Sistema Conversacional Inteligente

Autores:

Laruska Roldán Hontanilla

Rubén Rivilla Aparicio

Mª de las Mercedes Huertas Migueláñez

Profesor Director

Héctor Gómez Gauchía

Curso académico 2005-2006

Asignatura:

Proyecto de Sistemas Informáticos

Facultad de Informática

Universidad Complutense de Madrid

SISTEMA CONVERSACIONAL

INTELIGENTE

Realizado por:
LARUSKA ROLDÁN HONTANILLA
RUBEN RIVILLA APARICIO
MERCEDES HUERTAS MIGUELÁÑEZ

ÍNDICE

1. INTRODUCCIÓN

El objetivo del proyecto es definir un sistema de conversación a nivel de contenidos. Que tendrá un control de la conversación basado en:

1. La particularidad del dominio.
2. Tipo del sistema concreto a instanciar.

Está dividido en tres subtareas:

1. *Dominio*: es la que se encarga de los temas o contenidos de la aplicación. Tal cual están en los casos.
2. *Usuario*: adapta la respuesta global dependiendo de lo que ya sabe y su experiencia.
3. *Dinámica de la Conversación*: complementa la respuesta con un aspecto afectivo basado en el estado del usuario. Usa estrategias de la conversación.

Dotado de conocimiento proporcionado por:

- Ontologías: CCBRonto (conceptos para la mecánica de la conversación). Incluye a CBRonto (conceptos necesarios para CBR)
- Bases de casos: El conocimiento dinámico del dominio

Razonamiento: CBR Conversacional,

- Recibe una pregunta que será la descripción de casos a recuperar.
- Se presentan las respuestas de los casos recuperados.
- El usuario refina su descripción para recuperar nuevos casos hasta encontrar la respuesta buscada.

Para el desarrollo del proyecto ha sido necesaria la utilización de las siguientes herramientas y teorías:

- Ontologías
- OWL
- CBR
- Protegé
- Jena
- Pellet
- Eclipse
- Java

Las personas que han realizado este proyecto son:

Laruska Roldán Hontanilla: encargada de la implementación de la GUI de la aplicación y de todo lo referente al razonador DOR (ontología e implementación de las clases correspondientes).

Mercedes Huertas Miguelañez: encargada de todo lo referente al razonador USR (ontología e implementación de las clases correspondientes).

Rubén Rivilla Aparicio: encargado de la investigación e implementación del conector que usa la aplicación para manejar las ontologías y de todo lo referente al razonador DYR (ontología e implementación de las clases correspondientes).

La memoria ha sido creada por los tres componentes del proyecto, especializándose en las partes donde cada uno poseía más conocimiento.

2. INVESTIGACIÓN

→ ONTOLOGÍAS

La RAE (*Real Academia Española*) la define como *"parte de la metafísica que trata del ser en general y de sus propiedades transcendentes"*. Algunos autores piensan que las ontologías son *"colecciones de enunciados redactados en un lenguaje, como el RDF, que define las relaciones entre conceptos y especifica reglas lógicas para razonar con ellos"*.

Las ontologías proceden del campo de la Inteligencia Artificial; son vocabularios comunes para las personas y aplicaciones que trabajan en un dominio. Según el Grupo de Trabajo en Ontologías del consorcio W3C, una ontología define los términos que se usan para describir y representar un cierto dominio. Entendiendo por "dominio" un área específica de interés o un área de conocimiento (física, aeronáutica, medicina, contabilidad, fabricación de productos, etc.) Toda ontología representa cierta visión del mundo con respecto a un dominio. Por ejemplo, una ontología que defina "ser humano" como "espécimen vivo o muerto correspondiente a la especie *Homo sapiens*; expresa una visión del mundo totalmente distinta a la de una ontología que lo defina como "sujeto consciente y libre, centro y vértice de todo lo que existe".

Así como la Ontología –nótese la mayúscula inicial– estudia los tipos de objetos que pueblan la realidad (así como sus propiedades y relaciones), las ontologías catalogan y definen los tipos de cosas que existen en un cierto dominio, así como sus relaciones y propiedades. Por ejemplo, una ontología del mundo empresarial usará conceptos como *Venta*, *Compra*, *Transferencia*, *Pago*, etc.; y relaciones como "Una Transferencia corresponde a una Venta o a una Compra", "Un Pago corresponde a una o varias Transferencias", etc.

Hay diversos autores que han definido el concepto de ontología pero la definición más aceptable de ontología parece ser la expresada por Gruber (1993): una ontología es *"la especificación explícita y formal sobre una conceptualización consensuada*", dentro de un dominio específico. La interpretación de esta definición es que las ontologías definen sus conceptos, propiedades, relaciones, funciones, restricciones y axiomas de forma "explicita" en algún lenguaje de implementación capaz de contener este conocimiento. El término "conceptualización" se refiere a un modelo abstracto de algún fenómeno en el mundo. El conocimiento de las ontologías se establece para que se use de manera "consensuada" y compartida por distintos sistemas, que deberán comprometerse con el vocabulario que se maneja en ontología. Las ontologías también han de implementarse el algún lenguaje entendible o computable por máquina y a eso hace referencial el término "formal".

Existen ontologías específicas y ontologías de carácter general que proporcionan terminologías útiles para varios campos. Lo que una ontología representa para cualquier persona del mundo, se concibe normalmente como una descripción jerárquica de un conjunto de conceptos (jerarquía "es un tipo de"), un conjunto de propiedades y sus relaciones y un conjunto de reglas de inferencia.

Cualquier persona tiene en su cabeza ontologías mediante las que representa y entiende el mundo que lo rodea. Estas ontologías no son explícitas, en el sentido de que no se detallan en un documento ni se organizan de forma jerárquica o matemática. Todos usamos ontologías en las que automóvil representa un medio de transporte y tiene cuatro ruedas. Sin embargo, no formalizamos este tipo de ontologías. No necesitamos explicitar este conocimiento, pues forma parte de lo que todo el mundo sabe. Sin embargo, cuando se tratan términos poco comunes o cuando se quiere que estos términos sean procesados por máquinas, se precisa explicitar las ontologías; esto es, desarrollarlas en un documento o darles una forma que sea inteligible para las máquinas.

Las máquinas carecen de las ontologías con las que nosotros contamos para entender el mundo y comunicarse entre ellas; por eso necesitan ontologías explícitas.

Por otro lado, la mayoría de las ontologías existentes se caracterizan por los siguientes elementos comunes, que servirán para representar el conocimiento de algún dominio:

- ***Conceptos***: ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
- ***Relaciones***: representan la interacción y enlace entre los conceptos del dominio.
- ***Funciones***: son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
- ***Instancias***: se utilizan para representar objetos determinados de un concepto.
- **Axiomas**: teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología.

Se habla también de ontologías ligeras *(lightweight ontologies)* cuando no incluyen axiomas y de ontologías pesadas *(heavyweight ontologies)* cuando los incluyen y son más pesadas cuantos más axiomas contienen.

Existen diferentes clases de ontologías dependiendo de la vista que se quiera considerar. En el nivel superior de su clasificación se consideran cuatro categorías:

1. Ontologías de contenido: construidas para reutilizar su conocimiento, vistas como cajas blancas que permiten reutilizar su vocabulario.
2. Ontologías de Indización: permiten la recuperación de casos cuando los agentes comparten conocimientos a través de los casos.
3. Ontologías de comunicación: usadas por agentes para obtener respuestas a preguntas concretas, vistas como cajas negras de conocimientos.
4. Meta-ontologías: utilizadas para representar ontologías.

Atendiendo al ámbito de la conceptualización, las clasifican en:

1. Ontologías de representación: proporcionan conceptos subyacentes a los paradigmas o formalismos de representación del conocimiento, es decir, proporcionan el vocabulario necesario para modelizar otras ontologías, utilizando determinado paradigma de representación del conocimiento. Es el caso de *Frame Ontology* disponible en el servidor de *Ontolingua*.
2. Ontologías genéricas o meta-ontologías: proporcionan términos genéricos reutilizables en diferentes dominios, como por ejemplo, los términos; *estado, evento, acción, componente*, etc.
3. Ontologías de dominio: que expresan conceptos que son específicos de un dominio determinado. Los conceptos en este tipo de ontologías son definidos usualmente como especializaciones de conceptos existentes en ontologías genéricas.
4. Ontologías de aplicación: contienen todas las definiciones que son necesarias para modelizar los conocimientos requeridos por una aplicación particular. Incluyen conceptos tomados de ontologías de dominio y genéricas, a menudo definidas utilizando el vocabulario indicado en ontologías de representación. Pueden contener extensiones de métodos y tareas específicas. Las ontologías de aplicación tienden a ser menos que las anteriores, pues son especificaciones concretas del dominio que se necesita para realizar una tarea particular en ese dominio determinado.

Sin embargo el intentar clasificar claramente una ontología en uno de estos dos tipos de clasificación puede ser difícil, porque el límite entre ello no está claramente definido.

Dependiendo del grado de formalidad, las ontologías explícitas se clasifican en informales, semi-informales, semi-formales y formales. Las primeras se expresan directamente en cualquier lenguaje natural. Las segundas se expresan en una forma estructurada y restringida de algún

lenguaje natural. Las terceras se expresan en lenguajes estructurados, como RDF. Por último, las ontologías formales definen los términos mediante lenguajes lógico-matemáticos cuyos símbolos se definen exactamente y sin ambigüedades; en consecuencia, estas ontologías permiten emplear teoremas y demostraciones. Los dos últimos tipos de ontologías permiten que las aplicaciones puedan usar las definiciones de los conceptos del dominio y sus relaciones. Así como los tres primeros tipos de ontologías pueden contener términos ambiguos o inconsistentes, las ontologías formales no los permiten.

Las ontologías favorecen la comunicación entre personas, organizaciones y aplicaciones porque proporcionan una comprensión común de un dominio, de modo que se eliminan confusiones conceptuales y terminológicas.

En los campos de la Inteligencia Artificial, la Teoría de Decisiones y la Teoría de Sistemas Distribuidos (campos muy relacionados con la Web semántica): los investigadores de un campo no pueden leer fácilmente los resultados de los investigadores de los otros, pues se usan diferentes perspectivas y términos para las mismas ideas y conceptos. Construyendo una ontología común para los tres campos, las investigaciones de un campo serían inmediatamente aplicables a los otros.

El mundo empresarial no es tampoco ajeno a los problemas derivados de la falta de un entendimiento común: algunas empresas usan el término "recursos" para lo que son "máquinas" para otras empresas. Para otras, en cambio, los "recursos" son las "materias primas" que usan. Mediante 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).

Las ontologías favorecen también la comunicación entre aplicaciones y la comprensión común de la información entre ellas. Las ontologías serán imprescindibles en la Web semántica y en los futuros sistemas de gestión empresarial porque permitirán que las aplicaciones estén de acuerdo en los términos que usan cuando se comunican. Mediante ellas, será mucho más fácil recuperar información relacionada temáticamente, aun cuando no existan enlaces directos entre las páginas web.

Las ontologías también sirven para conseguir que los sistemas interoperen. Dos sistemas son interoperables si pueden trabajar conjuntamente de una forma automática, sin esfuerzo por parte del usuario. En el campo de la informática, las ontologías sirven para traducir los términos usados por una aplicación a otra (las aplicaciones

pueden estar escritas en distintos lenguajes de programación). Si se quiere que cuatro aplicaciones (A1, A2, A3 y A4) interoperen, se necesitan seis aplicaciones que actúen de "traductores" (A1-A2, A1-A3, A1-A4, A2-A3, A2-A4, A3-A4); con una ontología común (O), sólo se necesitarían cuatro "traductores" (A1-O, A2-O, A3-O, A4-O). Según aumenta el número de aplicaciones que deben interoperar, más necesario se hace emplear ontologías traductoras.

Las ontologías resultan muy útiles para facilitar el razonamiento automático, es decir, sin intervención humana. Partiendo de unas reglas de inferencia, un motor de razonamiento puede usar los datos de las ontologías para inferir conclusiones de ellos.

Se han llevado a cabo distintas investigaciones en diferentes aspectos de las ontologías, como son:

- Lenguajes de representación (Corcho, 2000).
- Desarrollo de ontologías (Jones, et al, 1998).
- Métodos de aprendizaje de ontologías (Maedche y Staab, 2001).
- Sistemas de bibliotecas de ontologías (Ding y Fensel, 2001).

Estas investigaciones están orientadas para llevar a cabo, fundamentalmente, tres tareas:

- Gestión: El propósito principal de las ontologías es habilitar la comparición del conocimiento y su reutilización y, por lo tanto, unos sistemas de bibliotecas de ontologías ayudarían a:
    - *El almacenamiento y organización* abierta nos ayuda a almacenar y organizar las ontologías en un sistema bibliotecario para facilitar el acceso a las mismas y la gestión de ontologías.
    - *La identificación* asocia a cada ontología con un único identificador.
    - *La edición de nuevas versiones* es un aspecto importante, pues las ontologías evolucionan a lo largo del tiempo y un mecanismo de actualización puede asegurar la consistencia de las diferentes versiones de las ontologías.

- Adaptación: Debido a que las ontologías evolucionan a lo largo del tiempo, como hemos dicho, un asunto importante es como difundir y actualizar las ontologías existentes. Esta tarea incluye la *búsqueda, edición y el razonamiento de ontologías* en un sistema de bibliotecas de ontologías.

- Normalización: La integración y la interoperatividad es siempre la preocupación de cualquier sistema abierto.

Las principales ontologías y herramientas ontológicas son:

- Ontolingua: Es un conjunto de herramientas, escritas en Common Liso, para el análisis y traducción de ontologías, que se desarrollaron principios de los años 90 en el KSL de la Universidad de Standford. Ontolingua. Consta de un servidor y un lenguaje de representación. Este servidor facilita un depósito de ontologías para ayudar a los usuarios a crear nuevas ontologías y enmendar en colaboración las ontologías existentes. La ontología almacenada en el servidor se puede convertir a diferentes formatos.

- WebOnto: WebOnto fue desarrollado por el Knowledge Media Institute de la Universidad Abierta (Open University). Se diseñó para soportar búsquedas colectivas y creación y edición de ontologías. Proporciona una interfaz de modificación directa presentando en pantalla las expresiones ontológicas y también una herramienta de debate sobre ontologías llamado Tadzebao, que puede soportar tanto debates sincrónicos como asincrónicos sobre ontologías.

- Conjunto de Herramientas Enterprise: Este conjunto está implementado usando una arquitectura basada en agentes (agent-based architecture) para integrar herramientas descatalogadas (off-the-shelf tools) al stilo “plug-and-play”. Los componentes de las herramientas Enterprise son:

    - Constructor de Procedimientos para capturar modelos de proceso.
    - Herramienta de Agentes para soportar el desarrollo de agentes.
    - Administrador de Tareas para la integración, visualización y soporte de procesos de “actuación” (enactment).
    - Ontología para la comunicación.

- Herramientas de KACTUS: VOID, herramienta de KACTUS, es un medio interactivo de búsqueda, edición y gestión bibliotecas de ontologías. VOID soporta paquetes de trabajo de orientación teórica y de aplicación mediante la provisión de medios en los cuales se puede experimentar con cuestiones teóricas: organización de bibliotecas de ontologías, traducción entre diversos formalismos ontológicos, etc. Para soportar la reutilización de ontologías, las herramientas puede manejar varios formalismos ontológicos (CML, EXPRESS y Ontolingua) y puede mejorar (parcialmente) las traducciones entre estos formalismos.

→ OWL

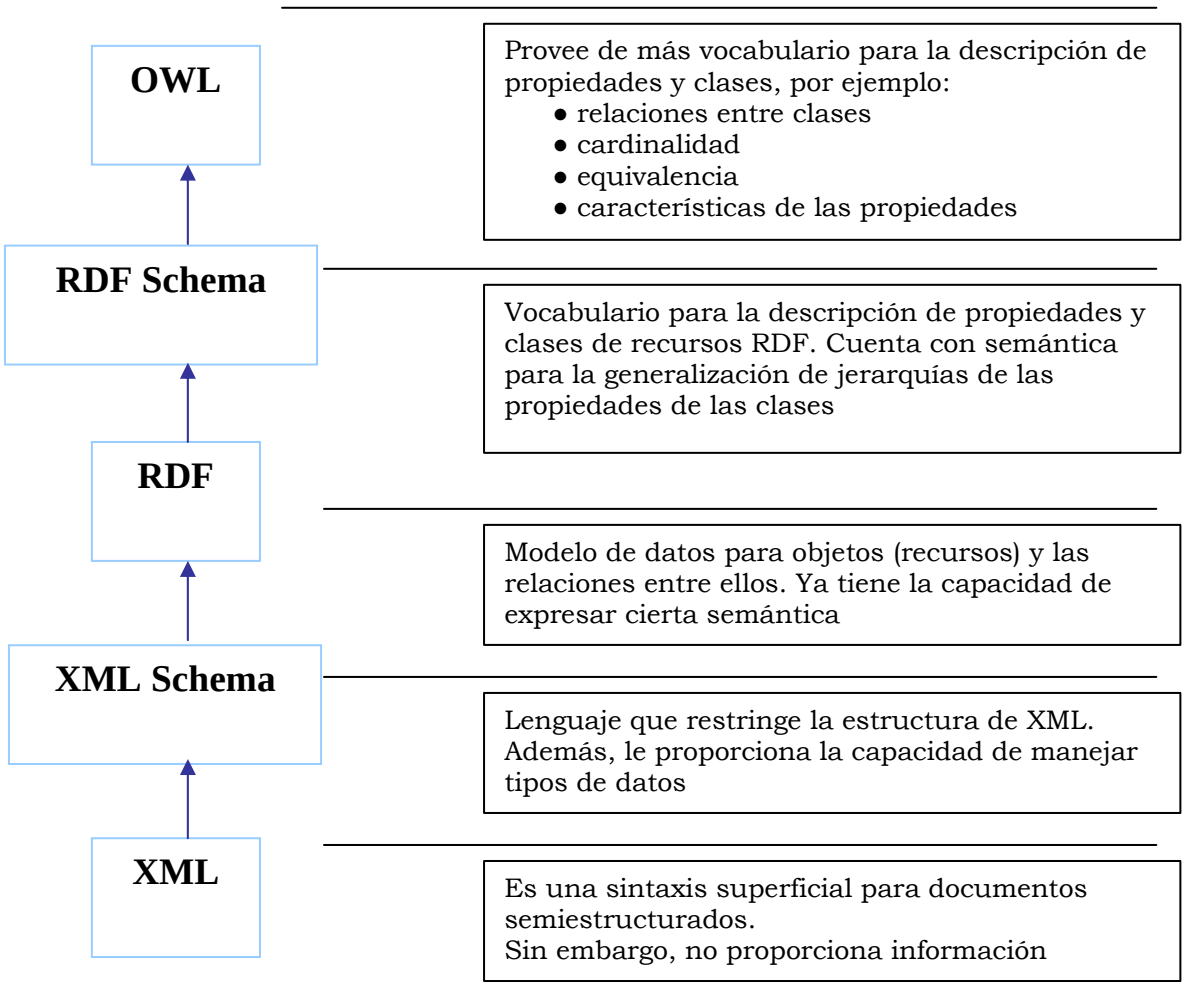
OWL es el acrónimo del inglés Web Ontology Language, un lenguaje de marcado para publicar y compartir datos usando ontologías en la WWW. Tiene como objetivo facilitar un modelo de marcado construido sobre RDF y codificado en XML.

