.getSource()==btnIniciar){
                String estadoProceso;
                BufferedReader buff;
                String linea;
                estadoProceso= "Iniciado";
                textEstado.setText(estadoProceso);
                Fichero g = new Fichero();
                buff= g.openFichRead("C:/progproyecto/protege/frases.txt"); //Lo almaceno en un objeto tipo buffer que es lo que me devuelve ese metodo de openFichRead
                linea = g.leeFichRead(buff); //Almaceno la primera linea que he leido del fichero
                System.out.println("linea leida ventana actualiza :"+linea);//Print consola de la linea leida
                while (linea != null){
                System.out.println("dentro del while :"+linea);
                //System.out.println("linea leidafn :"+lineaFi);
                 if (linea !=null){
                        // tipoCad=analizCadena(lineaFi);
                         System.out.println("linea leida :"+linea);
```

```
                        TratFrases frase = new TratFrases();
                                try {
                                        frase.analizCadena(linea);
                                    } catch (IOException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                    }
                                System.out.println("vuelvo de analizar cadena");


                    }//fin de if
                linea = g.leeFichRead(buff);
                System.out.println("leo siguiente linea"+linea);
                    }//fin de while
                   g.closeFichRead(buff);
                   estadoProceso= "Terminado";
                   g.limpiaFichero("C:/progproyecto/protege/frases.txt");
                   textEstado.setText(estadoProceso);

        }
        else if (e.getSource()==btnIniciar_1){
                String estadoProceso;
                BufferedReader buff;
                String linea;
                estadoProceso= "Iniciado";
                textEstado.setText(estadoProceso);
                TripletaAnalizador construirFichero = new TripletaAnalizador();
                construirFichero.analizarfrase();
                Fichero g = new Fichero();
                buff= g.openFichRead("C:/progproyecto/protege/tripletas.txt"); //Lo almaceno
en un objeto tipo buffer que es lo que me devuelve ese metodo de openFichRead
                linea = g.leeFichRead(buff); //Almaceno la primera linea que he leido del
fichero
                System.out.println("linea leida ventana actualiza :"+linea);//Print consola de la
linea leida
                while (linea != null){
                System.out.println("dentro del while :"+linea);
                //System.out.println("linea leidafn :"+lineaFi);
                 if (linea !=null){
                        // tipoCad=analizCadena(lineaFi);
                         System.out.println("linea leida :"+linea);
                         TratFrases frase = new TratFrases();
                                try {
                                        frase.analizCadena(linea);
                                    } catch (IOException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                    }
                                System.out.println("vuelvo de analizar cadena");
```

```
            }//fin de if
          linea = g.leeFichRead(buff);
          System.out.println("leo siguiente linea"+linea);
            }//fin de while
          g.closeFichRead(buff);
          estadoProceso= "Terminado";
          //g.limpiaFichero("C:/progproyecto/protege/tripletas.txt");
          //g.limpiaFichero("C:/progproyecto/protege/input.txt");
          textEstado.setText(estadoProceso);

    }
  }
  }
```

## 7. Comunicacion

```
  import java.net.URLDecoder;
  import javax.swing.*;
  import java.io.IOException;

  public class Comunicacion {
    public static void main(String[] args) {
          CSocketNode server;
          CWeb web;
          int n=0;



                public static void TcpMessage(int code, String msg) throws
IOException{
                System.out.println("En TcpMessage "+code+" "+ msg);
                long start = System.currentTimeMillis();
                CWeb web = new CWeb();
                int n=0;
                String respuesta = null;
                switch (code){
          /*
                case 1: //Pedir estadisticas Respuesta: 2-nº clases- nº objetos -
relaciones
                      Buscador c2 = new Buscador();
                      int nclases;
                      int ncla;
                      int ninstancias;
                      int nrelaciones;
                      ncla= c2.numclases();
                      System.out.println("numero clases final: "  +ncla );
                      ninstancias=c2.numinstancias();
```

```
                    System.out.println("numero instancias final: "  +ninstancias );
                    nrelaciones=c2.numrelaciones();
                    System.out.println("numero relaciones final: "  +nrelaciones );
                    nclases=ncla-nrelaciones-1;

                    respuesta="2"+" "+nclases+" "+ninstancias+" "+nrelaciones;
                    System.out.println("respuesta: "  +respuesta );
                    break;
                case 2: //Me da una clase para buscar y debo decir si lo conozco o no
                    Buscador c3 = new Buscador();
                    String buscaclase="pintor";

                    if(buscaclase==msg){
                        System.out.println("Si se produce igualdad");
                    }
                    System.out.println("clase a buscar: "  +msg );
                    buscaclase="pintor";
                    System.out.println("clase que mando: "  +buscaclase );
                    int solucion;
                    solucion=c3.buscarexiste(buscaclase);
                    if(solucion==0){
                        respuesta="3"+" "+"No existe en la ontologia";
                    }
                    else if(solucion==1){
                        respuesta="3"+" "+"Es una relacion";
                    }
                    else if(solucion==2){
                        respuesta="3"+" "+"Es una clase";
                    }
                    else if(solucion==3){
                        respuesta="3"+" "+"Es una instancia";
                    }
                    System.out.println("respuesta: "  +respuesta );
                    break;

                case 3: //Me da unos caracteres para buscar, devuelvo los individuos de
una clase:
                    //respondo con 4-nº de individuos-nombre de los individuos
                    Buscador c4 = new Buscador();
                    String clase = msg;
                    String responde;
                    String respu;
                    int sol;
                    sol=c4.buscarexiste(clase);
                    if(sol==2){ //tenemos que devolver los individuos
                        responde=c4.obtenerindi(clase);
                        respu=4+" "+responde;
                        System.out.println("responde: "  +respu );
                    }
                    else{
```

```
                    System.out.println("Clase no almacenada en la ontologia");
                }

                break;

            case 5://Definir una clase pasando nombre y clase superior(es decir, relacion de la que depende)
                String nomclase = "piano";
                String nomrelacion = "toca";
                int existerel=0;
                int existecla=0;
                Buscador c7 = new Buscador();
                addrelacion add = new addrelacion();
                existerel= c7.buscarexiste(nomrelacion); //vemos si existe la relacion en la ontologia
                if(existerel==1){                        //si existe entonces
                    add.anadirclase(nomrelacion, nomclase);