El Web Ontology Language OWL está diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos. OWL puede usarse para representar explícitamente el significado de términos en vocabularios y las relaciones entre aquellos términos. En realidad, OWL es una extensión del lenguaje RDF y emplea las tripletas de RDF, aunque es un lenguaje con más poder expresivo que éste.

OWL posee más funcionalidades para expresar el significado y semántica que XML, RDF, y RDFS.

El motivo del desarrollo de OWL ha sido la puesta en marcha de la Web Semántica, en realidad, una visión para el futuro de la Web en la cual el significado de la información será dado de forma explícita haciendo que las máquinas automaticen de forma más fácil los procesos e integren la información disponible en la Web. La Web Semántica se construirá sobre la sintaxis del lenguaje XML que se mejorará mediante el uso de esquemas RDF para representar el contenido de los datos. El primer nivel sobre RDF requerido para la Web Semántica es un lenguaje de ontologías que pueda describir formalmente el significado de la terminología usada en los documentos web. Si las máquinas son capaces de realizar tareas de razonamiento sobre los documentos en los que se utilice una semántica que vaya más lejos que la semántica básica de RDF Schema, la Web Semántica irá por buen camino. El OWL Use Cases and Requirements Document ofrece más detalles sobre ontologías, motiva la necesidad de un Lenguaje de Ontología Web en términos de seis casos de uso, y formula el diseño de objetivos, requerimientos y objetivos para OWL.

La estructura de la Web Semántica es la siguiente:



El lenguaje OWL tiene 3 sub-lenguajes que incrementan su expresión: OWL Lite, OWL DL, y OWL Full, diseñados para ser usados por comunidades específicas de desarrolladores y usuarios según el nivel de expresividad que precisen éstos.

- *OWL Lite* da soporte a aquellos usuarios que primordialmente necesitan una clasificación jerárquica y restricciones simples. Por ejemplo, soporta restricciones cardinales, pero solamente permite valores cardinales de 0 ó 1. Así pues, es más simple proveer herramientas de soporte para OWL Lite. En resumen, OWL Lite tiene una más baja complejidad formal que OWL DL.
- *OWL DL* da soporte a aquellos usuarios que quieren la máxima expresividad mientras conservan completamente la computacionalidad (todas las conclusiones son garantizadas para ser computables) y resolubilidad (todas

las computaciones terminarán en tiempo finito). OWL DL incluye todos los constructores del lenguaje OWL, pero pueden usarse solamente bajo ciertas restricciones (por ejemplo, mientras una clase puede usarse por una subclase de muchas clases, una clase no puede ser una instancia de otra clase). OWL DL se denomina así debido a su correspondencia con las descripciones lógicas (DL), un campo de investigación que han estudiado los lógicos para la fundación formal de OWL.

- *OWL Full* da soporte a usuarios que requieren el máximo de expresividad y la libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser tratada simultáneamente como una colección de individuos y como un individuo por derecho propio. OWL Full permite a una ontología aumentar el significado del vocabulario predefinido (RDF ó OWL). Es poco probable que algún software racional pueda soportar por completo el razonamiento para cada característica de OWL Full.

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

- Cada ontología legal OWL Lite es una ontología legal OWL DL.
- Cada ontología legal OWL DL es una ontología legal OWL Full.
- Cada conclusión válida OWL Lite es una conclusión válida OWL DL.
- Cada conclusión válida OWL DL es una conclusión válida OWL Full.

En OWL las clases se construyen a partir de descripciones que especifican las condiciones que deben ser satisfechas para que un individuo sea miembro de la clase. Y permite que el significado de las propiedades se vea enriquecido

→ PROTEGÉ

Protegé es un editor de ontologías y bases de conocimiento de libre distribución, que está basado en Java. Soporta Frames, XML Schema, RDF y OWL. Además cuenta con un ambiente "plug-and-play".

→ PELLET

Pellet es una herramienta de libre distribución de Java basada en razonadores OWL DL. Puede ser usada en conjunto con las librerías API de Jena y OWL y también proporciona una interfaz DIG. La API de Pellet proporciona funcionalidades para ver los tipos de validación, chequear la consistencia de las ontologías, clasificar la taxonomía, chequear implicaciones y responder un subconjunto de queries RDQL (Query Language for RDF).

→ JENA

Es un Framework desarrollado por HP Labs para manipular metadata desde una aplicación Java. Incluye una API para el manejo de Ontologías y soporta el lenguaje OWL.

→ CBR (RAZONADOR BASADO EN CASOS)

El razonamiento se plantea como un proceso en el cual se infieren conclusiones analizando reglas generales, comenzando desde el principio.

La fuente del conocimiento primario no son reglas generales, existe una memoria con casos almacenados de episodios anteriores.

Las nuevas soluciones generadas no producen cambios sino que los casos más relevantes de la memoria son enriquecidos y adaptados para encajar con nuevas situaciones.

Un Razonador Basado en Casos resuelve problemas nuevos mediante la adaptación de soluciones previas usadas para resolver problemas similares. La inspiración original de CBR se basó en el razonamiento humano a través del recuerdo.

CBR está basado en dos principios de la naturaleza:

1. El primer principio se basa en que el mundo es regular: los problemas similares tienen soluciones similares. Entonces, las soluciones de los problemas similares anteriores son un punto de partida útil para el nuevo problema a resolver.
2. El segundo principio es que los tipos de problemas que encuentra un agente tienden a repetirse. Entonces, es probable que los problemas futuros sean similares a los problemas actuales.

Los dos principios sostienen que vale la pena recordar y reusar el razonamiento actual: el razonamiento basado en casos es una estrategia de razonamiento eficaz.

Cuando CBR se aplica a problemas nuevos se aumentan los casos, el proceso CBR cambia y aprovecha todo lo nuevo que provee el problema resuelto.

El estudio de CBR está impulsado por dos motivaciones primarias.

1. El primero, el de la Ciencia Cognitiva, con el deseo de modelar la conducta humana.
2. El segundo, el de la Inteligencia Artificial, con el deseo de desarrollar tecnologías para hacer a los sistemas de AI más eficaces.

Gracias al razonamiento basado en casos, se pueden mejorar cinco problemas del razonamiento basado en reglas:

- La Adquisición de Conocimiento: El proceso de adquisición de reglas puede ser laborioso e inestable, las reglas pueden ser difíciles de sacar y no hay convicción de que realmente esas reglas sean suficientes para caracterizar el comportamiento específico. En algunos dominios, las reglas pueden ser difíciles de formalizar o el número de reglas requerido puede ser muy grande. Como los razonadores basados en casos razonan de episodios específicos completos es innecesario descomponer las experiencias y generalizar sus partes en reglas.
- Mantenimiento del Conocimiento: Definir una base de conocimiento inicial es sólo el primer paso hacia una aplicación de AI exitosa. El conocimiento inicial es a menudo imperfecto, requiriendo refinar el conocimiento del sistema. Los cambios en los requerimientos pueden dar un conocimiento obsoleto. En CBR en cambio, un usuario puede agregar los casos perdidos a la biblioteca de casos

sin la intervención de un especialista. Como los sistemas CBR tienen un aprendizaje incremental, ellos pueden tener un único y limitado conjunto de "casos semilla", para ser aumentado con los nuevos casos. Un sistema CBR sólo se ocupa de los tipos de problema que realmente ocurren en la práctica.

- Aumentando la Eficiencia a los Problemas Resueltos: La reutilización de soluciones anteriores aumenta la eficiencia de los problemas resueltos por construir el razonamiento anterior en lugar de repetir el esfuerzo anterior. CBR guarda las soluciones fallidas así como las exitosas, de esta manera puede advertir y evitar problemas potenciales.
- Aumentando la Calidad de las Soluciones: Cuando no se entienden bien los principios de un dominio, las reglas serán imperfectas. En esta situación, las soluciones sugeridas por los casos pueden ser más exactas que aquellas sugeridas por una serie de reglas, porque los casos reflejan lo que realmente pasa (o no pasa) en un conjunto dado de circunstancias.
- Aceptación del Usuario: Los sistemas de Redes Neuronales no pueden proporcionar explicaciones de sus decisiones, y los sistemas basados en reglas deben explicar sus decisiones por referencia a sus reglas, que el usuario no puede entender y aceptar totalmente. Los resultados de los sistemas CBR están basados en casos anteriores reales, que pueden ser presentados al usuario para proporcionar apoyo convincente a las conclusiones del sistema.

El uso exitoso de CBR depende de cómo adquirir, representar, indexar y adaptar los casos existentes.

Involucra cuatro pasos:

3. Realizar la valoración de la situación, para determinar que rasgos de la situación actual son relevantes.
3. Basándose en el paso 1, se recupera un caso anterior relevante o casos anteriores.
3. Se comparan esos casos con la nueva situación para determinar que interpretación se aplica.
3. La situación actual y la interpretación son guardadas como un nuevo caso, en el cual se puede basar un razonamiento futuro.

→ CONECTOR JAVA-OWL

Al no existir un conector específico para el proyecto, se tuvo que investigar como crear un conector que nos permitiese trabajar con las ontologías desde código Java, a través de Jena y Pellet.

Se pudo comprobar como todavía esta tecnología no está muy trabajada por lo que fue un trabajo tedioso que consistió en la documentación en internet, desde páginas web, foros, etc.

Al final se consiguió un conector bastante estable con el que se podía navegar por cualquier ontología, e insertar tanto individuos como los valores para sus propiedades.

3. ANÁLISIS

Actores de la aplicación:

Son los que interactúan con el sistema. Representan todo lo que necesite intercambiar el sistema. En este proyecto los actores son los siguientes:

- Usuario
- Vista
- Ejecutor
- Pizarra
- Modelo
- Razonador DOR
- Razonador DYR
- Razonador USR

Y sus principales Casos de Uso se detallan a continuación.

→ CASOS DE USO DE LA VISTA

CASO DE USO 1	**PROCESAR USR**
Objetivo en contexto	Se encarga de ejecutar la Acción de USR.
Entradas	Botón "Procesar USR" de la ventana interna USR.
Precondiciones	Que el usuario haya introducido una consulta en el campo de texto destinado a tal fin en la ventana USR.
Salidas	Modifica la agenda de la aplicación. Llama al ejecutor.
Poscondición si éxito	Avanza en el resultado de la consulta del usuario.
Poscondición si fallo	Muestra un mensaje indicativo de que el usuario no ha introducido ninguna consulta.
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción**
	1. Pulsar el botón "Procesar USR". 2. Si el usuario ha introducido alguna consulta. 2.1. Agrega a la pizarra el texto introducido por el usuario. 2.2. Añade la tarea "USR" al principio de la agenda. 2.3. Llama a ejecutar las tareas.

CASO DE USO 2	**PROCESAR DOR**
Objetivo en contexto	Se encarga de ejecutar la Acción de DOR.
Entradas	Botón "Procesar DOR" de la ventana interna DOR.
Precondiciones	Que el usuario haya introducido una consulta en el campo de texto destinado a tal fin, que haya contestado alguna pregunta ofrecida por la ventana DOR, o que haya seleccionado alguna de las opciones propuestas.
Salidas	Modifica la agenda de la aplicación. Llama al ejecutor.
Poscondición si éxito	Avanza en el resultado de la consulta del usuario.
Poscondición si fallo	Muestra un mensaje indicativo de que el usuario no ha introducido ninguna consulta, o no ha seleccionado ninguna opción, o no ha rellenado ninguna de las preguntas ofrecidas.
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar el botón "Procesar DOR". 2. Si el usuario ha introducido alguna consulta. 2.1. Deja constancia en la pizarra de que la fase de la ejecución debe comprobar el texto introducido por el usuario. 2.2. Agrega a la pizarra el texto introducido por el usuario. 2.3. Añade la tarea "DOR" al principio de la agenda. 2.4. Llama a ejecutar las tareas. 3. Si había opciones de selección. 3.1. Agrega a la pizarra el texto introducido por el usuario. 3.2. Añade la tarea "DOR" al principio de la agenda. 3.3. Llama a ejecutar las tareas. 4. Si el usuario no ha introducido ningún texto: 4.1. Mostrar mensaje indicando que debe introducir una consulta.

CASO DE USO 3	**PROCESAR DYR**
Objetivo en contexto	Se encarga de ejecutar la Acción de DYR.
Entradas	Botón "Procesar DYR" de la ventana interna DYR.
Precondiciones	Que el usuario haya introducido una consulta en el campo de texto destinado a tal fin en la ventana DYR.
Salidas	Modifica la agenda de la aplicación. Llama al ejecutor.
Poscondición si éxito	Avanza en el resultado de la consulta del usuario.
Poscondición si fallo	Muestra un mensaje indicativo de que el usuario no ha introducido ninguna consulta.
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar el botón "Procesar DYR". 2. Si el usuario ha introducido alguna consulta. 2.1. Agrega a la pizarra el texto introducido por el usuario. 2.2. Añade la tarea "DYR" al principio de la agenda. 2.3. Llama a ejecutar las tareas.

CASO DE USO 4	**PROCESAR COM**
Objetivo en contexto	Se encarga de ejecutar la Acción de COM.
Entradas	Botón "Procesar COM" de la ventana interna COM.
Precondiciones	Que el usuario haya introducido una consulta en el campo de texto destinado a tal fin en la ventana COM.
Salidas	Modifica la agenda de la aplicación. Llama al ejecutor.
Poscondición si éxito	Avanza en el resultado de la consulta del usuario.
Poscondición si fallo	Muestra un mensaje indicativo de que el usuario no ha introducido ninguna consulta.
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar el botón "Procesar COM". 2. Si el usuario ha introducido alguna consulta. 2.1. Deja constancia en la pizarra de que la fase de la ejecución debe comprobar el texto introducido por el usuario. 2.2. Agrega a la pizarra el texto introducido por el usuario. 2.3. Añade la tarea "COM" al principio de la agenda. 2.4. Llama a ejecutar las tareas.

CASO DE USO 5	COMENZAR DE NUEVO
Objetivo en contexto	Comenzar una nueva sesión en la aplicación.
Entradas	Seleccionar la opción "Comenzar de nuevo" dentro del menú Archivo.
Precondiciones	-
Salidas	Resetear las vistas.
Poscondición si éxito	Inicializar las vistas.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Comenzar de nuevo" del menú. 2. Se cierran todas las ventanas de la aplicación. 3. Se eliminan los elementos de los paneles. 4. Se inicializa la vista.

CASO DE USO 6	VOLVER A LA ÚLTIMA SELECCIÓN
Objetivo en contexto	Mostrar la última selección propuesta al usuario.
Entradas	Seleccionar la opción "Volver a la última selección" dentro del menú Herramientas.
Precondiciones	Se mostró una posible selección de múltiple, y el usuario no seleccionó ninguna de las opciones propuestas.
Salidas	Selección múltiple anterior.
Poscondición si éxito	Muestra la selección propuesta anteriormente.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Volver a la última selección" del menú. 2. Se vuelve a mostrar la última selección al usuario.

CASO DE USO 7	MODIFICAR OPCIONES
Objetivo en contexto	Modificar algunas de las opciones de la aplicación.
Entradas	Seleccionar la opción "Opciones…" dentro del menú Archivo.
Precondiciones	-
Salidas	-
Poscondición si éxito	Modifica las opciones seleccionadas por el usuario dentro de la ventana.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Opciones…" del menú. 2. Se muestra la ventana de Opciones de la aplicación. 3. El usuario modifica las opciones deseadas. 4. Pulsar al botón "Aceptar" 5. Graba los cambios seleccionados por el usuario.

CASO DE USO 8	GENERAR PALABRAS IMPORTANTES DE UN ARCHIVO
Objetivo en contexto	Conseguir la lista de palabras importantes del texto incluido en un archivo.
Entradas	Seleccionar la opción "Generar palabras importantes" dentro del menú Herramientas.
Precondiciones	-
Salidas	Dos archivos de texto: uno con la salida tratada (por si el archivo estaba en formato XML) y otro con la lista de las palabras importantes.
Poscondición si éxito	Muestra la opción de volver a procesar otro archivo.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Generar palabras importantes" del menú. 2. Se muestra un cuadro de diálogo donde seleccionar el archivo a procesar. 3. Si el texto tiene formato XML (con etiquetas), se realiza un tratamiento del texto del archivo y guarda el resultado en un archivo de texto con nombre: nombreDelArchivoSeleccionado + salidaTratada.txt 4. Se buscan las palabras importantes del archivo. 5. Se guardan en un archivo de texto con nombre: nombreDelArchivoSeleccionada + palabrasImportantes.txt

CASO DE USO 9	CONSULTAR CONTENIDO DEL ARCHIVO DE LOGs
Objetivo en contexto	Mostrar al usuario el contenido del archivo de LOGs actuales.
Entradas	Seleccionar la opción "Ver contenido del archivo de LOGs" dentro del menú Ver.
Precondiciones	-
Salidas	Ventana de LOGs.
Poscondición si éxito	Muestra los LOGs.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Ver contenido del archivo de LOGs" del menú. 2. Muestra el contenido del archivo de LOGs que tenga la fecha actual.

CASO DE USO 10	MOSTRAR AYUDA
Objetivo en contexto	Mostrar la ayuda referente al manejo de la aplicación.
Entradas	Seleccionar la opción "Ayuda" dentro del menú Ayuda.
Precondiciones	-
Salidas	Ventana de Ayuda.
Poscondición si éxito	Muestra la ayuda de la aplicación.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Ayuda" del menú. 2. Si en el archivo "rutaPDF.sib" se encuentra la ruta correcta a la aplicación PDF instalada en el sistema operativo. 2.1. Arranca la aplicación para PDF y muestra la ayuda de la aplicación en este formato. 3. Si no encuentra la ruta de la aplicación de PDF. 3.1. Muestra un mensaje indicando la posibilidad de incluir la ruta de la aplicación en el archivo: "rutaPDF.sib". 3.2. Muestra la ayuda de la aplicación en formato plano.

CASO DE USO 11	CONSULTAR LOS CRÉDITOS DE LA APLICACIÓN
Objetivo en contexto	Mostrar al usuario los créditos de la aplicación.
Entradas	Seleccionar la opción "Acerca de…" Ayuda.
Precondiciones	-
Salidas	Ventana de Créditos de la aplicación.
Poscondición si éxito	Muestra los créditos.
Poscondición si fallo	Indica que no se ha podido encontrar el archivo con los créditos de la aplicación.
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Acerca de…" del menú. 2. Muestra el contenido del archivo de "Creditos.sib".

CASO DE USO 12	ACTIVA / DESACTIVA DOR
Objetivo en contexto	Visualizar o no, la ventana DOR
Entradas	Seleccionar la opción "Ver DOR" dentro del menú Ver.
Precondiciones	-
Salidas	-
Poscondición si éxito	Hace visible o invisible la ventana DOR.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Ver DOR" del menú. 2. Si la ventana DOR está visible 2.1. Hace invisible la ventana DOR 3. Redimensiona las otras 3 ventanas.

CASO DE USO 13	ACTIVA / DESACTIVA USR
Objetivo en contexto	Visualizar o no, la ventana USR
Entradas	Seleccionar la opción "Ver USR" dentro del menú Ver.
Precondiciones	-
Salidas	-
Poscondición si éxito	Hace visible o invisible la ventana USR.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Ver USR" del menú. 2. Si la ventana USR está visible 2.1. Hace invisible la ventana USR 3. Redimensiona las otras 3 ventanas.

CASO DE USO 14	ACTIVA / DESACTIVA DYR
Objetivo en contexto	Visualizar o no, la ventana DYR
Entradas	Seleccionar la opción "Ver DYR" dentro del menú Ver.
Precondiciones	-
Salidas	-
Poscondición si éxito	Hace visible o invisible la ventana DYR.
Poscondición si fallo	-
Actores	El usuario de la aplicación
Secuencia normal	**Paso Acción** 1. Pulsar la opción "Ver DYR" del menú. 2. Si la ventana DYR está visible 2.1. Hace invisible la ventana DYR 3. Redimensiona las otras 3 ventanas.

→ CASOS DE USO DEL EJECUTOR

CASO DE USO 1	**AÑADIR TAREA AL FINAL DE LA AGENDA**
Objetivo en contexto	Se encarga de añadir una tarea al final de la agenda.
Entradas	-
Precondiciones	Al crear el objeto ejecutor de la aplicación se creó correctamente la agenda. El funcionamiento de la aplicación necesitará la ejecución de una tarea específica.
Salidas	-
Poscondición si éxito	La agenda tiene un elemento más al final de la misma.
Poscondición si fallo	-
Actores	La vista y el modelo de la aplicación
Secuencia normal	**Paso Acción** 1. Se necesita agregar una tarea al final de la agenda. 2. Se incorpora el elemento al final de la agenda.

CASO DE USO 2	**AÑADIR TAREA AL PRINCIPIO DE LA AGENDA**
Objetivo en contexto	Se encarga de añadir una tarea al principio de la agenda.
Entradas	-
Precondiciones	Al crear el objeto ejecutor de la aplicación se creó correctamente la agenda. El funcionamiento de la aplicación necesitará la ejecución de una tarea específica.
Salidas	-
Poscondición si éxito	La agenda tiene un elemento más al principio de la misma.
Poscondición si fallo	-
Actores	La vista y el modelo de la aplicación
Secuencia normal	**Paso Acción** 1. Se necesita agregar una tarea al principio de la agenda. 2. Se incorpora el elemento al principio de la agenda.

CASO DE USO 3	ELIMINAR TAREA DE LA AGENDA
Objetivo en contexto	Se encarga de eliminar una tarea de la agenda.
Entradas	-
Precondiciones	Existe algún elemento en la agenda. El funcionamiento de la aplicación no necesitará la ejecución de una tarea específica.
Salidas	-
Poscondición si éxito	La agenda tiene un elemento menos.
Poscondición si fallo	-
Actores	La vista y el modelo de la aplicación
Secuencia normal	**Paso Acción** 1. Se necesita eliminar una tarea de la agenda. 2. Se elimina el elemento de la agenda.

CASO DE USO 4	EJECUTAR TAREAS
Objetivo en contexto	Se encarga de ejecutar las tareas que se encuentren en la agenda.
Entradas	-
Precondiciones	Existe la agenda.
Salidas	-
Poscondición si éxito	Se ejecutan todas las tareas que se almacenaron en la agenda.
Poscondición si fallo	-
Actores	La vista y el modelo de la aplicación
Secuencia normal	**Paso Acción** 1. Mientras haya tareas en la agenda, va ejecutando las tareas en orden. 2. Va eliminando las tareas de la agenda.

→ CASOS DE USO DE LA PIZARRA

CASO DE USO 1	**AÑADIR ELEMENTO A LA PIZARRA**
Objetivo en contexto	Se encarga de añadir un elemento a la pizarra.
Entradas	-
Precondiciones	El elemento a agregar tiene una posición en la pizarra establecida.
Salidas	-
Poscondición si éxito	La pizarra tiene un elemento en la posición especificada.
Poscondición si fallo	-
Actores	El modelo de la aplicación y el Razonador DOR.
Secuencia normal	**Paso Acción** 1. Se necesita agregar un elemento a la pizarra. 2. Se agrega el elemento deseado en la posición especificada.

CASO DE USO 2	**CONSULTAR ELEMENTO DE LA PIZARRA**
Objetivo en contexto	Se encarga de devolver el elemento deseado.
Entradas	-
Precondiciones	El elemento existe en la pizarra.
Salidas	-
Poscondición si éxito	Devuelve el objeto del elemento.
Poscondición si fallo	-
Actores	El modelo de la aplicación y el Razonador DOR.
Secuencia normal	**Paso Acción** 1. Se necesita un elemento almacenado en la pizarra. 2. Se consulta el objeto de la posición predeterminada para el elemento deseado. 3. Devuelve el objeto.

→ CASOS DE USO DEL MODELO

CASO DE USO 1	**CREAR VENTANAS**
Objetivo en contexto	Crear las ventanas internas de la aplicación.
Entradas	-
Precondiciones	Inicialización de la Vista.
Salidas	Las ventanas internas
Poscondición si éxito	Las ventanas internas son creadas
Poscondición si fallo	
Actores	La vista.
Secuencia normal	**Paso Acción** 1. Creación de las ventas 2. Dimensionado de las ventanas. 3. Inicialización del menú Ver.

CASO DE USO 2	**EJECUTAR DOR**
Objetivo en contexto	Ejecutar las funciones oportunas para el proceso de la ventana DOR.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	Se ejecuta con éxito las funcionalidades de DOR.
Poscondición si fallo	-
Actores	El ejecutor.
Secuencia normal	**Paso Acción** 1. El ejecutor recupera la opción DOR 2. Llama a ejecutar DOR. 3. Si la fase de ejecución es 0 3.1. Ejecuta el inicio sesión, y por tanto uno de los casos base. 4. Si la fase de ejecución es 1 4.1. Analiza la entra del usuario. 4.2. Muestra los posibles casos. 5. Si la fase de ejecución es 2 5.1. Genera el resultado de la opción seleccionada por el usuario. 6. Si la fase de ejecución es 3 6.1. Recoge los valores de las respuestas del usuario.

CASO DE USO 3	**EJECUTAR USR**
Objetivo en contexto	Ejecutar las funciones oportunas para el proceso de la ventana USR.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	Se ejecuta con éxito las funcionalidades de USR.
Poscondición si fallo	-
Actores	El ejecutor.
Secuencia normal	**Paso Acción** 1. El ejecutor recupera la opción USR 2. Llama a ejecutar USR. 3. Llama a ejecutar DOR. 4. Si la fase de ejecución es 0 Pregunta al usuario si está registrado 5. Si la fase de ejecución es 1 5.1 Y usuario no registrado 5.1.1 Toma los datos para registrale en la aplicación 5.2. Y usuario registrado. 5.2.1 Pide que introduzca dni para identificarle. 6. Si la fase de ejecución es 2. 6.1 Y usuario no registrado. 6.1.2. Guarda los datos en la aplicación 6.2. Y usuario registrado. 6.2.1. Extrae los datos del usuario. 6.3 Realiza test para conocer el nivel de conocimiento del usuario. 7. Si la fase de ejecución es 3. 7.1 Evalua las respuestas del test realizado al usuario y le asigna un nivel de experto 8. Si la fase de ejecución es 4. 8.1. Compara los casos que muestra DOR al usuario con los que ha visitado en otras ocasiones. 9. Si la fase de ejecución es 5. 9.1. Almacena el caso que visita. 10. Si la fase de ejecución es 6. 10.1. Guarda todos los datos en la ontología

CASO DE USO 4	EJECUTAR DYR
Objetivo en contexto	Ejecutar las funciones oportunas para el proceso de la ventana DYR.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	Se ejecuta con éxito las funcionalidades de DYR.
Poscondición si fallo	-
Actores	El ejecutor.
Secuencia normal	**Paso Acción** 1. El ejecutor recupera la opción DYR 2. Llama a ejecutar DOR. 3. Si la fase de ejecución es 0 3.1. Muestra estados de ánimo, para que el usuario indique el suyo. 4. Si la fase de ejecución es 1 4.1. Ejecuta un ciclo del razonador y muestra las suggestions. 5. Si la fase de ejecución es 2 5.1. Muestra las questions.. 6. Si la fase de ejecución es 3 6.1. Recoge las respuestas de usuario, las analiza y muestra los estados de ánimo.

CASO DE USO 5	EJECUTAR COM
Objetivo en contexto	Ejecutar las funciones oportunas para el proceso de la ventana COM.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	Se ejecuta con éxito las funcionalidades de COM.
Poscondición si fallo	-
Actores	El ejecutor.
Secuencia normal	**Paso Acción** 1. El ejecutor recupera la opción COM 2. Llama a ejecutar COM.

→ CASOS DE USO DEL CONECTOR JAVAOWL

CASO DE USO 1	**TRADUCE ONTOLOGIA**
Objetivo en contexto	Carga en el conector la ontología indicada.
Entradas	- Nombre de la ontología.
Precondiciones	-
Salidas	- True o False, si la ontología a sido ó no cargada correctamente.
Poscondición si éxito	- Ontología cargada y lista para se consultada.
Poscondición si fallo	- Conector libre de ontología.
Actores	- Razonadores.
Secuencia normal	**Paso Acción** 1. Comprueba que ontología es. 2. Lee la ontología del fichero. 3. Crea los accesos necesarios.

CASO DE USO 2	**GUARDAR ONTOLOGÍA**
Objetivo en contexto	Guarda los nuevos cambios hechos en la ontología.
Entradas	- Nombre de la ontología.
Precondiciones	-
Salidas	-
Poscondición si éxito	- Ontología guardada.
Poscondición si fallo	- Ontología no guardada y excepción lanzada.
Actores	- Razonadores.
Secuencia normal	**Paso Acción** 1. Comprueba la consistencia de la ontología. 2. Si es buena, guarda la ontología en su fichero.

CASO DE USO 3	ACCESO CONCEPTOS
Objetivo en contexto	Inicializa el acceso a todos los conceptos de la ontología.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	- Acceso a los conceptos.
Poscondición si fallo	-
Actores	- Razonadores.
Secuencia normal	**Paso Acción**
	1. Inicia el acceso a los conceptos.

CASO DE USO 4	ACCESO PROPIEDADES
Objetivo en contexto	Inicializa el acceso a todas las propiedades de la ontología.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	- Acceso a las propiedades de la ontología.
Poscondición si fallo	-
Actores	- Razonadores.
Secuencia normal	**Paso Acción**
	1. Inicia el acceso a las propiedades.

CASO DE USO 5	ACCESO INDIVIDUOS
Objetivo en contexto	Inicializa el acceso a todos los individuos de la ontología.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	- Acceso a los individuos de la ontología.
Poscondición si fallo	-
Actores	- Razonadores.
Secuencia normal	**Paso Acción**
	1. Inicia el acceso a los individuos.

CASO DE USO 6	**ACCESO RESTRICCIONES**
Objetivo en contexto	Inicializa el acceso a todas las restricciones de la ontología.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	- Acceso a las restricciones de la ontología.
Poscondición si fallo	-
Actores	- Razonadores.
Secuencia normal	**Paso Acción**
	1. Inicia el acceso a las restricciones.

→ CASOS DE USO DEL RAZONADOR DOR

CASO DE USO 1	**CARGAR ONTOLOGIA**
Objetivo en contexto	Carga la ontología en el conector. Al usar varias ontologías en el proyecto es necesario este método para que cada vez que se quiera usar una determinada ontología primero se cargue.
Entradas	Nombre de la ontología a cargar.
Precondiciones	-
Salidas	-
Poscondición si éxito	La ontología ha sido cargada en el conector para trabajar con ella.
Poscondición si fallo	Se vuelve a la ontología inicial.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	1. Carga la ontología indicada.

CASO DE USO 2	**RESETEAR ONTOLOGÍA**
Objetivo en contexto	Una vez finalizado el trabajo con la ontología necesaria, se debe resetear para que otro razonador pueda trabajar con su ontología.
Entradas	-
Precondiciones	Ontología cargada.
Salidas	-
Poscondición si éxito	El conector queda libre para cargar otra ontología.
Poscondición si fallo	La ontología inicial sigue cargada en el conector.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	1. Resetea el conector y este queda libre.

CASO DE USO 3	EXTRAER CASOS ONTOLOGÍA
Objetivo en contexto	Extrae los casos de la ontología.
Entradas	-
Precondiciones	La ontología existe y no está vacía.
Salidas	-
Poscondición si éxito	Se han creado todos los objetos de la entrada a los casos de la ontología.
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Se consultan los individuos correspondientes a los Input_Slot. 2. Para cada uno de ellos se extrae: el nombre, las palabras importantes, las preguntas de activación

CASO DE USO 4	CONSULTA CASO BASE
Objetivo en contexto	Extrae un caso base de la ontología.
Entradas	-
Precondiciones	La ontología existe y no está vacía.
Salidas	El caso base.
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Se consultan los casos base de la ontología. 2. Si existe más de un caso base 2.1 Escoge uno de los casos base aleatoriamente. 3. Devuelve el caso base.

CASO DE USO 5	CONSULTAR ÚLTIMAS OPCIONES
Objetivo en contexto	Consultar las últimas opciones ofrecidas.
Entradas	-
Precondiciones	Se mostraron opciones al usuario anteriormente.
Salidas	Array con las últimas opciones ofrecidas.
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Se recuperan las últimas opciones mostradas. 2. Devuelve las últimas opciones.

CASO DE USO 6	RAZONAR
Objetivo en contexto	Razona que opciones son las que más se aproximan al texto introducido por el usuario.
Entradas	Consulta introducida por el usuario. Array de las palabras importantes que se encontraban en la consulta del usuario.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Array con las opciones posibles.
Poscondición si éxito	Ultimas opciones actualizadas
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Comprueba las preguntas de activación existentes. 2. Si la pregunta del usuario se corresponde con una pregunta de activación. 2.1 Devuelve las opciones del caso correspondiente 2.2 Salta al punto 6. 3. Busca los casos existentes que tengan relación con las palabras importantes 4. Ordena los casos encontrados según el número de coincidencias (de más a menos). 5. Extrae las opciones de los casos ordenados. 6. Quita las opciones con el mismo identificador. 7. Devuelve las opciones

CASO DE USO 7	CONSULTAR DESCRIPCIÓN TEXTUAL
Objetivo en contexto	Consulta la descripción textual de un caso.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Descripción textual
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Consulta en la ontología la descripción textual de un caso en concreto. 2. Devuelve la descripción

CASO DE USO 8	CONSULTAR DESCRIPCIÓN GRÁFICA
Objetivo en contexto	Consulta la descripción gráfica de un caso.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Descripción gráfica
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Consulta en la ontología la descripción gráfica de un caso en concreto. 2. Devuelve la descripción

CASO DE USO 9	CONSULTAR PREGUNTAS INTENCIONALES
Objetivo en contexto	Consulta las Intentional Question de un caso.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Array con las preguntas.
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Consulta en la ontología las preguntas intencionales de un caso en concreto. 2. Devuelve las preguntas.

CASO DE USO 10	CONSULTAR SUGERENCIAS
Objetivo en contexto	Consulta las sugerencias de un caso.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Sugerencias
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Consulta en la ontología las sugerencias de un caso en concreto. 2. Devuelve las sugerencias

CASO DE USO 11	PROCESAR CASOS CON PARES VALORES
Objetivo en contexto	Extraer los pares concepto-valor de cada caso y procesarlos para sacar el caso más probable.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	-
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Extrae los casos a comprobar. (Aquellos de las últimas opciones, que tenían el mismo identificador) 2. Comprueba los datos existentes en la pizarra para cada concepto, de los casos extraídos. 3. Ordena de mayor a menor probabilidad los casos.

CASO DE USO 12	COMPROBAR SI PRIMER CASO TIENE TODOS LOS VALORES
Objetivo en contexto	Comprueba si el primer caso de los procesados por pares valores, tiene todos sus valores en la pizarra y concuerdan.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Verdadero: si tiene todos los valores Falso: si no los tiene
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Comprueba si el primer caso tiene todos sus valores coincidentes. 2. Si tiene todos sus valores 2.1 Devuelve Verdadero 3. Sino Devuelve Falso

CASO DE USO 13	CONSULTAR PRIMER CASO
Objetivo en contexto	Consulta el primer caso de los procesados por pares valores.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	El primer caso.
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Consulta el primer caso de los procesados 2. Devuelve el caso.

CASO DE USO 14	CONSULTAR PREGUNTAS A REALIZAR
Objetivo en contexto	Consulta las preguntas a realizar.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Preguntas
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Para todos los casos relacionados con el caso seleccionado por el usuario. 2. Comprueba si cada uno de ellos tiene valores que no coincidan con los datos del usuario. 2.1 Si alguno de ellos tiene algún valor que no coincida. Se quita de la lista 2.1 Sino, 2.1.2 Consulta los conceptos de los que no tiene valores establecidos. 2.2.2 Devuelve las preguntas oportunas.