                }
                if(existerel!=1){           //como la relacion no existe la añadimos a la ontologia
                    add.anadirrel(nomrelacion);
                    System.out.println("Hemos añadido la nueva relacion");
                    //a continuacion procedemos a añadir la clase
                    add.anadirclase(nomrelacion, nomclase);
                }
                break;

            case 6://definir relación OK!
                String addrelacion = "posee";
                int existe;
                Buscador c11 = new Buscador();
                existe= c11.buscarexiste(addrelacion);
                System.out.println("recibimos de existe :"+existe);
                if(existe!=1){
                    addrelacion ad = new addrelacion();
                    ad.anadirrel(addrelacion);
                    System.out.println("Hemos añadido la nueva relacion");
                }
                else{
                    System.out.println("Ya existia anteriormente");
                }
                break;

            case 7: //procesar un fichero OK!
                BufferedReader buff;
                String linea;
                Fichero g = new Fichero();
```

```
                    buff=     g.openFichRead("C:/progproyecto/protege/frases.txt");
//Lo almaceno en un objeto tipo buffer que es lo que me devuelve ese metodo de
openFichRead
                    linea = g.leeFichRead(buff); //Almaceno la primera linea que he
leido del fichero
                    System.out.println("linea leida comunicacion :"+linea);//Print
consola de la linea leida
                    while (linea != null){
                    System.out.println("dentro del while :"+linea);
                     if (linea !=null){
                              System.out.println("linea leida :"+linea);
                              TratFrases frase = new TratFrases();
                                       try {
                                                frase.analizCadena(linea);
                                             } catch (IOException e1) {
                                                e1.printStackTrace();
                                             }
                                          System.out.println("vuelvo de analizar cadena");

                     }//fin de if
                   linea = g.leeFichRead(buff);
                   System.out.println("leo siguiente linea"+linea);
                     }//fin de while
                   g.closeFichRead(buff);
                   break;
              default:
                   break;
              case 8:
                   try {
                            n=web.RandomLearn(50);
                   } catch (IOException e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                   }
                   try {
                            web.Freeling();
                   } catch (IOException e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                   }
                   // calcular tiempo transcurrido
                   long end = System.currentTimeMillis();
                   long res = end - start;
                   System.out.println("Segundos: "+res/1000);
                   break;
              case 9:
                   String cadena;
                   cadena="pintor";
                   System.out.println(cadena);
                   try {
```

```
                        n=web.getHTML(cadena);
                    } catch (IOException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                    }
                    try {
                        web.Freeling();
                    } catch (IOException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                    }
                    // calcular tiempo transcurrido
                    long end1 = System.currentTimeMillis();
                    long res1 = end1 - start;
                    System.out.println("Segundos: "+res1/1000);
                    break;
                //Aquí iría el send mensaje de respuesta
                }
}}
```

**8. CShocketNode**

```
import java.io.*;
import java.net.*;

public class CSocketNode extends Thread{
  protected int type;
  ServerSocket socketServer = null;
  Socket socketConn = null;
  InetSocketAddress inetAddress;
  DataOutputStream output;
  DataInputStream input;

  //Thread threadConn;
  int threadStatus;

  public int Init(String address, int port, int t ) {
        type = t;
        threadStatus = 0;
        //SERVER
        if (type==0){ //SERVER
            try {
                socketServer = new ServerSocket(port);
            } catch(IOException e) {
                System.out.println( e );
            }
            return 0;
        }
```

```
		//CLIENT
		if (type==1) { // CLIENT
			try {
				inetAddress = new InetSocketAddress(address, port);
			}
			catch (Exception e) {
				return 1;
			}

			return 0;
		}
		return -1;
	}
	public void HandleConnection(){

		if (type==0){ //SERVER
			if (socketConn == null){
				try{
					if(socketServer==null)socketServer = new
ServerSocket(14005);
					socketServer.setSoTimeout(10);
					Socket socketAux =socketServer.accept();
					socketConn = socketAux;
					input = new
DataInputStream(socketConn.getInputStream());
					output = new
DataOutputStream(socketConn.getOutputStream());
					socketConn.setSoTimeout(10);
				} catch(SocketTimeoutException e){}
				catch(IOException e){
					System.out.println(e);
					}
				if(socketConn == null){
					return;
				}
				OnConnection();
			}
		}
		if (type==1) { // CLIENT
			if (socketConn == null){
				Socket socketAux = new Socket();
				try{
					socketAux.connect(inetAddress, 10);
					socketConn = socketAux;
					input = new
DataInputStream(socketConn.getInputStream());
					output = new
DataOutputStream(socketConn.getOutputStream());
					socketConn.setSoTimeout(10);
				} catch(SocketTimeoutException e){
```

```
                                try{
                                        socketAux.close();
                                } catch(IOException e2){}
                                return;
                        }
                        catch(IOException e){}
                        OnConnection();
                }
        }
}
public void SendMsg(byte _code, String str){
        byte[] byteAux = new byte[2048];
        char[] charAux = new char[2048];

        if(IsConnected()==0) return;
        str.getChars(0, str.length(), charAux, 0);

        byteAux[0] = 57;
        byteAux[1] = 48;
        byteAux[2]=(byte) ((str.length()+1)% 256);
        byteAux[3]=(byte) ((str.length()+1)/ 256);
        byteAux[4]=(byte) _code;
        for (int i = 0;i<str.length();i++){
                byteAux[5 + i] = (byte)charAux[i];
        }
        try{
                output.write(byteAux, 0, str.length()+5);
        } catch(IOException e){}
}


public void ReceiveMsg(byte[] buffer){
        byte code;
        try{
                if(input.available() > 0) {
                        input.read(buffer, 0, 5);
                        int lon = buffer[2]+buffer[3]*256;
                        if (lon < 0)
                                lon = 256 + lon;
                        code=buffer[4];
                        lon--;
                        input.read(buffer, 0, lon);
                        buffer[lon] = 0;
                        OnMsg(buffer,code);
                }}
        catch (IOException e){Error();
        }
}
public void OnMsg(byte[] cad, int length) throws IOException{
        String str;
```