CASO DE USO 15	COMPROBAR SI EXISTE OTRO CASO SIMILAR
Objetivo en contexto	Comprueba si en las últimas opciones recuperadas existe otro caso que tenga el mismo identificador, que el caso seleccionado por el usuario.
Entradas	Selección del usuario de un caso en concreto.
Precondiciones	El usuario ha realizado una nueva consulta.
Salidas	Verdadero: si existe otro caso Falso: si no existe
Poscondición si éxito	-
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Comprueba si existe otro caso similar en las últimas opciones recuperadas. 2. Devuelve verdadero si existe. 3. Sino devuelve falso.

CASO DE USO 16	CAMBIAR NÚMERO DE OPCIONES
Objetivo en contexto	Cambia el número de opciones a mostrar inicialmente al usuario.
Entradas	Modificación de las opciones de la aplicación.
Precondiciones	El usuario ha modificado la opción del número de opciones a mostrar
Salidas	-
Poscondición si éxito	El número de opciones a mostrar es modificado.
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción**
	1. Establece el nuevo número de opciones a mostrar.

CASO DE USO 17	MODIFICAR ONTOLOGÍA
Objetivo en contexto	Modifica la ontología que se está utilizando en el razonamiento de la aplicación.
Entradas	Modificación de las opciones de la aplicación.
Precondiciones	El usuario ha modificado la ontología a utilizar.
Salidas	-
Poscondición si éxito	La ontología ha cambiado a la deseada por el usuario.
Poscondición si fallo	Se vuelve a la ontología inicial.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	1. Establece la nueva ontología. 2. Extrae los casos de la nueva ontología

→ CASOS DE USO DEL RAZONADOR DYR

CASO DE USO 1	**CARGAR ONTOLOGIA**
Objetivo en contexto	Carga la ontología en el conector. Al usar varias ontologías en el proyecto es necesario este método para que cada vez que se quiera usar una determinada ontología primero se cargue.
Entradas	Nombre de la ontología a cargar.
Precondiciones	-
Salidas	-
Poscondición si éxito	La ontología ha sido cargada en el conector para trabajar con ella.
Poscondición si fallo	Se vuelve a la ontología inicial.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	2. Carga la ontología indicada.

CASO DE USO 2	**RESETEAR ONTOLOGÍA**
Objetivo en contexto	Una vez finalizado el trabajo con la ontología necesaria, se debe resetear para que otro razonador pueda trabajar con su ontología.
Entradas	-
Precondiciones	Ontología cargada.
Salidas	-
Poscondición si éxito	El conector queda libre para cargar otra ontología.
Poscondición si fallo	La ontología inicial sigue cargada en el conector.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	2. Resetea el conector y este queda libre.

CASO DE USO 3	ACTIVAR AUTOMÁTICO
Objetivo en contexto	Activa la ejecución automática del razonador DYR, para evitar la iteración con el usuario y facilitarle el trabajo.
Entradas	- Ejecutor
Precondiciones	-
Salidas	-
Poscondición si éxito	- Ejecución automática activada.
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Inicializa el reloj de la ejecución automática. 2. Inicia el proceso que controla la ejecución automática.

CASO DE USO 4	RESTART AUTOMÁTICO
Objetivo en contexto	Resetea el reloj de la ejecución automática y lo coloca a cero.
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	- Inicializa el reloj de la ejecución automática.
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 1. Coloca a cero el reloj de la ejecución automática.

CASO DE USO 5	CONTROL NUMWORSENCYCLES
Objetivo en contexto	Controla el aumento o disminución del número de ciclos que el usuario lleva empeorando su estado de ánimo.
Entradas	- Puntuación del estado actual y nivel de empeoramiento actual
Precondiciones	-
Salidas	- Aumento o disminución del número de ciclos de empeoramiento
Poscondición si éxito	- Aumento o disminución del número de ciclos de empeoramiento
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Consulta los estados de ánimo de la ontología. 2. Identifica los mejores y los peores estados de ánimo. 3. Identifica en donde se encuentra el estado de ánimo actual del usuario. 4. Asigna el aumento o disminución que corresponda.

CASO DE USO 6	AJUSTAR URGENCYDEGREE
Objetivo en contexto	Sirve para encontrar el grado de urgencia que mas se ajusta en cada momento.
Entradas	- Individuo, tiempo de ejecución, ciclos de ejecución.
Precondiciones	-
Salidas	- Grado de urgencia que más se ajusta en el momento actual.
Poscondición si éxito	- Grado de urgencia asignado.
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Identifica el tiempo de ejecución actual en la ontología. 2. Identifica el número de ciclos ejecutado en la ontología. 3. Busca el grado de urgencia que más se ajuste al estado actual.

CASO DE USO 7	ANALIZAR RESPUESTAS
Objetivo en contexto	Analiza el beneficio obtenido de las respuestas del usuario.
Entradas	- Respuestas del usuario.
Precondiciones	-
Salidas	- Aumento o disminución que provocan las respuestas del usuario sobre el número de ciclos empeorando.
Poscondición si éxito	- Aumento asignado.
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Análisis sencillo de las respuestas. 2. Si es una respuesta positiva reduce en número de ciclos empeorando. 3. Si es una respuesta negativa aumenta el número de ciclos empeorando.

CASO DE USO 8	GET FACES
Objetivo en contexto	Devuelve las caras que representan los estados de ánimo conocidos por la ontología.
Entradas	-
Precondiciones	-
Salidas	- Lista con el path de las imágenes que representa las caras de los estados de ánimo.
Poscondición si éxito	- Caras de representación de los estados de ánimo.
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Lista las caras que contiene la ontología y las devuelve en un ArrayList.

CASO DE USO 9	GET CYCLES NUMBER
Objetivo en contexto	Recupera el individuo de la ontología que mejor represente el tiempo que lleva ejecutándose la aplicación
Entradas	-
Precondiciones	-
Salidas	- Individuo que representa el tiempo que lleva ejecutándose la aplicación.
Poscondición si éxito	- Individuo que representa el tiempo que lleva ejecutándose la aplicación.
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Obtiene una lista con los individuos conocidos por la ontología. 2. Analiza y busca el que más se ajuste en este momento.

CASO DE USO 10	GET CYCLES NUMBER
Objetivo en contexto	Recupera el individuo de la ontología que mejor represente el número de ciclos que lleva ejecutados el razonador DYR.
Entradas	-
Precondiciones	-
Salidas	- Individuo que representa el número de ciclos que lleva ejecutados el razonador DYR.
Poscondición si éxito	- Individuo que representa el número de ciclos que lleva ejecutados el razonador DYR.
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Obtiene una lista con los individuos conocidos por la ontología. 2. Analiza y busca el que más se ajuste en este momento.

CASO DE USO 11	**EJECUTAR**
Objetivo en contexto	Ejecuta un ciclo del razonador DYR.
Entradas	-
Precondiciones	-
Salidas	- Resultados de ejecutar un ciclo del razonado DYR.
Poscondición si éxito	- Resultados de ejecutar un ciclo del razonado DYR.
Poscondición si fallo	-
Actores	Razonador DYR.
Secuencia normal	**Paso Acción** 1. Asigna el estado de ánimo. 2. Asigna el nivel de empeoramiento. 3. Asigna el grado de urgencia. 4. Asigna el número de ciclos empeorando. 5. Ejecuta la acción de diagnóstico. 6. Ejecuta la acción de Predicción. 7. Ejecuta la acción de Planificación. 8. Guarda los nuevos datos en la pizarra. 9. Devuelve los resultados.

→ CASOS DE USO DEL RAZONADOR USR

CASO DE USO 1	**CARGAR ONTOLOGIA**
Objetivo en contexto	Carga la ontología en el conector. Al usar varias ontologías en el proyecto es necesario este método para que cada vez que se quiera usar una determinada ontología primero se cargue.
Entradas	Nombre de la ontología a cargar.
Precondiciones	-
Salidas	-
Poscondición si éxito	La ontología ha sido cargada en el conector para trabajar con ella.
Poscondición si fallo	Se vuelve a la ontología inicial.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	1. Carga la ontología indicada.

CASO DE USO 2	**RESETEAR ONTOLOGÍA**
Objetivo en contexto	Una vez finalizado el trabajo con la ontología necesaria, se debe resetear para que otro razonador pueda trabajar con su ontología.
Entradas	-
Precondiciones	Ontología cargada.
Salidas	-
Poscondición si éxito	El conector queda libre para cargar otra ontología.
Poscondición si fallo	La ontología inicial sigue cargada en el conector.
Actores	El modelo.
Secuencia normal	**Paso Acción**
	1. Resetea el conector y este queda libre.

CASO DE USO 3	**ESTRAER DATOS ONTOLOGÍA**
Objetivo en contexto	Extrae los datos de la ontología.
Entradas	-
Precondiciones	La ontología existe y no está vacía.
Salidas	-
Poscondición si éxito	Se han creado todos los objetos de la entrada a los casos de la ontología.
Poscondición si fallo	-
Actores	El modelo.
Secuencia normal	**Paso Acción** 2. Se extraen los individuos de User. 3. Se extraen los individuos de Personal_Data 4. Se extraen los individuos de Sesion_Data.

CASO DE USO 4	**COMPROBAR USUARIO**
Objetivo en contexto	Obtiene si un usuario está registrado en la aplicación.
Entradas	Dni del usuario.
Precondiciones	El dni introducido es correcto
Salidas	Existencia o no del usuario en la ontología
Poscondición si éxito	True
Poscondición si fallo	False
Actores	El modelo
Secuencia normal	**Paso Acción** 1. Obtiene el dni de cada usuario de la ontología. 2. Si es igual al introducido por el usuario, para y devuelve True. 3. Si no es igual, sigue buscando hasta que lo encuentre o hasta que haya visitado todos los usuario,

CASO DE USO 5	OBTENER NIVEL DE EXPERTO
Objetivo en contexto	Conocer el nivel de experiencia del usuario sobre la aplicación
Entradas	-
Precondiciones	Usuario registrado
Salidas	Nivel de experto
Poscondición si éxito	Nivel de experto que posee el usuario
Poscondición si fallo	-
Actores	El modelo
Secuencia normal	**Paso Acción** 1. Recorre el array de usuarios de la aplicación hasta que encuentre el usuario que se corresponde con el dni que introdujo el usuario o hasta que se acabe. 2. Obtiene el nivel de experto que le corresponde al usuario.

CASO DE USO 6	GUARDAR DATOS PERSONALES
Objetivo en contexto	Almacenar los datos personales de un usuario nuevo en la ontología.
Entradas	Datos personales del usuario nuevo
Precondiciones	-
Salidas	-
Poscondición si éxito	True y datos personales guardados en la ontología.
Poscondición si fallo	False y datos personales no guardados en la ontología
Actores	Razonador USR
Secuencia normal	**Paso Acción** 1. Crea un objeto con todos los datos personales del usuario nuevo 2. Guarda en ontología.

CASO DE USO 7	GUARDA DATOS SESIONES
Objetivo en contexto	Alamcena los datos de la sesiones de un usuario.
Entradas	Datos del usuario.
Precondiciones	-
Salidas	-
Poscondición si éxito	True y datos de sesiones guardados en la ontología.
Poscondición si fallo	False y datos de sesiones no guardados en la ontología
Actores	Razonador USR
Secuencia normal	**Paso Acción** 1. Crea un objeto con los datos de la sesión del usuario. 2. Guarda en la ontología.

CASO DE USO 8	GUARDAR USUARIO
Objetivo en contexto	Alamcenar un usuario en la ontología.
Entradas	Usuario nuevo
Precondiciones	-
Salidas	-
Poscondición si éxito	True y usuario guardado en la ontología.
Poscondición si fallo	False y usuario no guardado en la ontología.
Actores	Razonador USR
Secuencia normal	**Paso Acción** 1. Crea un objeto con los datos que necesita el usuario 2. Guarda en ontología.

CASO DE USO 9	CREA USUARIO
Objetivo en contexto	Crear un usuario nuevo en la ontología
Entradas	-
Precondiciones	-
Salidas	-
Poscondición si éxito	Datos personales creados y guardados. Datos de Sesiones creados y guardados. Usuario creado y guardado.
Poscondición si fallo	Objetos no creados ni guardados.
Actores	El modelo
Secuencia normal	**Paso Acción** 1. Crea el objeto Datos Personales. 2. Crea el objeto Datos de Sesion 3. Crea el objeto Usuario y le asigna los objetos Sesion y Datos Personales. 4. Guarda Datos Personales en la ontología. 5. Guarda Sesion en la ontología 6. Guarda Usuario en la ontología.

CASO DE USO 10	RECUPERA PERSONAL DATA
Objetivo en contexto	Recuperar los datos personales del usuario.
Entradas	Nombre del Personal Data a reuperar.
Precondiciones	Que esxisa ese Personal Data.
Salidas	Objeto Personal Data recuperado
Poscondición si éxito	-
Poscondición si fallo	-
Actores	Razonador USR.
Secuencia normal	**Paso Acción** 1. Recupera los datos personales que se corresponden con el nombre de usuario pasado por parámetro. 2. Crea un objeto de datos personales.

CASO DE USO 11	RECUPERA USUARIO CREADO
Objetivo en contexto	Recupera los datos de un usuario.
Entradas	Nombre del usuario a recuperar.
Precondiciones	Usuario creado.
Salidas	Usuario recuperado.
Poscondición si éxito	-
Poscondición si fallo	-
Actores	Razonador USR
Secuencia normal	**Paso Acción** 1. Recupera los datos que se corresponden con el nombre de usuario pasado como parámetro. 2. Crea un objeto usuario.

CASO DE USO 12	RECUPERA SESION
Objetivo en contexto	Recuperar una sesion
Entradas	Nombre de una sesion pasada como parámetro
Precondiciones	Secion creada.
Salidas	Sesión.
Poscondición si éxito	-
Poscondición si fallo	-
Actores	Razonador USR
Secuencia normal	**Paso Acción** 1. Recupera los datos que se corresponden con el nombre de sesión pasado como parámetro. 2. Crea un objeto Sesion.

CASO DE USO 13	**GUARDAR SESION EJECUTADA**
Objetivo en contexto	Guarda la sesión recién ejecutada.
Entradas	-
Precondiciones	Usuario creado
Salidas	-
Poscondición si éxito	Sesion almacenada en el usuario
Poscondición si fallo	-
Actores	Razonador USR.
Secuencia normal	**Paso Acción** 1. Recoge los datos de la sesión que se ha ejecutado en el ciclo actual. 2. Crea un objeto Sesion. 3. Crea un objeti Usuario. 4. Asigna al usuario el nivel de experto y la sesión. 5. Guarda en la ontología

4. DISEÑO E IMPLEMENTACIÓN

4.1- TECNOLOGÍAS:

→ ECLIPSE

Una IDE puede hacer el trabajo mucho más sencillo, sobretodo si el desarrollo ya va manejando un buen número de Clases. Además estos entornos permiten mucha más versatilidad para depurar programas puesto que tienen debbugers mucho más avanzados.

Eclipse es una IDE multiplataforma libre para crear aplicaciones clientes de cualquier tipo. La primera y más importante aplicación que ha sido realizada con este entorno es la afamado IDE Java llamado Java Development Toolkit (JDT) y el compilador incluido en Eclipse, que se usaron para desarrollar el propio Eclipse.

Eclipse fue producto de una inversión de cuarenta millones de dólares de IBM en su desarrollo antes de ofrecerlo como un producto de código abierto al consorcio Eclipse.org que estaba compuesto inicialmente por Borland e IBM. IBM sigue dirigiendo el desarrollo de Eclipse a través de su subsidiaria OTI (Object Technologies International), creadora de Eclipse. La Fundación Eclipse, que actualmente lo desarrolla, es una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

El proyecto Eclipse se divide en tres subproyectos:

- *El Core de la aplicación*, que incluye el subsistema de ayuda, la plataforma para trabajo colaborativo, el Workbench (construido sobre SWT y JFace) y el Workspace para gestionar proyectos.
- *Java Development Toolkit (JDT)*, donde la contribución de Erich Gamma ha sido fundamental.
- *Plug-in Development Enviroment (PDE)*, que proporciona las herramientas para el desarrollo de nuevos módulos.

El entorno integrado de desarrollo (IDE) de Eclipse emplea módulos para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. El mecanismo de módulos permite que el entorno de desarrollo soporte otros lenguajes además de Java.

Pese a que Eclipse está escrito en su mayor parte en Java (salvo el núcleo), se ejecute sobre máquina virtual de ésta y su uso más popular sea como un IDE para Java, Eclipse es neutral y adaptable a cualquier tipo de lenguaje, por ejemplo C/C++, Cobol, C#, XML, etc.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".

La característica clave es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. La forma en que los plug-ins interactúan es mediante interfaces o puntos de extensión; así, las nuevas aportaciones se integran sin dificultad ni conflictos.

Un plug-in es la mínima unidad de la plataforma que puede ser desarrollado por Separado y que la aporta una nueva funcionalidad. Los hay freeware y de pago; incluso puedes programar uno tú mismo. Se instalan descomprimiendo el zip del plug-in en el directorio plugins de Eclipse.

Las características principales del IDE son las siguientes:

- Permite trabajar con varios proyectos a la vez
- El editor de código tiene colores para la sintaxis y también lo que es conocido como "code highlighting".
- Los errores de compilación a parte de darte una descripción de error te indica donde se ha producido el error en los márgenes
- Formateador de código.
- Permite encontrar código duplicado.
- Tiene lo que se conoce como "code folding".
- Permite personalizar el entorno.
- El editor tiene intellisense
- Cuando estás escribiendo te permite terminar de completar el código.
- Puedes ver el javadoc del api del jdk que se esté utilizando en ese momento.
- Refactorización del código.
- El editor permite buscar y reemplazar palabras.
- Integración con aplicaciones controladoras de versión como por ejemplo CVS.
- Permite compara archivos.
- Mantiene una historia local de los archivos de la que se pueden restaurar.
- En la compilación a parte de la generación de las classes (*.class) permite una compilación incremental.
- Permite el uso de herramientas externas como es ANT o JUNIT.
- Para el debug de los programas es compatible con JPDA.
- Permite codificar código java, c/c++, xml, jsp, html, etc.

→ JAVA

Java es un lenguaje de propósito general, orientado a objetos, y con una fuerte inclinación hacia el desarrollo de entornos distribuidos y entornos gráficos. Aunque cada día crece más y se le añaden nuevas posibilidades, por lo que ya es un lenguaje apto para cualquier tipo de desarrollo.

Las fechas más importantes de la historia de Java, según Patrick Naughton, uno de los creadores originales del navegador HotJava, el primer intérprete de applets de Java, son las siguientes:

- Junio 1991. James Gosling comienza a trabajar en el interprete "Oak" el cual años después pasaría a conocerse como Java
- Abril 1993. Aparece NCSA Mosaic 1.0, el primer navegador gráfico para Internet
- Junio 1994. Nace el proyecto "Liveoak" cuyo objetivo era usar Oak en el diseño de un pequeño sistema operativo
- Julio 1994. Naughton replantea el proyecto "Liveoak" como una adaptación de Oak a Internet, realizando una primera implementación de un navegador de Web con soporte para Oak.
- 16 de Septiembre 1994. Payne y Naughton comienzan a escribir "WebRunner", un navegador similar a Mosaic al que más tarde se le llamaría "HotJava"
- 29 de Septiembre 1994. El primer prototipo de HotJava es presentado a la junta ejecutiva de Sun.
- Otoño 1994. Van Hoff implementa el compilador de Java en Java (Gosling lo había implementado en C)
- 23 de Mayo 1995. Sun anuncia formalmente Java y HotJava en el SunWorld '95. Ahondando un poco más en la historia, todo comenzó cuando en SUN, parte de su departamento software se reveló ante un proceso de desarrollo software que consideraban caótico. Lograron que en SUN se les dejara trabajar juntos y formaron un grupo de seis personas (entre las que figuraban todas las personas referidas en la cronología) y comenzaron a estudiar como crear algo nuevo y espectacular, condiciones exigidas por Sun para permitir el grupo.

Comenzaron a estudiar como se podían comunicar diferentes equipos electrónicos de consumo, y vieron que cada uno de ellos tenía diferentes CPUs, por lo que cada fabricante debía de crear soluciones independientes para añadir funcionalidad a sus productos, algo además complicado ya que la mayoría de la lógica estaba cableada dentro del producto. Por ello pensaron en desarrollar un sencillo lenguaje de

programación (Oak) orientado a objetos, y que no fuera descabellado introducir en este tipo de electrónica de consumo. Analizaron con intensidad los aspectos visuales de las aplicaciones que atraían a la gente y como las personas interactuaban con las máquinas. Con el resultado de sus estudios realizaron un prototipo llamado "*7" en el que ya aparecía la mascota Duke de Java, y cuyo objetivo de diseño fue que su interfaz fuera amigable y divertida, y el dispositivo en sí debía de ser un pequeño artefacto personal. Este dispositivo es en cierto modo la máquina virtual que hoy nos encontramos en el interior de todos nuestros navegadores, y el lenguaje interpretado Oak es lo que hoy conocemos como Java. Analizando estos orígenes es sencillo descubrir porqué Java es un lenguaje de programación mucho más reducido que C++, y porqué uno de sus criterios fue la sencillez. Y es más, la idea original de meter Java en los dispositivos electrónicos no ha sido ni mucho menos abandonada. Java se esta introduciendo en teléfonos, televisores, tarjetas inteligentes e incluso anillos.

Java ha necesitado de 4 años para su desarrollo, y su objetivo original no era ni mucho menos Internet. Inicialmente el interprete de Java junto con la máquina virtual Java tenían como objetivo formar un sistema operativo de pequeñas dimensiones y pocos recursos, que pudiera ser introducido en todo tipo de aparatos electrónicos de consumo, tal y como aparecía en Junio de 1994. Pero coincidió que en ese cierto momento se produjo la explosión de Internet, y la estrategia se cambio hacia el uso de esta pequeño máquina virtual y el lenguaje Java a Internet.

Llegaron las applets a Internet revolucionando las hasta entonces páginas estáticas del WWW, hechas íntegramente en HTML. Al comienzo dichas applets no hacían más que cosas sencillas y vistosas, pero poco a poco se fue utilizando la mayor potencia que pone a nuestro servicio Java, comenzando una nueva era de aplicaciones en Internet, aplicaciones portables a lo largo de muchas plataformas, y que proporcionan una gran variedad de servicios.

Java es un conjunto de tecnologías, no es solo un lenguaje. De dichas tecnologías destacan las siguientes características:

- Es un lenguaje de programación cuyos principios fundamentales son los de Smalltalk pero con una sintaxis más cercana al C++. Sus objetivos de diseño fueron ser multiplataforma, distribuido, y orientado a las redes de comunicación.
- Java tiene un entorno completo que lo recubre, ya que librerías como la de entornos gráficos o la red, están incluidas dentro del entorno de ejecución de Java.

- Java es un lenguaje interpretado, lo que le hace independiente de la plataforma de ejecución, aunque menos eficiente que los lenguajes clásicos compilados.
- Existe un entorno de ejecución común, conocido como máquina virtual Java.
- Es independiente de la plataforma.
- Tiene un amplio soporte para Internet, gracias a su librería TCP/IP de red.
- Tiene un modelo de componentes muy completo conocido como JavaBeans, diseñado desde los comienzos de Java.

La máquina virtual Java (JVM) fue pensada como un sistema muy sencillo, ya que debía ser albergado en dispositivos electrónicos de consumo.

La JVM es un ordenador abstracto que es capaz de ejecutar programas Java compilados. El término virtual es debido a que inicialmente no existían arquitecturas reales Java, por lo la JVM se solía implementar por software sobre plataformas ya existentes. Pero poco a poco van apareciendo los chips Java, que darán apoyo a la aparición de máquinas "reales" Java.

Como es de esperar al ser una máquina virtual, el rendimiento que logra en la ejecución del código Java está muy alejado de las aplicaciones desarrolladas para una plataforma específicamente, pero este factor que es muy importante, se ve compensado por otro factor aún más importante: la portabilidad.

Debido a la sencillez de diseño de la JVM, rápidamente se ha extendido a todas las arquitecturas existentes, logrando una capa software común para todas ellas.

Desde sus orígenes el entorno de desarrollo de Java más utilizado ha sido el proporcionado por JavaSoft, conocido como JDK: Java Development Kit.

Este es un entorno más pensado para analizar las características de Java, y aprender su utilización, que para un desarrollo comercial de programas Java. Pero hasta el momento no existe ningún entorno de desarrollo lo suficientemente evolucionado como para que merezca la pena invertir tiempo y dinero en él. El JDK se puede obtener de forma gratuita y es el único entorno de desarrollo que siempre esta al día, y con la velocidad que aún hoy cambia Java, se hace muchas veces imprescindible utilizar la última versión, ya que los cambios son tan significativos, que el no utilizarla podría dejar nuestras aplicaciones obsoletas incluso antes de haber llegado a la fase de producción.

Algunas de las herramientas que se incluyen dentro del JDK, son:

- Compilador de Java (javac) → Compila programas escritos en el lenguaje de programación Java convirtiéndolos en bytecodes
- Interprete de Java (java) → Ejecuta los bytecodes Java, es decir, ejecuta los programas Java
- Interprete de ejecución de Java (jre) → Similar a java, pero simplificado para los usuarios que no necesiten herramientas de desarrollo en para Java.
- Java AppletViewer (appletviewer) (Herramienta para depurar y ejecutar applets
- Depurador de Java (jdb) (Herramienta para encontrar los fallos dentro de los programas Java
- Desensamblador de ficheros de clases (javap) (Herramienta que a partir de bytecode, genera ficheros de código en Java
- Generador de documentación Java (javadoc) (Analiza las declaraciones y los comentarios que están dentro de un conjunto de ficheros fuente Java y genera un conjunto de páginas HTML describiendo las clases "public" y "protected", las interfaces, los métodos, los constructores y los campos. También genera una jerarquía de clases y un índice con todos los miembros de la aplicación.
- Generador de cabeceras y cabos para C (javah) (Para utilizar métodos nativos en lenguaje C desde Java
- Herramienta de archivación Java (jar) (Combina varias clases Java y sus recursos asociados en un único fichero
- Herramienta de firmas digitales (javakey) (Gestiona entidades, incluyendo sus claves y certificados
- Convertidor Native-To-ASCII (native2ascii) (Convierte un fichero con codificación nativa a un fichero ascii en el que los caracteres no ascii aparecen con la notación de Unicode \udddd
- Generador de cabos para aplicaciones Java RMI (rmic)
- Registro de Objetos Remotos Java (rmiregistry) (Servidor de directorio para objetos de RMI
- Comando para control de versiones (serialver) (Proporciona el "serialVersionUID" para una o más clases en un formato adecuado para copiarlo en clases en desarrollo
- Utilidad de conversión AWT 1.1 (updateAWT) (Actualiza los nombres que han cambiado en los métodos de AWT 1.02 en una aplicación.

Tras cada versión aparecen nuevas herramientas, nuevas clases dentro de las librerías.

La instalación del entorno de desarrollo es muy sencilla. Tan solo hay que incluir en la ruta principal (path) la ruta para llegar a los binarios de JDK, y hay que definir la variable de entorno "CLASSPATH", indicando a la JVM donde se encuentran las clases dentro del sistema.

→ JAVA MULTIPLATAFORMA

Java es un conjunto de tecnologías que a grandes rasgos, son independientes de la arquitectura, por lo que en principio, no nos importa que desarrollemos en plataformas Linux, Solaris, HP-UX o Windows 95 y NT. El desarrollador tiene pues la libertad de elegir la plataforma que más se adecue a sus necesidades, y en la que se sienta más a gusto.

Desde sus inicios, las herramientas de desarrollo de Java estuvieron disponibles para Linux, y es más, se incluyó soporte en el núcleo de Linux para reconocer de forma automática a las clases de Java y ejecutarlas.

4.2- ARQUITECTURA SOFTWARE:

ESQUEMA

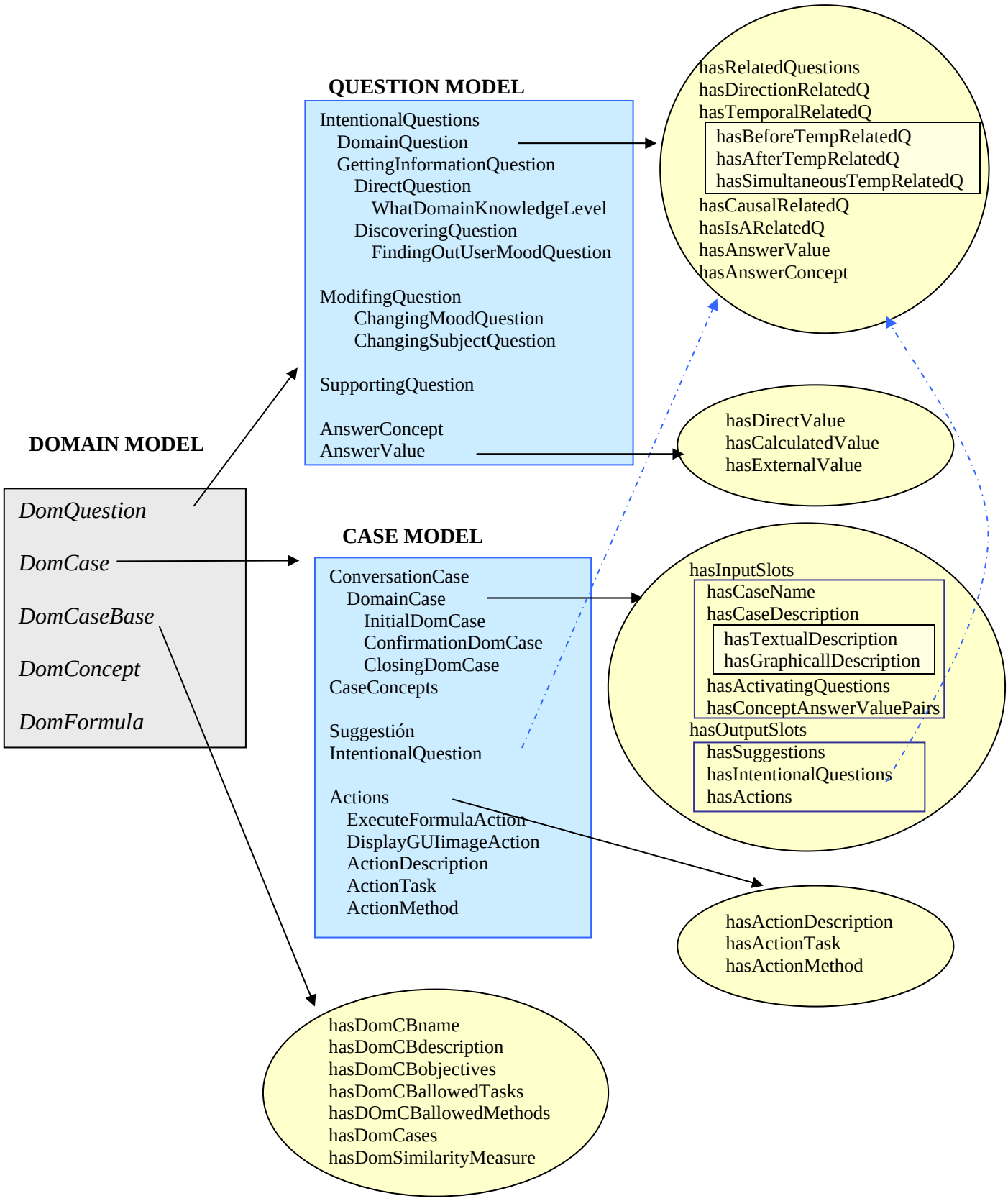
El esquema de la aplicación está basado en tres ontologías diferentes, cada una específica de uno de los razonadores (DOROnto, DYROnto y USROnto). Todas ellas comparten una ontología común, CCBROnto.

CCBROnto

Se ha definido CCBROnto, una ontología que incluye el esqueleto donde se pueden definir características adicionales del sistema CCBR.

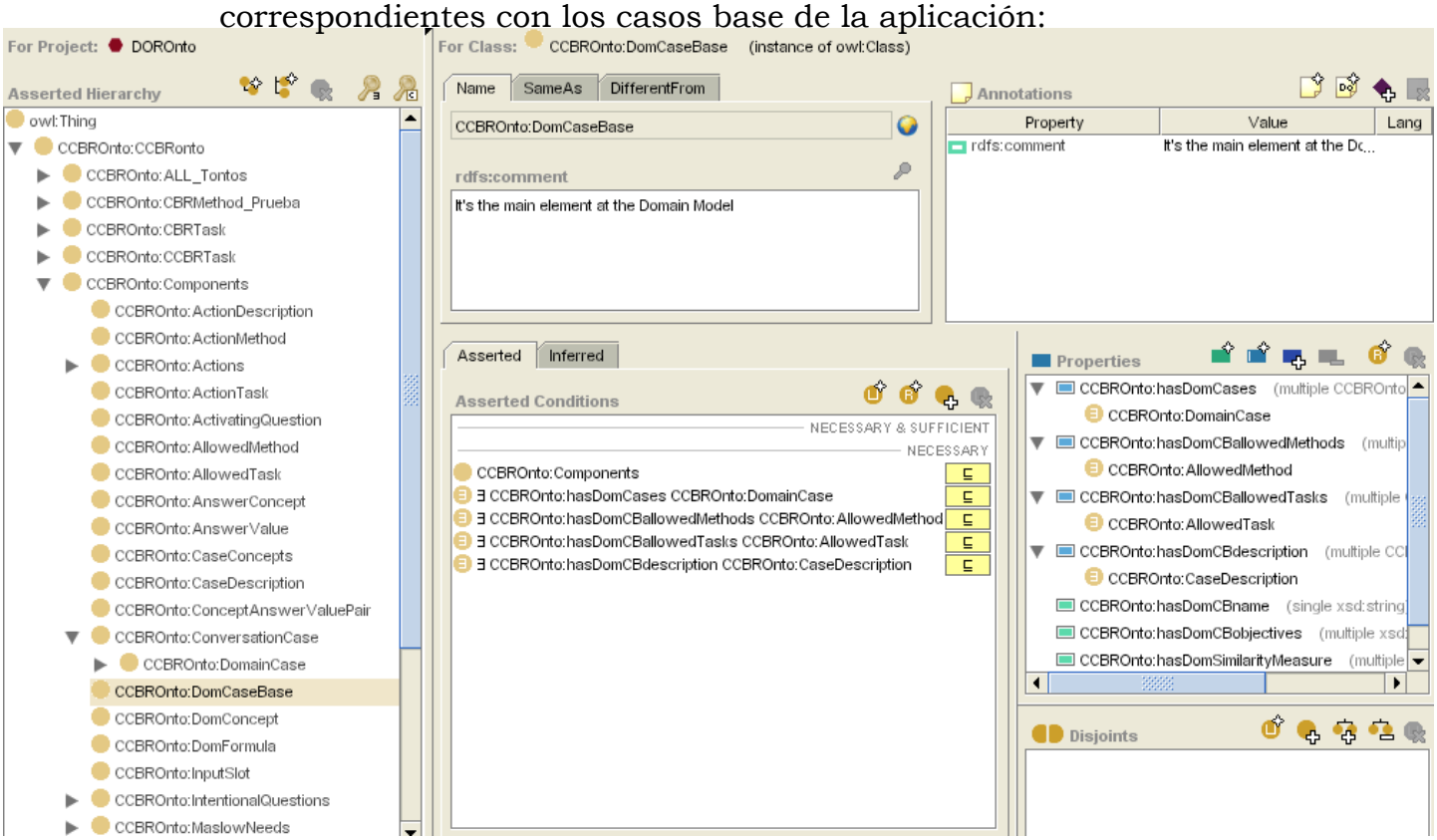
El modelo del dominio define todo el conocimiento sobre el dominio e incluye la pregunta y los modelos de casos. El esqueleto de elementos más relevante del dominio del modelo son: descripción del dominio, base de casos del dominio, caso del dominio y un concepto genérico. El más importante de ellos es DomCaseBase.

En la página siguiente se muestra un esquema del CCBROnto utilizado, donde los cuadros azules contienen las clases y las circunferencias amarillas contienen las propiedades de las clases. Los cuadrados dentro de las circunferencias amarillas, simbolizan que las propiedades que se encuentran dentro de los mismos, pertenecen a la clase a la que está ligada la propiedad que se encuentra en el borde superior del cuadrado.