```
            str="Fin";
            try {
               str=new String(cad,"UTF8");
            } catch (UnsupportedEncodingException e) {
               e.printStackTrace();
            }
            Comunicacion.TcpMessage(length,str);
        }

        public void OnConnection(){

              try {
            Thread.sleep(5000);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
            if(IsConnected() == 1)
            {
               System.out.println("conectado");
            }
            else
            {
               System.out.println("Desconectado");
            }
        }


        public int IsConnected(){
                if(socketConn == null){
                        return -1;
                }
                if(socketConn.isConnected()){
                        return 1;
                } else{
                        return 0;
                }
        }
        @Override
        public void run(){
                threadStatus = 1;
                OnConnection();
                while(threadStatus == 1){
                        if (IsConnected() == 1){
                                byte msg[] = new byte[2048];
                                ReceiveMsg(msg);
                        } else{
                                HandleConnection();
                        }
                        try{
                                Thread.sleep(10);
```

```
                    } catch(InterruptedException e){}
            }
            threadStatus = 0;
            Close();
            System.out.println("fin del thread");
    }
    public void StartThread(){
            start();
    }
    public void Close(){
            Error();
    }
    public void Error(){
            try{
                    if(output!=null)output.close();
                    if(input!=null)input.close();
                    if(socketConn!=null)socketConn.close();
                    //socketServer.close();
            } catch(IOException e){}
            socketConn = null;
    }
}
```

**9. CWeb**

```
import java.io.*;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLDecoder;
import javax.*;

public class CWeb {
  public
  String aux;
  byte buffer[];
  int indice=0;

  int ya;
  int okya;
  int oldya;
  int nlineas;
  int l;
  int tope;
  int posLista=0;
  boolean con1,con2;

  String value1;
  String value2;
```

```
	String Datos[];

	public CWeb(){
		buffer = new byte[64000];
	}

	public int getHTML(String materia) throws IOException {

		String direccion="https://es.wikipedia.org/wiki/"+materia;
		String ref;
		String inputLine;
		String outputLine;
		URLConnection uc;
		URL url=null;
		nlineas=0;
			con1=false;
			con2=false;
		try {
			url = new URL(direccion);
			uc = url.openConnection();
			uc.connect();
			BufferedReader in = new BufferedReader(new
InputStreamReader(uc.getInputStream()));
			while ((inputLine = in.readLine()) != null) {
				Buscar(inputLine);
				if(con1 && con2){
					in.close();
					outputLine=value1+" es "+value2;

	AppendLine("C:/progproyecto/protege/input.txt",outputLine+" @ ");
					AppendLine("C:/progproyecto/protege/input.txt","\r\n");
					nlineas++;
					break;
					}
				}
		}catch (FileNotFoundException fo){
			System.out.println("");
		}
		catch (IOException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		int res= Wiki(direccion);
		return res;
	}

	public int Wiki(String direccion) throws IOException{
		String ref;
		URL url = new URL(direccion);
		URLConnection uc = url.openConnection();
```

```
                uc.connect();
                //Creamos el objeto con el que vamos a leer
                BufferedReader in = new BufferedReader(new
InputStreamReader(uc.getInputStream()));
                String inputLine;
                String outputLine;
                ya=0;
                okya=0;
                oldya=0;
                l=0;
                con1=false;
                con2=false;
                buffer[0]=0;

                while ((inputLine = in.readLine()) != null) {
                        ref=Referencia(inputLine);
                        if(!ref.isEmpty()){
                                //Encontrada
                                in.close();
                                direccion="https://www.wikidata.org/wiki/"+ref;
                                url = new URL(direccion);
                                uc = url.openConnection();
                                uc.connect();
                                in = new BufferedReader(new
InputStreamReader(uc.getInputStream()));
                                while ((inputLine = in.readLine()) != null) {
                                        Buscar(inputLine);
                                        if(con1 && con2){
                                                in.close();
                                                outputLine=value1+" es "+value2;

        AppendLine("C:/progproyecto/protege/input.txt",outputLine+" @ ");

        AppendLine("C:/progproyecto/protege/input.txt","\r\n");
                                                nlineas++;
                                                return nlineas;
                                                }
                                        }
                        break;
                        }
                }
                in.close();
                return nlineas;
        }

        public int RandomLearn(int _tope) throws IOException {

                tope=_tope;
                int res=0;
                nlineas=0;
```

```
		for (int i=0;i<tope;i++){
			String direccion="https://es.wikipedia.org/wiki/Especial:aleatoria";
			try {
				Thread.sleep(50);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			res= Wiki(direccion);
		}
		return res;

	}

	public int ModelLearn(String what, int _tope) throws IOException {
		String ref;
		int res=0;
		String direccion,direccion2;
		String inputLine;
		String inputLine2;
		String outputLine;
		String outputLine2;
		URLConnection uc,uc2;
		URL url=null;
		URL url2=null;
		boolean mas;

	direccion="https://www.wikidata.org/w/index.php?title=Special:Search&limit=100&s
earch="+what;
		try{
			url = new URL(direccion);
			uc = url.openConnection();
			uc.connect();

			//Creamos el objeto con el que vamos a leer
			BufferedReader in = new BufferedReader(new
InputStreamReader(uc.getInputStream()));
			BufferedReader dos;

			while ((inputLine = in.readLine()) != null) {
				posLista=0;

				while(posLista >=0){
				ref=Contenido(inputLine);
				if(!ref.isEmpty()){
					UnoLearn(ref);
				}}
			}
			in.close();
		}catch (FileNotFoundException fo){
```

```
                        System.out.println("");
                }
                catch (IOException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                }
                return res;
        }

        public int UnoLearn(String Numb) throws IOException {
                String ref;
                String direccion;
                String inputLine;
                String outputLine;
                URLConnection uc;
                URL url=null;
                int res=0;
                nlineas=0;
                        con1=false;
                        con2=false;

                        direccion="https://www.wikidata.org/wiki/"+Numb;
                        try {
                                url = new URL(direccion);
                                uc = url.openConnection();
                                uc.connect();
                                BufferedReader in = new BufferedReader(new
InputStreamReader(uc.getInputStream()));
                                while ((inputLine = in.readLine()) != null) {
                                        Buscar(inputLine);
                                        if(con1 && con2){
                                                in.close();
                                                outputLine=value1+" es "+value2;