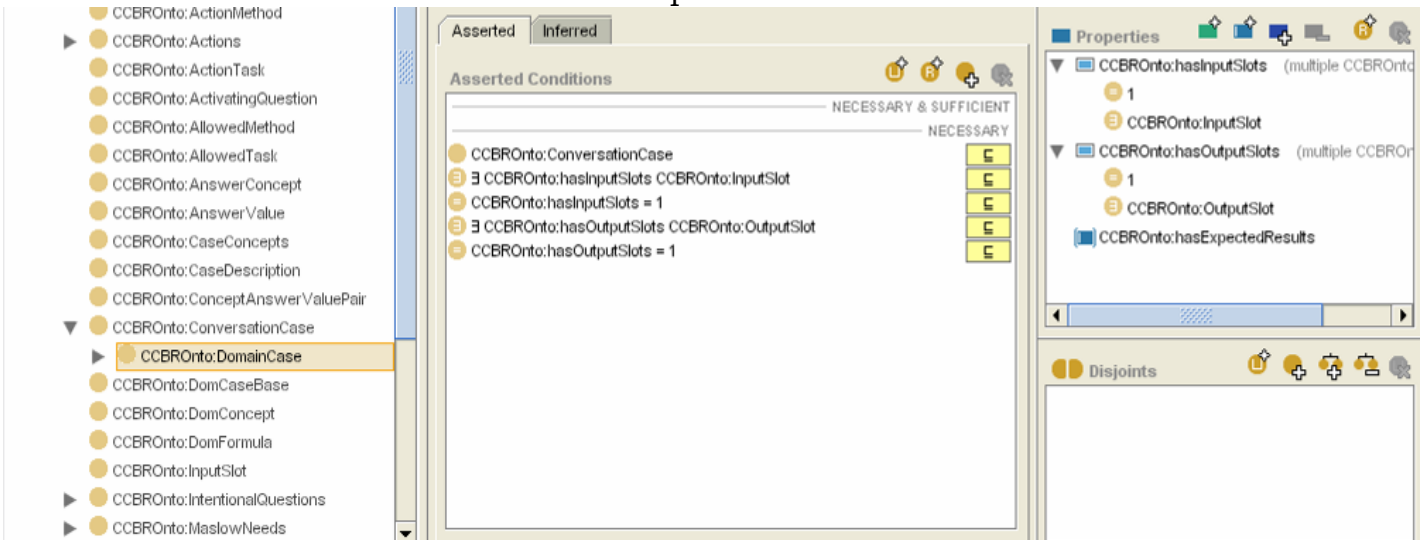


Las principales clases del CCBROnto que se utilizan son:

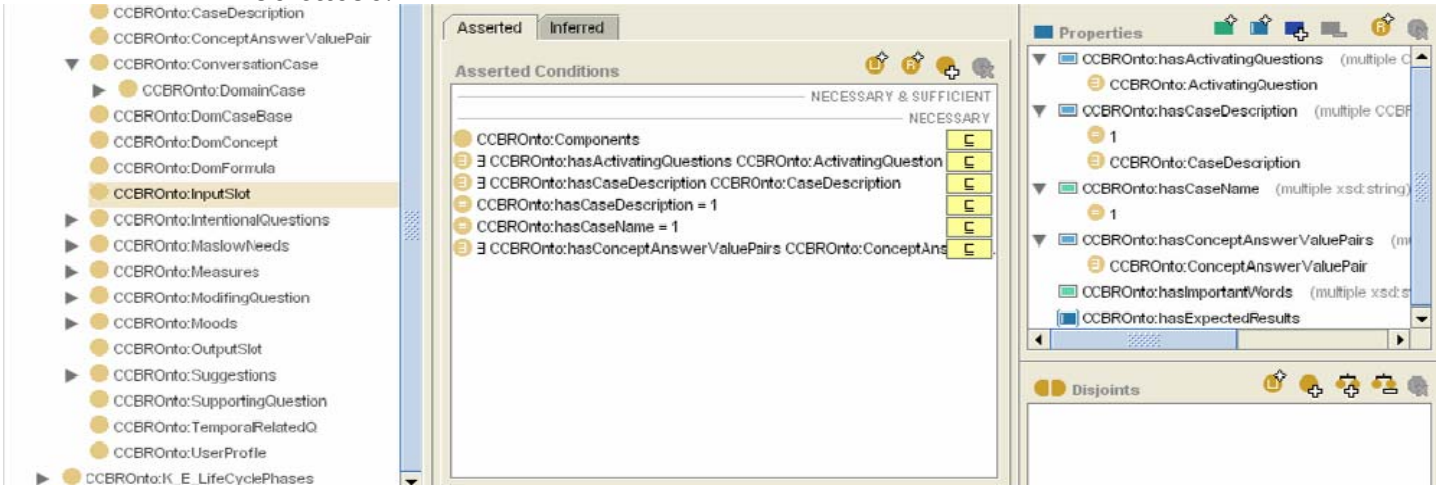
- DomCaseBase → Donde se incluyen los individuos correspondientes con los casos base de la aplicación:



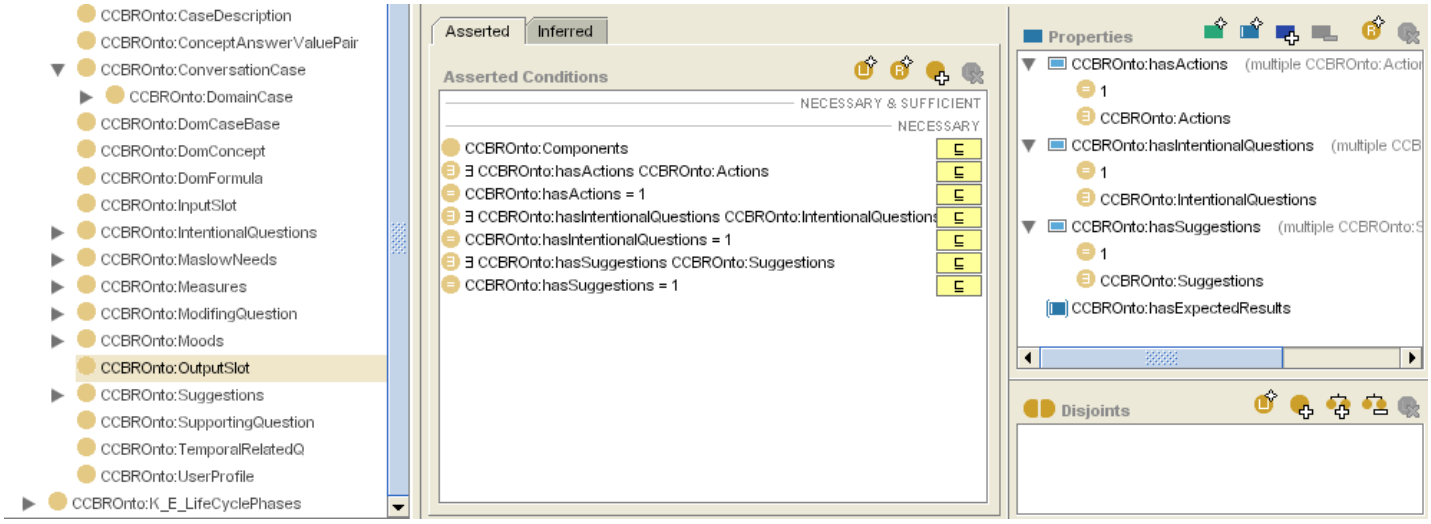
- DomainCase → Incluye todos los individuos correspondientes con el resto de los casos de la aplicación.



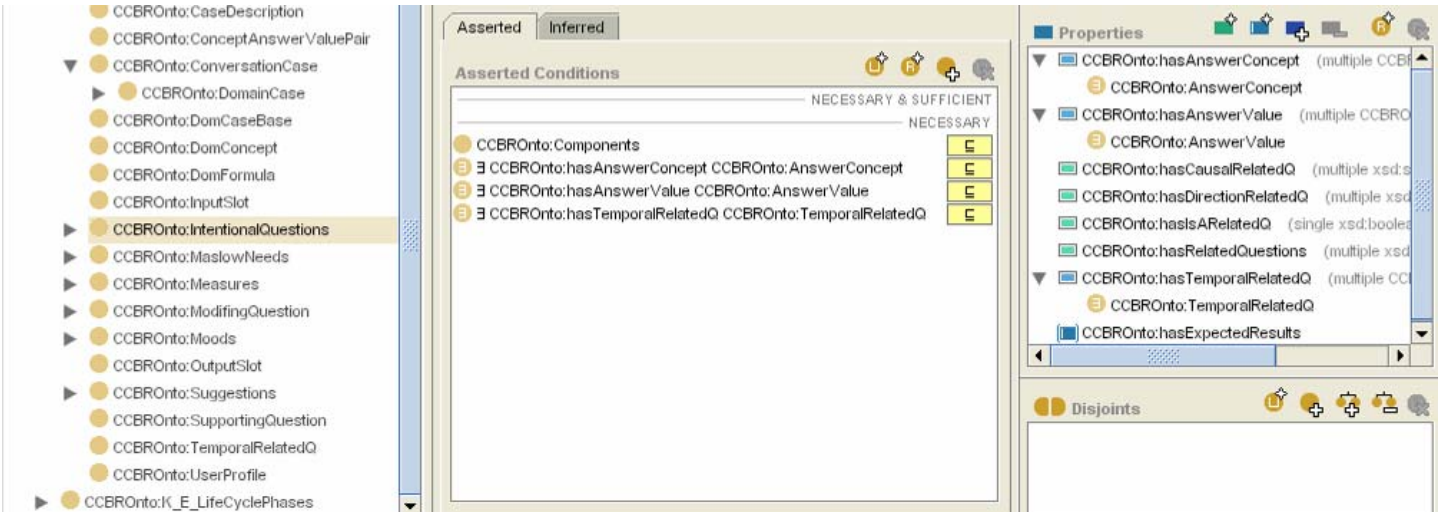
- InputSlot → Se almacenan los datos de entrada de cada uno de los casos.



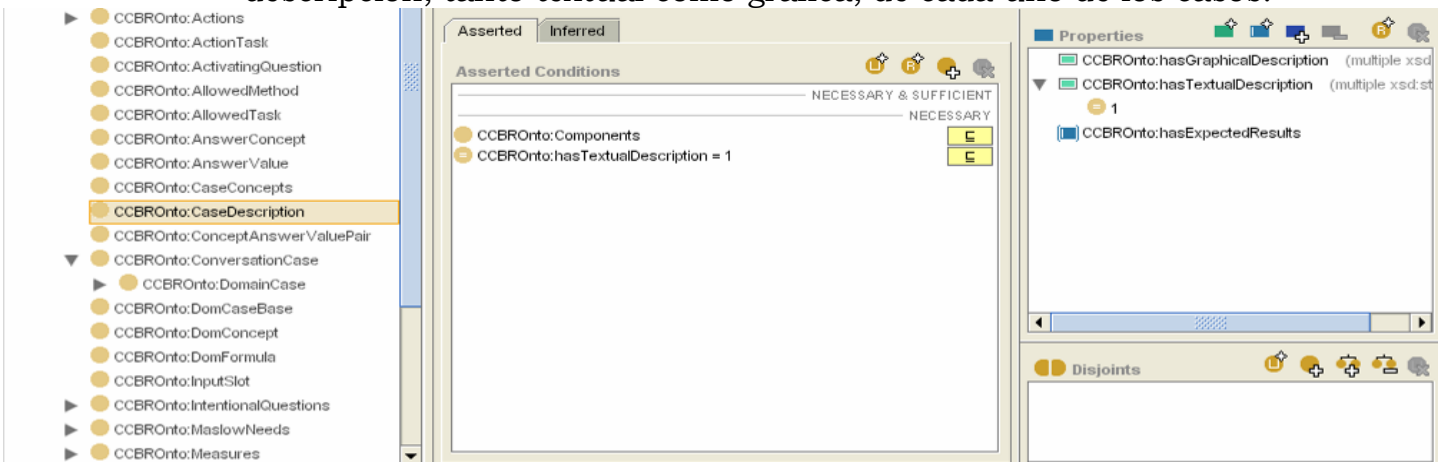
- OutputSlot → Almacena los datos de salida de cada uno de los casos.



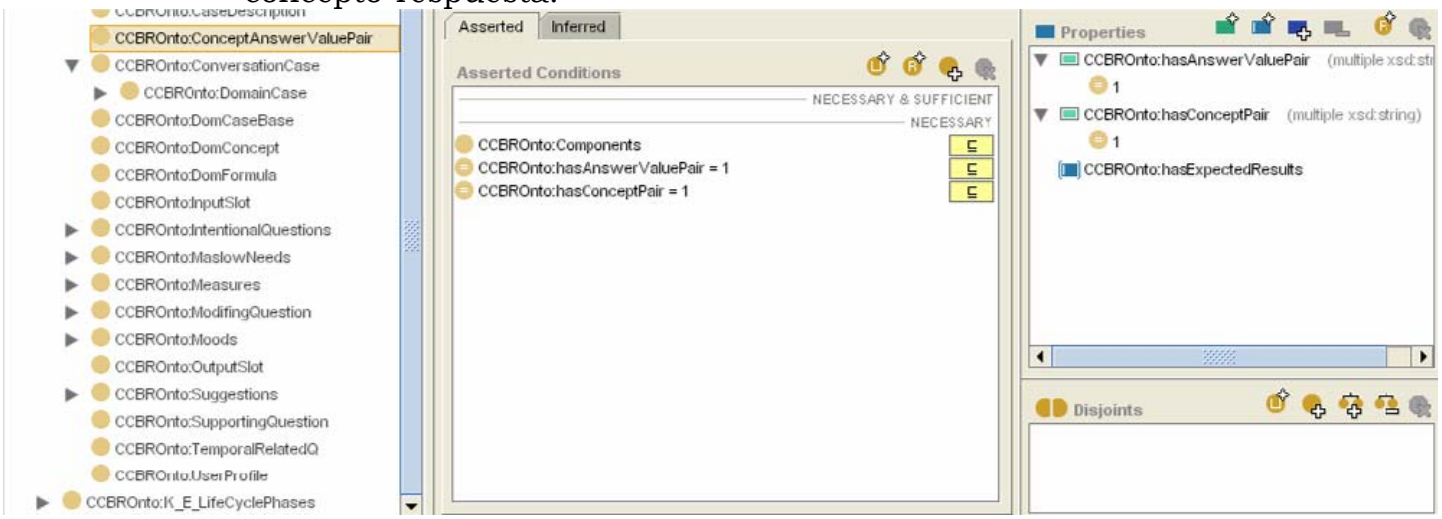
- IntentionalQuestions → se almacenan las preguntas generadas para generar acciones.



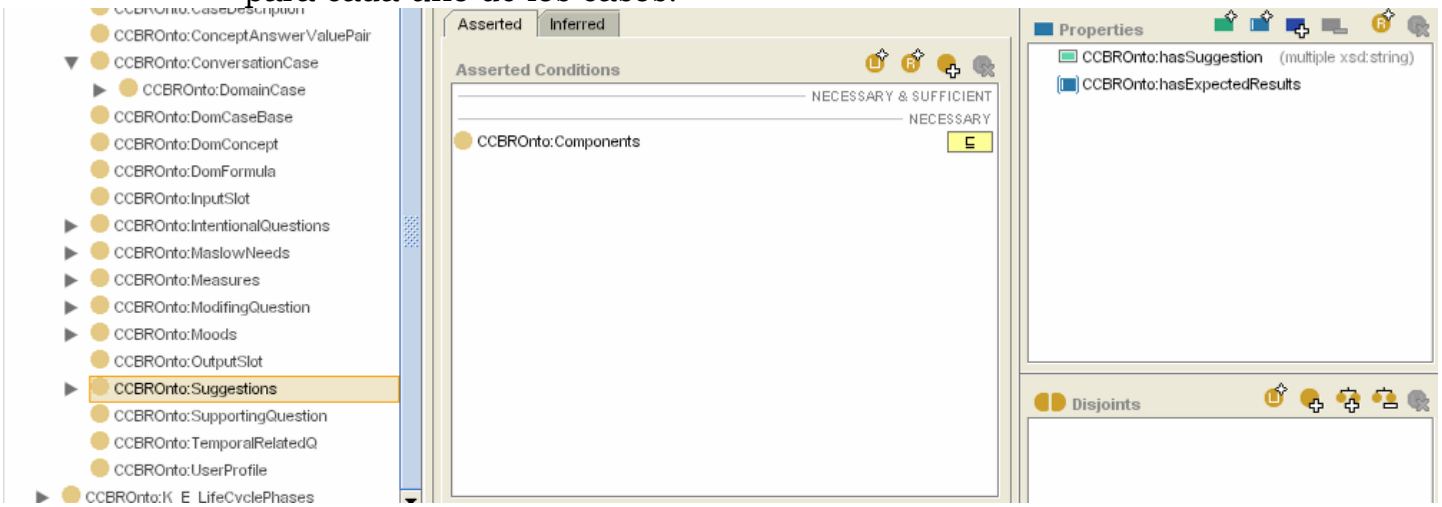
- CaseDescription → Para guardar la información referente a la descripción, tanto textual como gráfica, de cada uno de los casos.



- ConceptAnswerValuePais → Amacena los pares de valores concepto-respuesta.



- Suggestions → Amacena las sugerencias a mostrar al usuario para cada uno de los casos.



DOROnto

DOROnto es la ontología correspondiente a la tarea del dominio de la aplicación. En ella se incluye toda la información a procesar y a mostrar al usuario.

Ejemplo de Caso en DOROnto:

InputSlot

hasCaseName: "Gestión de Empresas con Visual Factu."
hasActivatingQuestions:
¿Cuántas empresas puede gestionar visual Factu?
¿Qué ficheros lleva vinculados una empresa?
¿Se pueden asociar varias empresas?
…
hasCaseDescription:
"Visual Factu permite gestionar tantas empresas como se desee. No se debe asociar necesariamente empresas con gestiones de Sociedades diferentes, ya que…"
hasImportantWords:
"vis, fact, permit, gestion, empres, dese, asoc, necesar,…"
hasConceptAnsweValuePais: --

OutputSlot

hasIntentionalQuestions:
Consultar borrar empresa
Consultar crear empresa
Consultar seleccionar empresa
…
hasSuggestions:
¿Qué desea hacer ahora?
hasActions: --

DYROnto

DYROnto es la ontología correspondiente a la tarea de la dinámica de la conversación. Esta ontología incluye toda la información necesaria para conocer los posibles estados de ánimo del usuario y posee las transiciones posibles dependiendo de los cambios de ánimo producidos en el usuario.

Ejemplo de Caso de diagnóstico en DYROnto:

hasWorsenDegree

Individual_Worsen_High_Degree

hasAction

GenerateToDo_Change_Fast

Ejemplo de Caso de predicción en DYROnto:

hasPreviousStep

GenerateToDo_Change_Fast

hasMoodState

Individual_Glad_State

hasNumberWorseningCycles

Individual_Low_Cycle

hasUrgencyDegree

Individual_Extreme_Degree
Individual_High_Degree_01
Individual_High_Degree_02
Individual_Average_Degree
Individual_Low_Degree_01
Individual_Low_Degree_02
Individual_Null_Degree

hasChangeTo

CASE_CB3_12

Ejemplo de Caso de diagnóstico en DYROnto:

hasSuggestion

DYRSuggestion_02
DYRSuggestion_12

hasIntentionalQuestion

-

USROnto

USROnto es la ontología correspondiente a la tarea del dominio del usuario. En ella se incluye toda la información que se posee del usuario.

Ejemplo de caso de uso de USROnto.Crear un usuario:

hasPersonalData

hasDNI:
Valor del recogido por pantalla para dni.
hasName.
Valor del recogido por pantalla para el nombre.
hasSurname:
Valor recogido por pantalla para apellidos.

hasSesionData

hasIdSesion:
Creación de un nuevo identificador para la sesión.
hasCaseName:
Vacío.

hasExpertiseLcvel

isExpertiseLevel:
Inicialmente posee un valor nulo.

COMPONENTES Y CLASES PRINCIPALES

La implementación de la aplicación está estructurada en varios paquetes:

- ConectorJavaOwl
  Paquete que incorpora las clases correspondientes para conectar Java con la ontología, a través de Pellet en conjunto con Jena.
  - AccesoConceptos → A través de esta clase se accede a los conceptos de la ontología
  - AccesoIndividuos → Para realizar consultas referentes a los individuos de la ontología.
  - AccesoPropiedades → Para realizar consultas referentes a las propiedades.
  - ConectorJavaOwl → Es la clase "principal" del paquete. Es la que se encarga de traducir la ontología y de crear los objetos para el acceso a sus conceptos, individuos y propiedades.
  - MainPruebasConector → Con esta clase se pueden realizar pruebas de la conexión con la ontología.

- COM
  Paquete correspondiente a las funcionalidades de la tarea COM.
  - VistaCOM → Es la vista correspondiente con la ventana COM de la aplicación.

- DOR
  Paquete correspondiente a las funcionalidades de la ventana DOR.
  - VistaDOR → Es la vista correspondiente con la ventana DOR de la aplicación.
  - EntradaCaso → Es el objeto correspondiente a la entrada a un caso de la ontología para ser consultado por el dominio.
  - RazonadorDOR → Es el encargado de razonar y mostrar los casos oportunos en la ventana DOR.
  - PalabraImportante → Objeto para almacenar una palabra importante de la ontología, junto con la tabla de casos en los que aparece la palabra.

- USR
  Paquete correspondiente a las funcionalidades de la ventana USR.
  - VistaUSR → Es la vista correspondiente con la ventana USR de la aplicación.
  - ControlErroresUSR → Es el encargado de manejar la excepciones producidas por el acceso a la ontología si las hubiese, e intentar solucionar dicho error.

  - RazonadorUSR → Es el encargado de razonar y mostrar los datos oportunos en la ventana USR según el estado de la ejecución
  - PersonalData → Es la clase mediante la cual es posible manipular los datos personales de los usuarios de la ontología
  - SesionData → Es la clase mediante la cual es posible manipular los datos de las sesiones de la ontología
  - User → Es la clase mediante la cual es posible manipular los datos de los usuarios de la ontología: tanto las sesiones como los datos personales o los niveles de experto.

- DYR

  Paquete correspondiente a las funcionalidades de la ventana DYR.
  - VistaDYR → Es la vista correspondiente con la ventana DYR de la aplicación.
  - Clock → Es el objeto responsable de controlar la ejecución automática del razonador DYR, así como de controlar el tiempo de ejecución de dicho razonador.
  - ControlErrores → Es el encargado de manejar la excepciones producidas por el acceso a la ontología si las hubiese, e intentar solucionar dicho error.
  - MotorDYR → Es el objeto encargado de ejecutar un ciclo del razonador DYR. Una vez cargados todos los argumentos necesario para su ejecución, realiza las tres tareas del razonador DYR (Diagnostico, Predicción y Planificación).
  - RazonadorDYR → Se encarga de la iteración con el usuario, a la hora de ver que argumentos necesita el motor del razonador, buscárselos y mandárselos.

- GUI

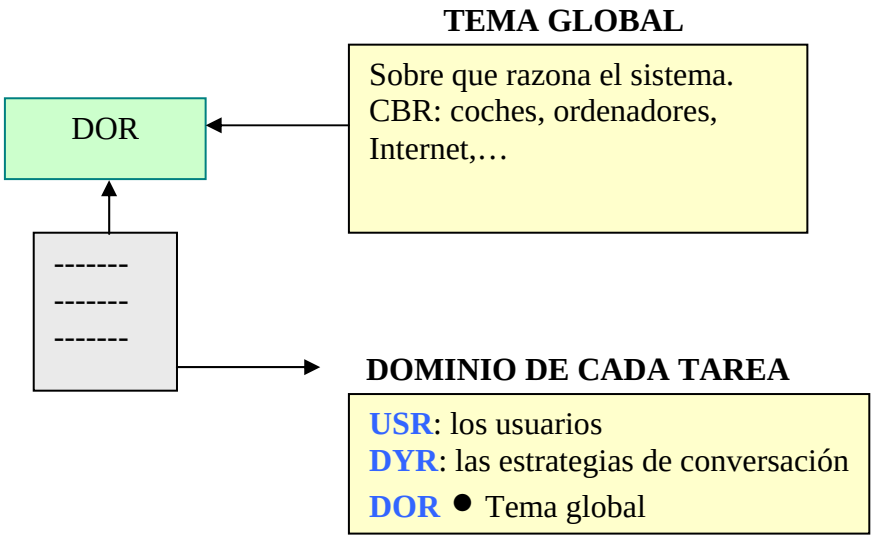
  Paquete para los componentes de la Interfaz gráfica.
  - Modelo → Es el modelo de la aplicación.
  - VentanaEspera → Ventana para indicar cuando la aplicación está procesando algún dato.
  - VentanaMensaje → Ventan estándar para los mensajes de la aplicación.
  - VentanaOpciones → Ventana para la modificación de las opciones propuestas al usuario.
  - VentanaPresentación → Ventana de presentación al aplicativo.
  - Vista → Vista principal de la aplicación, su funcionamiento lo modela la clase Modelo. En ella se encuentran incluidas todas las demás vistas a modo de ventanas internas.

- SistemaConversacional
  Paquete de acceso a la aplicación
  - ProgPrincipal → Clase con el main (punto de acceso) de la aplicación

- Utils
  Compuesto por las clases de las utilidades.
  - Executer → Ejecutor de la aplicación
  - LOG → Clase para almacenar los LOGs de la aplicación.
  - Pizarra → Pizarra de la aplicación
  - Procesador → Encargada del procesamiento de los archivos seleccionados por el usuario y de los textos introducidos por el mismo.
  - Regla → Clase para las reglas utilizadas en la extracción de la raíz de una palabra.
  - Stemmer → Extractor de raíz de una palabra.
  - UtilidadesTextos → Utilidades para el tratamiento de textos.

TAREAS DETALLADAS

Las tres principales subtareas de las que está compuesta la aplicación: Dominio (DOR), Usuario (USR), Dinámica de la Conversación (DYR); interactúan entre si mediante una pizarra, un ejecutor y una agenda.



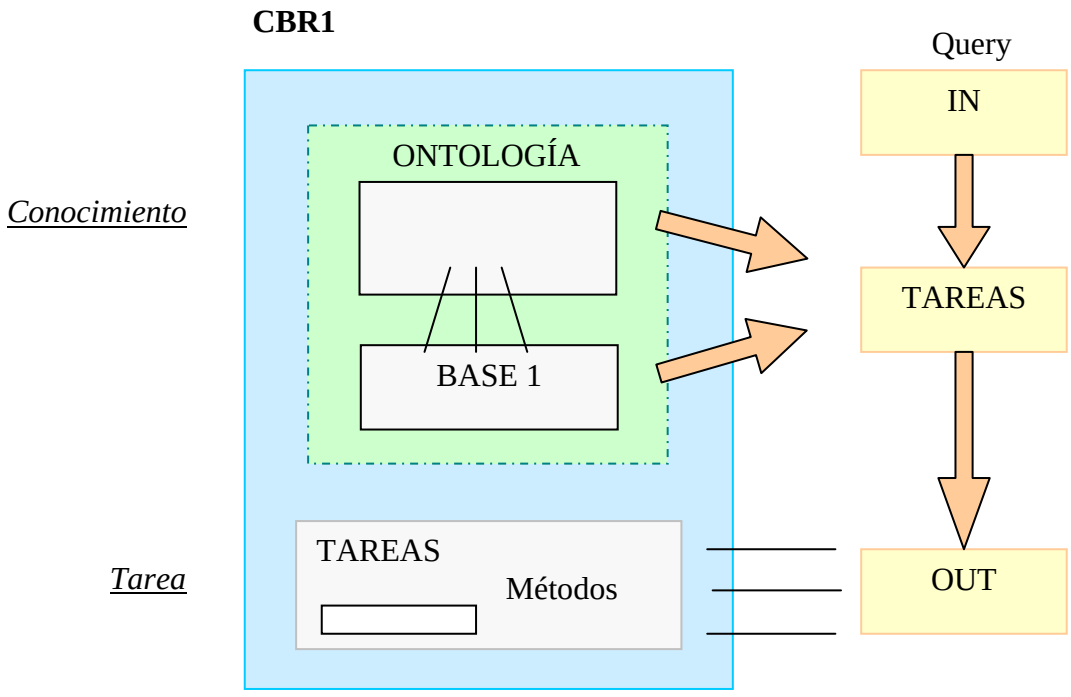
En la agenda, se van almacenando las tareas a ejecutar con un orden preestablecido. Cuando el usuario decide procesar una consulta en la ventana DOR, por ejemplo, se almacena al principio de la agenda la tarea "DOR". De manera que cuando se llama al ejecutor, para ejecutar las tareas, va cogiendo desde el principio hasta el final, todas y cada una de las tareas, y las va ejecutando en orden. Si al ejecutar una tarea, como es el caso de la tarea correspondiente a la ventana COM, se

necesita ejecutar inmediatamente después otra de las tareas, esta última se incluye en la agenda en la posición oportuna.

En la extracción de cada tarea para su ejecución, se procede de igual forma, a la eliminación de la misma en la agenda.

Como el término pizarra indica: *"Trozo rectangular y pulido de roca metamórfica y homogénea formada por la compactación de arcillas, que se utiliza para escribir anotaciones, realizar explicaciones,... de manera que permite el aprendizaje a más de una persona.;* en este proyecto la pizarra sirve para que cada una de las tareas puedan ver lo que las demás anotan. Es decir, cuando una de las tareas de la aplicación consigue un dato del usuario que pueda ser útil a alguna de las otras tareas, este dato es almacenado en la pizarra. Se utiliza para almacenar cualquier información; referente al ciclo en el que se encuentra la aplicación, para poder ser consultada en cualquier momento, por cualquiera de los razonadores de las tareas. Se debe ver como una pizarra "normal", en la que cada elemento tiene su sitio establecido.

En cada ciclo de la ejecución, cuando el usuario introduce una consulta, está es procesada por las tareas de la aplicación; que interactúan con los datos almacenados en las ontologías, y obtienen una salida como resultado de la consulta.



Si el usuario no es capaz de ofrecer una buena descripción del problema, el razonador no será capaz de encontrar un caso apropiado. Por ello, el sistema va guiando al usuario para entender mejor el problema. Si podemos formalizar cada tarea separadamente y buscar

una solución para cada uno, entonces podemos buscar soluciones parciales incrementadamente hasta que cubramos el tema principal.

El lenguaje a usar es formal y soporta la implementación al mismo tiempo. El sistema hace preguntas para completar la consulta del usuario y obtener el dato para resolverla. Puede decidir cuales son las preguntas relevantes que se deben hacer en el próximo ciclo, y puede ejecutar acciones fuera del sistema CCBR mostrando imágenes o ejecutando funciones,…

Teniendo en cuenta el dominio o tema, se puede distinguir entre sistemas de diagnóstico, soporte de decisión, recomendaciones y sistemas que justifican sus recomendaciones,…la aplicación está diseñada para que funcione indiferentemente del tema a tratar. Para ello, sólo habría que modificar la ontología del dominio con el nuevo tema.

CICLO GLOBAL

Al arrancar la aplicación, lo primero que se visualiza es la ventana de presentación. Una vez que pulsemos en el botón "Sistema Conversacional Inteligente", se crea la Vista principal del proyecto. Esta a su vez pide al modelo de la aplicación que cree las ventanas internas.

Además el modelo se encargará de crear todos los razonadores que serán necesarios en la ejecución del programa.

Cada uno de los razonadores correspondientes con las tareas de la aplicación, se encargan de crear un objeto conector Java-OWL para poder conectarse a su ontología. Haciendo uso posteriormente de este conector para traducirla.

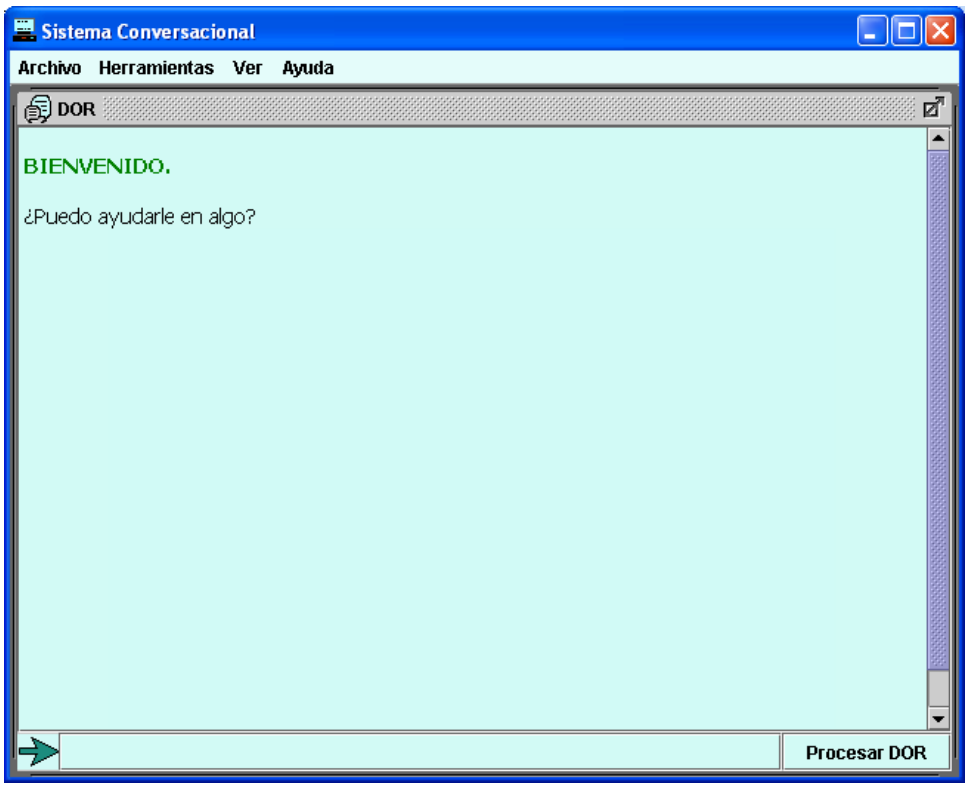
Ciclo del razonador DOR:

El razonador DOR al iniciar la ejecución, consulta todos los individuos existentes en la ontología DOROnto de "InputSlot" y extrae de cada uno de ellos: su nombre, su lista de palabras importantes, sus preguntas de activación, y su objeto individuo como tal. Esto es debido a que cada vez que el usuario realice una determinada consulta, el sistema debe buscar en las entradas de los casos de la ontología del dominio, aquellos que más se aproximen a la consulta del usuario. Para poder disfrutar de una mayor velocidad de respuesta, por parte de la aplicación, se almacenan estos datos la primera vez en la consulta del caso base. El resto de las veces, será más rápido su estudio.

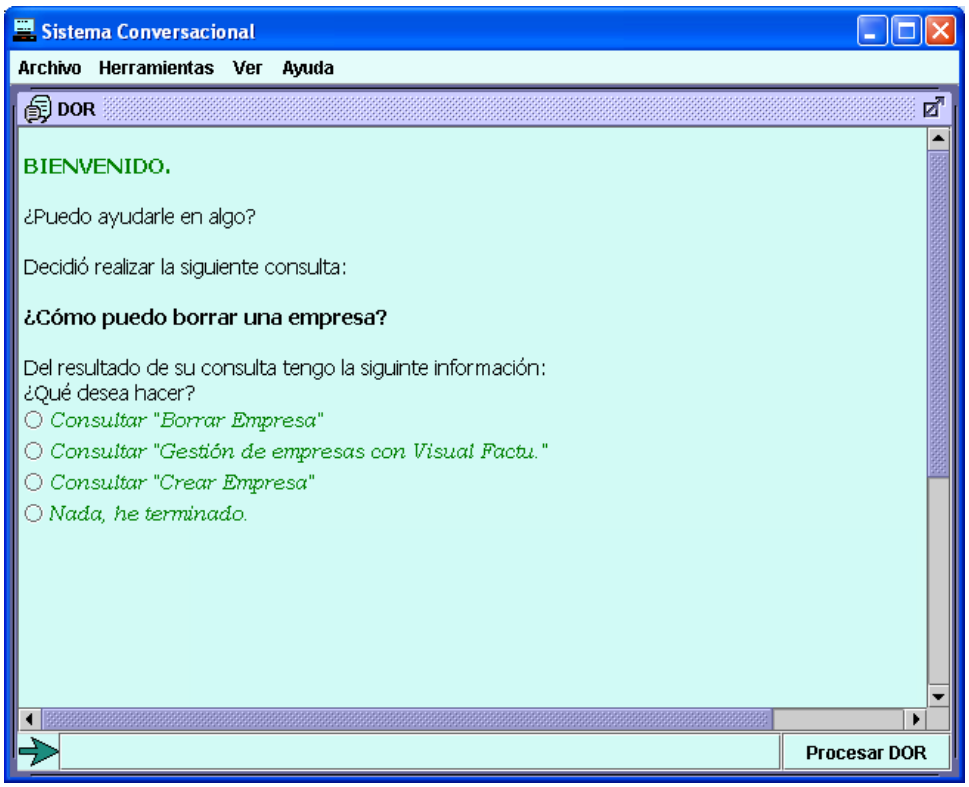
Una vez que se identifique el caso a mostrar al usuario, se consultarán en la ontología los valores de la descripción del caso, los

pares conceptos-respuestas, y las salidas del caso (OutputSlot). Hasta ese momento, estos valores no son necesarios para la aplicación, y por tanto, no se almacenan.