        AppendLine("C:/progproyecto/protege/input.txt",outputLine+" @ ");

        AppendLine("C:/progproyecto/protege/input.txt","\r\n");
                                                nlineas++;
                                                break;
                                                }
                                        }
                        }catch (FileNotFoundException fo){
                                System.out.println("");
                        }
                        catch (IOException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }
                return res;
        }
```

```
	public int SystemicLearn(int inIni, int inEnd) throws IOException {
		String ref;
		String direccion;
		String inputLine;
		String outputLine;
		URLConnection uc;
		URL url=null;
		int res=0;
		nlineas=0;
		for (int i=inIni;i<=inEnd;i++){
			con1=false;
			con2=false;

			ref="Q"+Integer.toString(i);
			System.out.print(ref+" ");
			direccion="https://www.wikidata.org/wiki/"+ref;
			try {
				url = new URL(direccion);
				uc = url.openConnection();
				uc.connect();
				BufferedReader in = new BufferedReader(new
InputStreamReader(uc.getInputStream()));
				while ((inputLine = in.readLine()) != null) {
					Buscar(inputLine);
					if(con1 && con2){
						in.close();
						outputLine=value1+" es "+value2;

	AppendLine("C:/progproyecto/protege/input.txt",outputLine+" @ ");

	AppendLine("C:/progproyecto/protege/input.txt","\r\n");
						nlineas++;
						break;
						}
					}
			}catch (FileNotFoundException fo){
				System.out.println("");
			}
			catch (IOException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
		}
		return res;
	}


	public String Referencia(String msg){
		int n,pos;
```

```
		String ref="";
		//"wgWikibaseItemId":"Q76"
		n=msg.indexOf("wgWikibaseItemId", 0);
		if(n>0){
			n=n+20;
			pos=msg.indexOf("", n);
			ref="Q"+msg.substring(n,pos);
		}
		return ref;
	}

	public String Contenido(String msg){
		int n,pos;
		String ref="";
		n=msg.indexOf("wb-itemlink-id", posLista);
		if(n>0){
			n=n+17;
			pos=msg.indexOf(')', n);
			ref=msg.substring(n,pos);
			posLista=pos;
		}
		else{
			posLista=-1;
		}
		return ref;
	}

	public void Buscar(String msg) throws UnsupportedEncodingException{
		int posI,posF;
		String str="\\\"language\\\":\\\"es\\\",\\\"value\\\":";
		int lon=str.length();
		posI=msg.indexOf(str);
		if(!con1){
		if(posI >= 0){
			posI=posI+lon+2;
			posF=msg.indexOf("}", posI)-2;
			value1=msg.substring(posI, posF);
			value1=Acento(value1);
			con1=true;
		}}

		posI=msg.indexOf(str,posI);
		if(posI >= 0){
			posI=posI+lon+2;
			posF=msg.indexOf("}", posI)-2;
			value2=msg.substring(posI, posF);
			value2=Acento(value2);
			con2=true;
		}
	}
```

```
	public void Localizar(String msg){
		String str;

		int lon=msg.length();

		for (int i=0; i < lon; i++)
		{
			if((i<(lon-3))&&((msg.substring(i, i+3)).equals("<p>")))
			{
				i=i+3;
				ya=1;
				oldya=1;
				okya=1;
			}
			else if((i<(lon-4))&&((msg.substring(i,i+4)).equals("</p>")))
			{
				buffer[l]=0;
				ya=0;
				okya=0;
				i=i+4;
			}
			else if(msg.substring(i,i+1).equals("["))
			{
				ya=0;
			}
			else if(msg.substring(i,i+1).equals("]"))
			{
				ya=1;
				i++;
			}

			else if(msg.substring(i,i+1).equals("("))
			{
				ya=0;
			}
			else if(msg.substring(i,i+1).equals(")"))
			{
				ya=oldya;
				i++;
			}
			if(l > 80 && ((msg.substring(i,i+1).equals(" "))
||(msg.substring(i,i+1).equals","))))
			{
				buffer[l]=0;
				try {
					aux= new String(buffer,"UTF-8");
					//Datos[nlineas]=aux.substring(0,l);
				} catch (UnsupportedEncodingException e) {
					e.printStackTrace();
```

```
                        }
                        nlineas++;
                        if(nlineas > tope)return;
                        buffer[0]=0;
                        l=0;
                }
                if(ya>0)
                {

                        if(msg.substring(i,i+1).equals("<"))
                        {
                                okya=0;
                        }
                        else
                        {
                                if(msg.substring(i,i+1).equals(">"))
                                {
                                        okya=1;
                                        ya=oldya;
                                }
                                else
                                {
                                        if(okya>0)
                                        {
                                                buffer[l]=(byte) msg.charAt(i);

                                                if((buffer[l]==',') || (buffer[l]==';'))
                                                {
                                                        buffer[l]=0;
                                                        try {
                                                                aux=                    new
String(buffer,"UTF-8");

        //Datos[nlineas]=aux.substring(0,l);
                                                        }                               catch
(UnsupportedEncodingException e) {
                                                                e.printStackTrace();
                                                        }
                                                        nlineas++;
                                                        if(nlineas > tope)return;
                                                        buffer[0]=0;
                                                        l=0;
                                                }
                                                else if(buffer[l]=='.')
                                                {
                                                                //Es un número
                                                                byte
siguiente=(byte)msg.charAt(i+1);
                                                                if(siguiente>='0'       &&
siguiente <='9')
```

```
                                                        {
                                                                l++;
                                                                buffer[l]=0;
                                                        }
                                                        else
                                                        {

                                                                buffer[l]=0;
                                                                try {
                                                                        aux=    new
String(buffer,"UTF-8");

        //Datos[nlineas]=aux.substring(0,l);
                                                                }               catch
(UnsupportedEncodingException e) {

        e.printStackTrace();
                                                                }
                                                                nlineas++;
                                                                if(nlineas          >
tope)return;
                                                                buffer[0]=0;
                                                                l=0;
                                                        }
                                                }
                                                else
                                                {
                                                        l++;
                                                        buffer[l]=0;
                                                }
                                        }
                                }
                        }
                }
        }
        public void AppendLine(String donde,String msg){
                try{
                        FileOutputStream fos = new FileOutputStream(donde, true);
                        fos.write(msg.getBytes());
                        fos.write('\n');
                        fos.close();
                        System.out.println(msg);
                        }
                catch(FileNotFoundException ex)
                {System.out.println("FileNotFoundException : " + ex);}
                catch(IOException ioe)
                {System.out.println("IOException : " + ioe);}
        }
        public String Acento(String msg){
```

```
                msg=msg.replace("\\\\u00c1","A");
                msg=msg.replace("\\\\u00e1","a");
                msg=msg.replace("\\\\u00c9","E");
                msg=msg.replace("\\\\u00e9","e");
                msg=msg.replace("\\\\u00cd","I");
                msg=msg.replace("\\\\u00ed","i");
                msg=msg.replace("\\\\u00d3","O");
                msg=msg.replace("\\\\u00f3","o");
                msg=msg.replace("\\\\u00da","U");
                msg=msg.replace("\\\\u00fa","u");
                msg=msg.replace("\\\\u00dc","Ü");
                msg=msg.replace("\\\\u00fc","ü");
                msg=msg.replace("\\\\u00d1","Ñ");
                msg=msg.replace("\\\\u00f1","ñ");
                return msg;

        }
       /*public void Print(String donde,int n) throws IOException{

            File archivo = new File(donde);
            BufferedWriter bw;

            bw = new BufferedWriter(new FileWriter(archivo));

            for (int i=0;i < n;i++)
            {    System.out.println("Estamos en el for ");
                bw.write(Datos[i]);
                bw.newLine();
                JOptionPane.showMessageDialog(new JFrame(), Datos[i],
"TCP_SERVER",JOptionPane.ERROR_MESSAGE);
            }
            bw.close();

            try
            {
                Process p = Runtime.getRuntime().exec
("C:/progproyecto/Freeling/bin/analyzer.bat");

            }
            catch (Exception e)
            {
                System.out.println("NO");
            }
        }*/

        public void Freeling() throws IOException{

            try
            {
```

```
                Process p = Runtime.getRuntime().exec
("C:/progproyecto/Freeling/bin/analyzer.bat");
            }
            catch (Exception e)
            {
                System.out.println("NO freeling");
            }
        }
    }
```

**10. Buscador**

```
    import java.io.*;
    import java.util.*;
    import org.apache.jena.*;

    public class Buscador {
      public  int numclases(){
       int clase= 0;
       OntClass cls = null;
       String ch =null;

       OntModel model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
       System.out.println("Creamos el modelo de la Ontología ");
       model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
       System.out.println("Leemos la onto ");

       for (Iterator<OntClass> i = model.listClasses();i.hasNext();){
                    cls = i.next();
                    ch= cls.getLocalName();
                    if(ch!=null){
                        clase++;
                    System.out.println("numero clases: "  +clase );
                    }

       }
      return clase;
      }
      public  int numinstancias(){
                int ins= 0;

                OntModel model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
                System.out.println("Creamos el modelo de la Ontología ");

model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                System.out.println("Leemos la onto ");
```

```
                for (Iterator<OntClass> i = model.listClasses();i.hasNext();){
                        OntClass cls1 = i.next();
                        for(Iterator it = cls1.listInstances(true);it.hasNext();){
                                Individual ind = (Individual)it.next();
                                if(ind.isIndividual()){
                                        ins++;
                                }
                        }
                }
                return ins;

        }

         public int numrelaciones(){
                 int rel= 0;


                 OntModel                                        model                                   =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
                 System.out.println("Creamos el modelo de la Ontología ");

model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                 System.out.println("Leemos la onto ");
                 String NS = "relaciones";
                 OntClass clasrel = model.getOntClass(NS+":"+"relaciones");


                 for(Iterator<OntClass> x = clasrel.listSubClasses();x.hasNext();){
                                 OntClass subcls=x.next();
                                 String re= subcls.getLocalName();
                                 if(re!=null){
                                         rel++;
                                 System.out.println("numero clases: "  +rel );
                                 }
                 }
                 return rel;
                 }
         public int buscarexiste(String busca){
         int sol=0;
                 OntModel                                        model                                   =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
                 System.out.println("Creamos el modelo de la Ontología ");

        model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                 System.out.println("Leemos la onto ");
                 String NS = "relaciones";
                 OntClass claserel = model.getOntClass(NS+":"+"relaciones");
                 for(Iterator<OntClass> x = claserel.listSubClasses();x.hasNext();){
                         OntClass subcls=x.next();
```

```
                        String re= subcls.getLocalName();
                        if(busca.equals(re)){
                                sol=1;
                                }
                        }
                if(sol!=1){
                        for (Iterator<OntClass> i = model.listClasses();i.hasNext();){
                                OntClass cls = i.next();
                                String clase=cls.getLocalName();
                                if(clase.equals(busca)){
                                        sol=2;
                                }
                }}
                if(sol!=1&sol!=2){
                        for (Iterator<OntClass> i = model.listClasses();i.hasNext();){
                                OntClass cls1 = i.next();
                                for(Iterator it = cls1.listInstances(true);it.hasNext();){
                                        Individual ind = (Individual)it.next();
                                        String indivi= ind.getLocalName();
                                        if(indivi.equals(busca)){
                                                sol=3;}
                                        }

                }}

        return sol;
        }


        public String obtenerindi(String clase){
                String responder = null;
                int numindi = 0;
                String cadena = null;
                OntModel                                model                                =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
                        System.out.println("Creamos el modelo de la Ontología ");

       model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                        System.out.println("Leemos la onto ");
                        String NS = "relaciones";
                        OntClass clasebus = model.getOntClass(NS+":"+clase);

                                for(Iterator it = clasebus.listInstances(true);it.hasNext();){
                                        Individual ind = (Individual)it.next();
                                        String indivi= ind.getLocalName();
                                        if(indivi!=null){
                                                numindi++;
                                                }
                                        if(cadena==null){
                                                cadena=indivi;
```

```
                                        }
                                        else{
                                        cadena=cadena+" "+indivi;
                                        System.out.println("contenido de cadena: "  +cadena );}

                                }
                        responder=numindi+" "+cadena;
                return responder;
        }
        public String obtenerclase (String relacion){
                String responder = null;
                int numclase = 0;
                String cadena = null;
                OntModel                                model                                =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
                        System.out.println("Creamos el modelo de la Ontología ");

        model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                        System.out.println("Leemos la onto ");
                        String NS = "relaciones";
                        OntClass clasebus = model.getOntClass(NS+":"+relacion);
                        for(Iterator<OntClass> x = clasebus.listSubClasses();x.hasNext();){
                                OntClass subcls=x.next();
                                String re= subcls.getLocalName();
                                if(re!=null){
                                        numclase++;
                                        }
                                if(cadena==null){
                                        cadena=re;
                                }
                                else{
                                cadena=cadena+" "+re;
                                System.out.println("contenido de cadena: "  +cadena );}
                        }
                responder=numclase+" "+cadena;
         return responder;
          }



       }
```

**11. Fichero**

```
       import java.io.*;

       public class Fichero {
```

```
	//private static final String nombreFichero="C:/progproyecto/protege/frases.txt" ;

	public void escribefichero(String nombreFichero,String cadena,boolean append){
//creo un fichero extendido que no se va a borrar(append)
		try{
			FileWriter fichSalida = null; //Para acceso al archivo
			fichSalida = new FileWriter(nombreFichero,append);//Apertura fichero
			BufferedWriter buffSalida = new BufferedWriter(fichSalida);//Buffer
para guardar los datos del texto
			buffSalida.write(cadena);//llamo al metodo write para escribir en el
buffer
			buffSalida.newLine();//salto de linea
			buffSalida.close();//Cerrar fichero

		}
		catch(IOException e){
			System.out.println("Error en fichero");
		}//Fin del catch
	}//Fin del metodo escribir

	public void limpiaFichero(String nombreFichero){
		try
		{
		FileWriter fichSalida1 = null;
		fichSalida1 = new FileWriter(nombreFichero); //Lo abro de escritura (no
append)
		fichSalida1.close();                                    //Al cerrarlo se
borran los datos del fichero
		}catch(IOException e){
			System.out.println("Error al vaciar fichero de salida ");
		} // fin de catch
	}//fin limpia fichero


	public BufferedReader openFichRead(String nombreFichero){
		FileReader fichEntrada1;
		BufferedReader buffEntrada1;
		try
		{
			fichEntrada1 = null;
			fichEntrada1 = new FileReader(nombreFichero);
			buffEntrada1=new BufferedReader(fichEntrada1); //El buffer apunta al
fichero
		}catch(IOException e){
			System.out.println("Error en apertura de fichero de lectura ");
			buffEntrada1=null;
		} // fin de catch
		return buffEntrada1; //Devuelve el buffer
	}//fin openFichRead
```

```
  public String leeFichRead(BufferedReader buffEntrada){
        String lineaRead;
        try
        {
              lineaRead=buffEntrada.readLine();

        }catch(IOException e){
              System.out.println("Error en lectura del fichero de entrada ");
              lineaRead=""; //Si hay un error dejo la linea vacia
        } // fin de catch

        return lineaRead;
  }// fin leeFichRead

  public void closeFichRead(BufferedReader buffEntrada){

        try
        {
              buffEntrada.close();
        }catch(IOException e){
              System.out.println("Error en lectura del fichero de entrada ");
        } // fin de catch
  }//fin closeFichRead

}//Fin de clase
```

**12.AddOnto**

```
import java.io.*;
import java.util.*;
import org.apache.jena.*;

public class addOnto {
  public void anadir(ArrayList<String> cadena) throws IOException   {
        String relacion;
        String derecha;
        String izquierda;
        String clase;
        String subclase;
        String Individuo;
        int banderaClase=0;
        int banderasubClase=0;
        int banderaIndi=0;
        int banderaprop=0;
        OntClass cls = null;
        Individual defindi=null;
        OntClass def =null;
```

```
                OntClass subcls=null;
                OntClass defsubcla=null;




                izquierda = cadena.get(0);
                relacion = cadena.get(1);
                derecha = cadena.get(2);

                System.out.println("la relacion es :"+relacion);
                System.out.println("la palabra dcha es :"+derecha);
                System.out.println("la palabra izquda es :"+izquierda);

                //Creamos un modelo de la ontologia (es una extension del modelo de Jena rdf)
                        OntModel model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
                        System.out.println("Creamos el modelo de la Ontología ");

        model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                        System.out.println("Leemos la onto ");
                        //Establecemos el NameSpace por defecto para nuestra ontología
                        String NS = "relaciones";



                //buscamos si esta la relacion en la ontologia
                        for (Iterator<OntClass> i = model.listClasses();i.hasNext();){
                                cls = i.next();
                                System.out.println(cls);
                                clase=cls.getLocalName();
                                System.out.println(cls.getLocalName()+": ");
                                System.out.println("Dentro del for ");
                                if(clase.equals(relacion)){
                                        System.out.println("La relacion ya existe en la Ontología
");
                                        def = cls;
                                        banderaClase=1;
                                }//fin if
                        }
                //Si la relacion no esta en la ontologia, procedemos a añadirla
                        if(banderaClase==0){
                                System.out.println("La relacion NO existe en la Ontología ");
                                OntClass miclase = model.getOntClass(NS+":"+"relaciones");
                                OntClass cla = model.createClass(NS+":"+relacion);
                                miclase.addSubClass(cla);

                                //Almacenamos la ontología en un fichero OWL (Opcional)
                                File file = new
File("C:/progproyecto/protege/OntosOWL/ontologia.owl");
```

```
                                //Hay que capturar las Excepciones
                                if (!file.exists()){
                                    file.createNewFile();
                                }
                                model.write(new PrintWriter(file));
                                banderaClase=1;
                                def=cla;

                        }
                //Si la relacion si esta en la ontologia vemos si tiene la subclase
                        if(banderaClase==1){
                                System.out.println("Pasamos a ver las subclases de la relacion seleccionada");
                                System.out.println(def);
                                for(Iterator<OntClass> x = def.listSubClasses();x.hasNext();){
                                        subcls=x.next();
                                        System.out.println(subcls);
                                        subclase=subcls.getLocalName();
                                        System.out.println(subcls.getLocalName()+": ");

                                        if(subclase.equals(derecha)){
                                                System.out.println("La clase ya existe en la Ontología ");
                                                banderasubClase=1;
                                                defsubcla = subcls;
                                                }
                                }
                        }
                //Si la subclase no esta en la ontologia la añadimos
                        if(banderasubClase==0){
                                System.out.println("La clase NO existe en la Ontología, procedemos a añadirla ");
                                OntClass misubclase = model.getOntClass(NS+":"+relacion);
                                OntClass subcla = model.createClass(NS+":"+derecha);
                                misubclase.addSubClass(subcla);

                                //Almacenamos la ontología en un fichero OWL (Opcional)
                                File file = new File("C:/progproyecto/protege/OntosOWL/ontologia.owl");
                                //Hay que capturar las Excepciones
                                if (!file.exists()){
                                    file.createNewFile();
                                }
                                model.write(new PrintWriter(file));
                                banderasubClase=1;
                                defsubcla=subcla;
                        }
                //Si la subclase esta en la ontologia vemos si tiene la instancia
                        if(banderasubClase==1){
                                for(Iterator it = defsubcla.listInstances(true);it.hasNext();){
```

```
                                        Individual ind = (Individual)it.next();
                                        if(ind.isIndividual()){
                                                Individuo=ind.getLocalName();
                                                if (Individuo.equals(izquierda)){
                                                        banderaIndi=1;
                                                        System.out.println("El individuo ya existe
en la ontología");
                                                        defindi=ind;
                                                        banderaprop=1;
                                                }
                                        }
                                        }

                        }
                //Si el individuo no esta en la ontología procedemos a añadirlo
                        if(banderaIndi==0){
                                System.out.println("El individuo no esta en la ontologia,
procedemos a añadirlo ");
                                Individual                      izqu                    =
model.createIndividual(NS+":"+izquierda,defsubcla);
                                //DatatypeProperty              confianza               =
model.getDatatypeProperty(NS+":"+"confianza");
                                //izqu.setPropertyValue(confianza,
model.createTypedLiteral(20));

                                //Almacenamos la ontología en un fichero OWL (Opcional)
                                File            file            =               new
File("C:/progproyecto/protege/OntosOWL/ontologia.owl");
                                //Hay que capturar las Excepciones
                                if (!file.exists()){
                                    file.createNewFile();
                                }
                                model.write(new PrintWriter(file));
                                banderaIndi=1;
                                defindi=izqu;
                        }
                //Si el individuo estaba en la ontologia incrementamos el factor de confianza
                        if(banderaprop==1){
                                //DatatypeProperty              fiable                  =
model.getDatatypeProperty(NS+":"+defindi);
                                System.out.println("El indvalor de confianza es ");
                                //System.out.println(fiable.getPropertyValue(fiable));
                        }

        }
        }//fin de la clase
```

**13.AddRelcion**

```
import java.io.*;
import java.util.*;
import org.apache.jena.*;

public class addrelacion {
  int banderaClase=0;
  OntClass def =null;


  public void anadirrel(String relacion) throws IOException{
        OntClass def;
        //Creamos un modelo de la ontologia (es una extension del modelo de Jena rdf)
        OntModel model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        System.out.println("Creamos el modelo de la Ontología ");

  model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
        System.out.println("Leemos la onto ");
        //Establecemos el NameSpace por defecto para nuestra ontología
        String NS = "relaciones";

              System.out.println("La relacion NO existe en la Ontología ");
              OntClass miclase = model.getOntClass(NS+":"+"relaciones");
              OntClass cla = model.createClass(NS+":"+relacion);
              miclase.addSubClass(cla);

              //Almacenamos la ontología en un fichero OWL (Opcional)
              File file = new
File("C:/progproyecto/protege/OntosOWL/ontologia.owl");
              //Hay que capturar las Excepciones
              if (!file.exists()){
                  file.createNewFile();
              }
              model.write(new PrintWriter(file));
  }


  public void anadirclase(String relacion, String clase) throws IOException{
        System.out.println("Pasamos a ver las subclases de la relacion seleccionada");
        System.out.println(relacion);
        String subclase;
        OntClass subcls;
        int banderasubClase=0;
        //Creamos un modelo de la ontologia (es una extension del modelo de Jena rdf)
              OntModel model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
              System.out.println("Creamos el modelo de la Ontología ");
```

```
        model.read("file:///progproyecto/protege/OntosOWL/ontologia.owl","RDF/XML");
                                System.out.println("Leemos la onto ");
                                //Establecemos el NameSpace por defecto para nuestra ontología
                                String NS = "relaciones";

                        OntClass mirel = model.getOntClass(NS+":"+relacion);

                        for(Iterator<OntClass> x = mirel.listSubClasses();x.hasNext();){
                                subcls=x.next();
                                System.out.println(subcls);
                                subclase=subcls.getLocalName();
                                System.out.println(subcls.getLocalName()+": ");
                                if(subclase.equals(clase)){
                                        System.out.println("La clase ya existe en la Ontología ");
                                        banderasubClase=1;
                                        }

                        }
                        if(banderasubClase==1){
                                System.out.println("La clase ya existe en la Ontología ");
                        }
                        else if(banderasubClase==0){
                                System.out.println("La añadimos en la Ontología ");
                                OntClass subcla = model.createClass(NS+":"+clase);
                                mirel.addSubClass(subcla);

                                //Almacenamos la ontología en un fichero OWL (Opcional)
                                File                    file                    =                       new
File("C:/progproyecto/protege/OntosOWL/ontologia.owl");
                                //Hay que capturar las Excepciones
                                if (!file.exists()){
                                    file.createNewFile();
                                }
                                model.write(new PrintWriter(file));
                        }
                  }
                }
```

**14.TripletaAnalizador**

```
        import java.io.*;
        import java.util.*;

        public class TripletaAnalizador {


          private String palabra1="";
```

```
	private String palabra2="";
	private String palabra3="";


	public void analizarfrase(){
	BufferedReader buff;
	String linea;
	ArrayList<String> listaTripleta = new ArrayList<String>();
	Iterator<String> numT = listaTripleta.iterator();
	int in=1;
	String tablaT[];
	tablaT = new String[3];
	int indiTabla=0;
	int completo=0;

	Fichero fich = new Fichero();
	buff= fich.openFichRead("C:/progproyecto/protege/output.txt");
	linea = fich.leeFichRead(buff);
	System.out.println("linea leida ventana actualiza :"+linea);
	while (linea != null){
		System.out.println("dentro del while :"+linea);
		StringTokenizer st = new StringTokenizer(linea, " ");
		String cadena="";

		//int numPalabras = st.countTokens();

		while(st.hasMoreTokens()&&(completo==0)) {
			System.out.println("linea hasmore" + in);
			if (in==1){
				palabra1=st.nextToken();
				System.out.println("if1 :"+palabra1);
				in=in+1;
			 }//fin if in1
			if(palabra1.equals("@")){
				for (int x=0;x<tablaT.length;x++){
					cadena=cadena+ tablaT[x]+" ";
				}
				System.out.println("tripleta a almacenar:" +cadena);
				Fichero f = new Fichero();
				String cad= cadena.toLowerCase();
				System.out.println("cadena minusculas: "+cad);
				f.escribefichero("C:/progproyecto/protege/tripletas.txt",cad,
true);
				indiTabla=0;
				tablaT[0]="";
				tablaT[1]="";
				tablaT[2]="";
				in=1;
				completo=1;
			}//fin del if@
```

```
                        else{
                                if(in==2){
                                        palabra2= st.nextToken();
                                        System.out.println("tomamos la palabra2 :"+palabra2);
                                        in++;
                                }//fin if in2
                                else if(in==3){
                                        palabra3=st.nextToken();
                                        System.out.println("tomamos la palabra3 :"+palabra3);
                                        if(indiTabla==0){
                                                String primeraletra= palabra3.substring(0, 1);
                                                System.out.println("entramos en indiTabla==0 y
primera letra es:" +primeraletra );
                                                if(primeraletra.equals("N")){
                                                        tablaT[indiTabla]=palabra1;
                                                        System.out.println("almaceno en tabla
palabra1 :"+ tablaT[indiTabla]);
                                                        indiTabla++;
                                                        System.out.println("indice de tabla :"+
indiTabla);
                                                        completo=1;
                                                }
                                        }//Fin de if indi0
                                        else if(indiTabla==1){
                                                String primeraletra= palabra3.substring(0, 1);
                                                        if(primeraletra.equals("V")){
                                                                tablaT[indiTabla]=palabra1;
                                                                System.out.println("almaceno en
tabla palabra2:"+ tablaT[indiTabla]);
                                                                indiTabla++;
                                                                completo=1;
                                                        }
                                         }      //Fin if indi1
                                        else if(indiTabla==2){
                                                String primeraletra= palabra3.substring(0, 1);
                                                if(primeraletra.equals("N")){
                                                        tablaT[indiTabla]=palabra1;
                                                        System.out.println("almaceno en tabla
palabra3:"+ tablaT[indiTabla]);
                                                        indiTabla++;
                                                        completo=1;
                                                }
                                        }//fin if indi2
                                }//Fin else in3
                         }//fin else@
                        }//finwhile moretokens

                linea = fich.leeFichRead(buff);
                completo=0;
                in=1;
```

```
  }//Fin de While linea
  fich.closeFichRead(buff);


  }//Fin analizarfrase

}//Fin tripletaanalizador
```

**15. TratFrases**

```
import java.io.*;
import java.util.*;

public class TratFrases {
  String cadena;
  public void analizCadena(String linanaliz) throws IOException   {
            ArrayList<String> listaPalabras = new ArrayList<String>();
            Iterator<String> num = listaPalabras.iterator();
            int numPalabras;
            int in=1;
            String tipo="";
            String tferror="n";
            String palabra1="";
            String palabra2="";
            System.out.println("linea recibida :"+linanaliz);
            StringTokenizer st = new StringTokenizer(linanaliz, " ");
            numPalabras=st.countTokens();
            cadena=linanaliz;
            System.out.println("numero palabras :"+numPalabras);
            if (numPalabras < 3){
                  System.out.println("error menos de 3 palabras :"+numPalabras);
                  tferror="s";
                  String linea;
                  linea=cadena;
                  System.out.println(cadena);
                  Fichero f = new Fichero();
                  f.escribefichero("C:/progproyecto/protege/rechazados.txt",cadena,
true);//Mandar la linea al fichero de no tratados
            }
            else if (numPalabras > 3){
                  System.out.println("error menos de 3 palabras :"+numPalabras);
                  tferror="s";
                  String linea;
                  linea=cadena;
                  System.out.println(cadena);
```

```
                    Fichero f = new Fichero();

f.escribefichero("C:/progproyecto/protege/input.txt",cadena.toLowerCase(), true);//Mandar la
                                                                    linea al fichero input
                    //para pasar por el analizador por si es posible su tratamiento
                }
                else{
                    while(st.hasMoreTokens()) {
                        if (in==1){
                            palabra1=st.nextToken();
                            System.out.println("if1 :"+palabra1);
                        }//fin if
                        else if (in == 2) {
                            tipo= st.nextToken();
                        }//fin if
                        else if(in==3){
                            palabra2=st.nextToken();
                            System.out.println("if3 :"+palabra2);
                            listaPalabras = new
ArrayList<String>(Arrays.asList(palabra1, tipo, palabra2));
                        }//fin if
                        in=in+1;
                    }//fin while

                }//fin de else
            if (tferror=="n"){
            System.out.println("segunda palabras :"+tipo);
            num = listaPalabras.iterator();
            while(num.hasNext()){
                String elemento = num.next();
                System.out.println(elemento);
            }
            String linea;
            linea=cadena;
            System.out.println(cadena);
            Fichero f = new Fichero();
            f.escribefichero("C:/progproyecto/protege/historico.txt",cadena, true);
            addOnto array = new addOnto();
            array.anadir(listaPalabras);
            }//fin del if tferror
            Fichero f = new Fichero();
        }//fin analizCadena
    }//fin clase
```