Para la recuperación del caso base, cuando se crea el ejecutor del sistema, se agrega a la agenda la tarea DOR. De esta manera, al crear la vista principal y llamar al ejecutor para que ejecute las tareas disponibles, éste ejecutará DOR que a su vez buscará en la ontología los casos base disponibles, y aleatoriamente elegirá uno de ellos. Mostrando el saludo inicial del sistema.



El usuario escribirá en el recuadro de texto su consulta y pulsará sobre el botón “Procesar DOR” o en su defecto, pulsará la tecla ENTER.

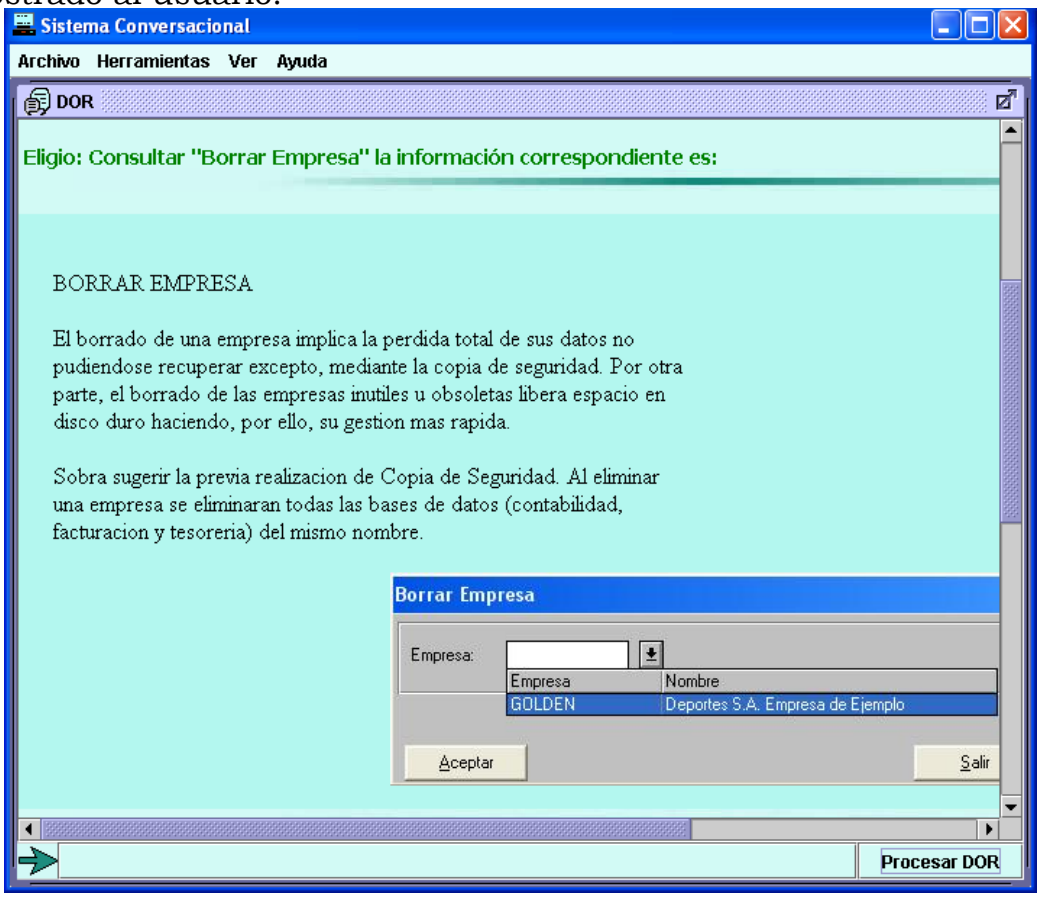


Una vez hecho esto, el sistema recogerá el texto introducido por el usuario y realizará un tratamiento del mismo. De esta forma, se le quitarán las tildes, se eliminarán los espacios no deseados, los signos de puntuación y se pondrá todo en minúsculas, para realizar una búsqueda estándar en la información de la ontología. Al mismo tiempo, se procederá a la extracción de las palabras importantes del texto introducido.

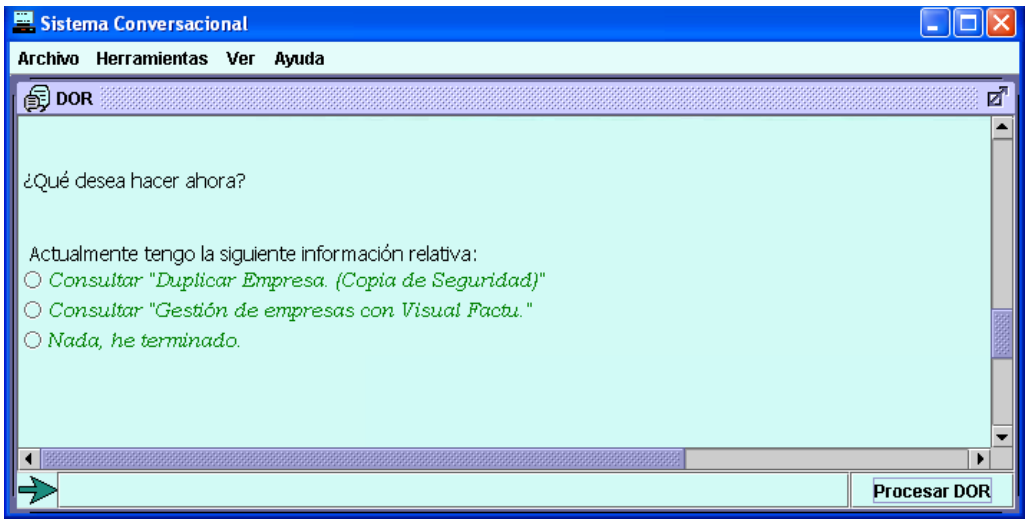
Llamamos palabras importantes a las obtenidas a partir de un texto, una vez eliminadas las cadenas vacías y las palabras superfluas cuyo significado no aporta nada. Además únicamente nos quedamos con la raíz de estas palabras, evitando así los cambios semánticos debidos al género y número.

Cuando el sistema tiene el texto con el formato adecuado, el razonador DOR consulta si alguno de los casos tiene una pregunta de activación como la realizada por el usuario. Si es así se le mostrará al usuario. Sino busca los casos que tengan relación con la consulta realizada a través de las palabras importantes, los ordena según el número de coincidencias y le muestra al usuario las opciones oportunas.

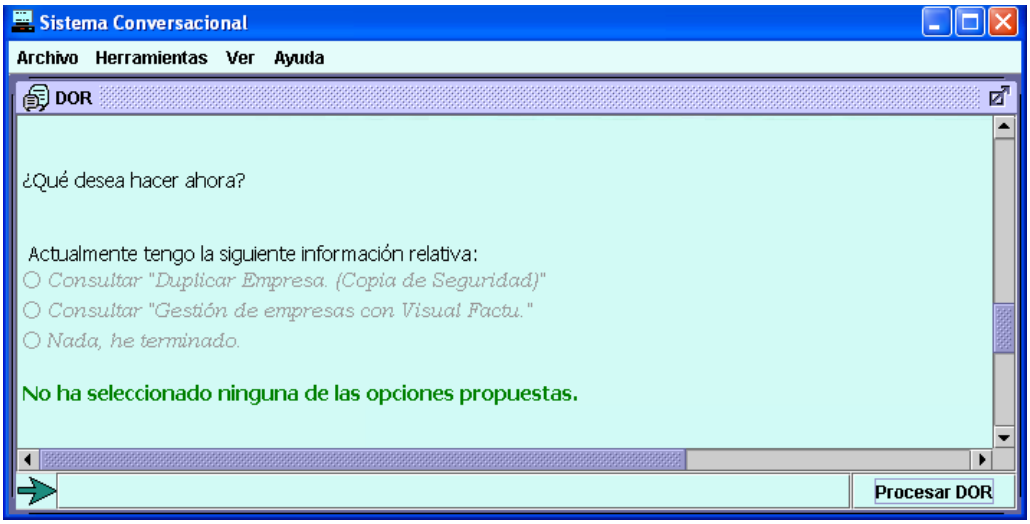
El usuario, podrá elegir una de las opciones indicadas, y si el caso asociado con su elección tiene toda la información necesaria, este será mostrado al usuario.



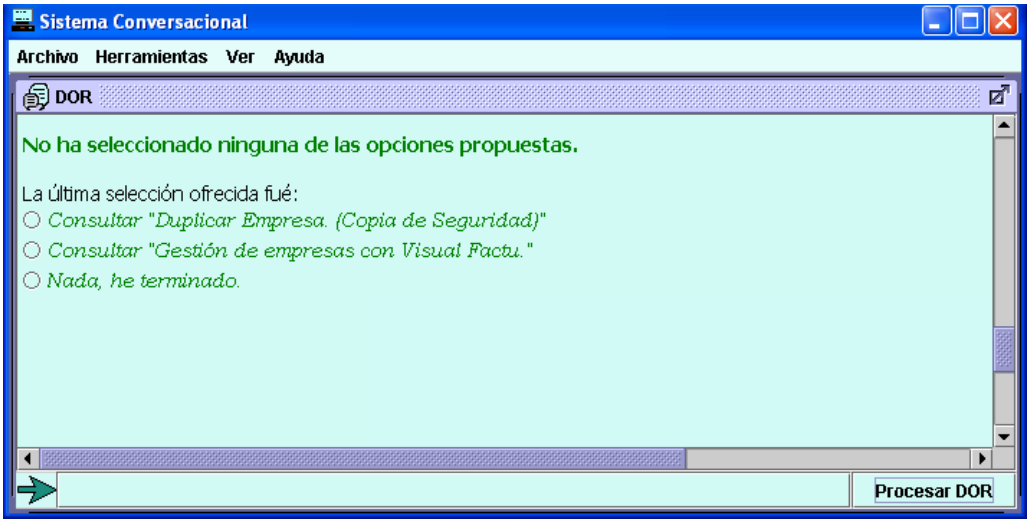
Además la aplicación le mostrará los casos relacionados con el seleccionado, dándole la opción al usuario de poder elegir uno de estos casos.



Si el usuario no quiere seguir consultando ninguno de los casos propuestos y pulsa el botón “Procesar DOR”, la aplicación le indicará que no ha seleccionado ninguno.

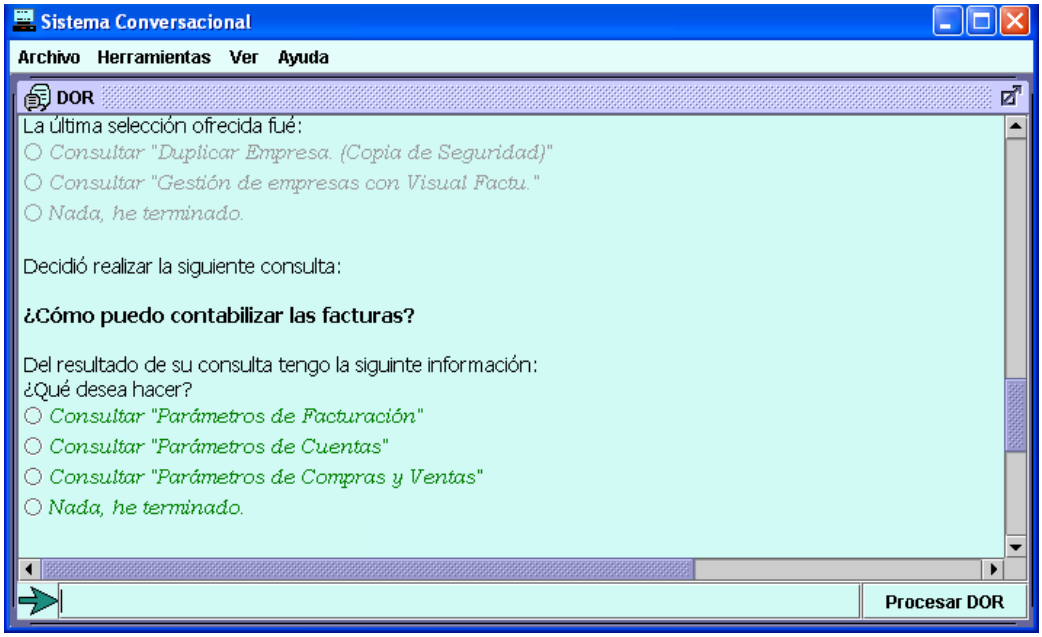


En este punto, si el usuario desease volver a poder seleccionar uno de los puntos anteriores, podría irse al menú Herramientas y seleccionar “Volver a la última selección”, de manera que la aplicación le volverá a mostrar los puntos anteriores permitiéndole la selección de alguno de ellos.

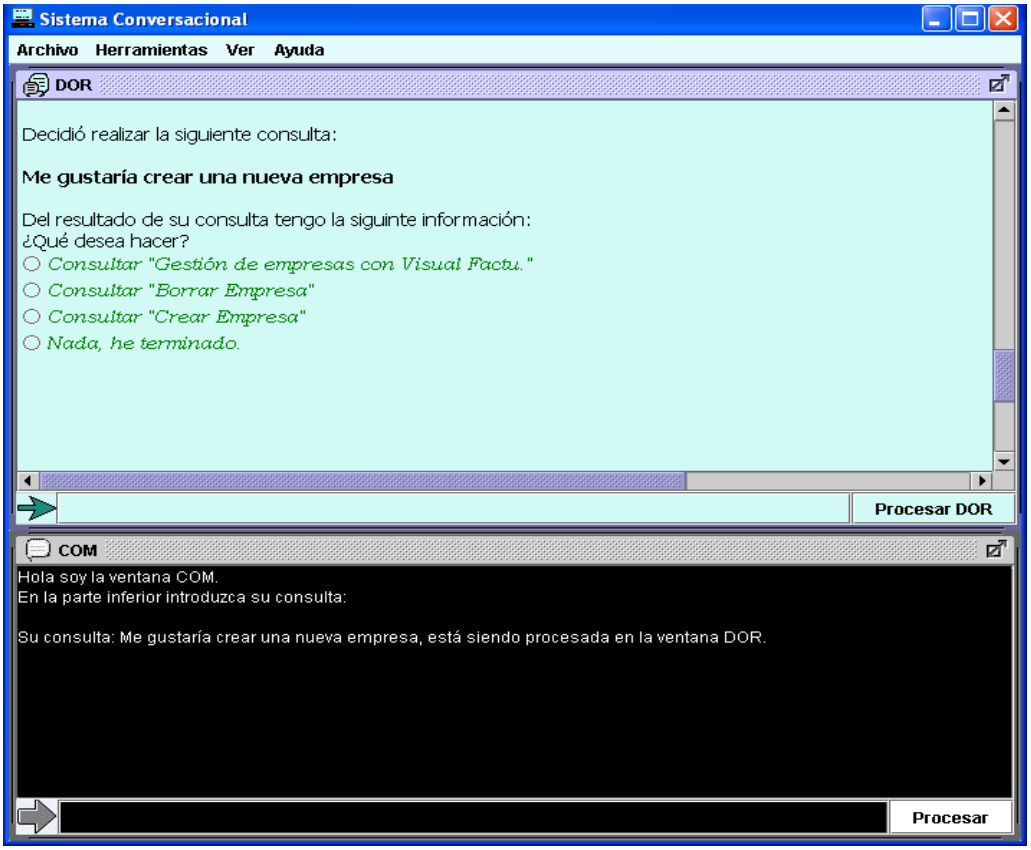


Como se puede observar, además de los casos relacionados, el sistema siempre le muestra la posibilidad de terminar su consulta ("Nada, he terminado").

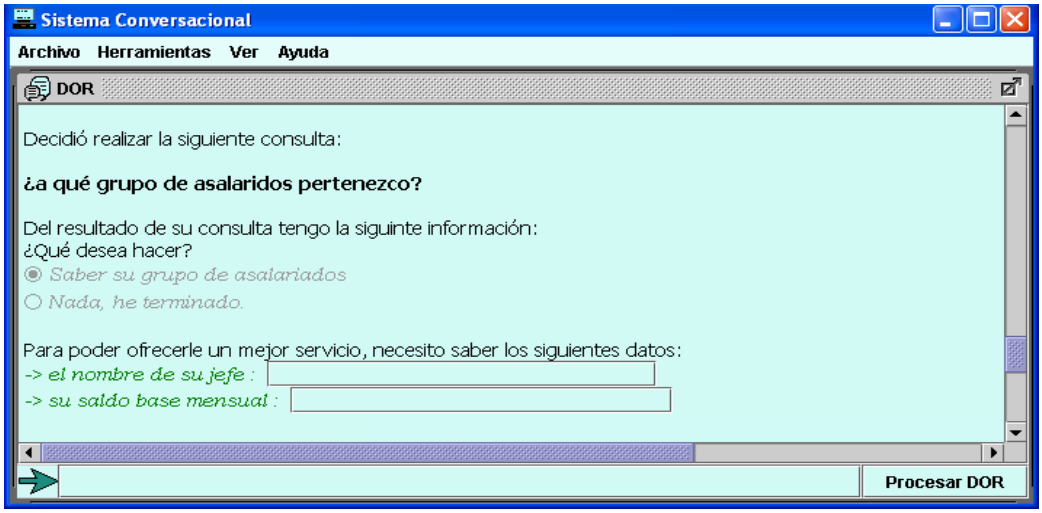
También existe la posibilidad de realizar una consulta diferente a la que se está mostrando por pantalla. Para ello, simplemente se debe escribir la nueva consulta en el cuadro de texto y pulsar procesar (o en su defecto, la tecla ENTER). El sistema pasará a procesar la nueva consulta introducida.



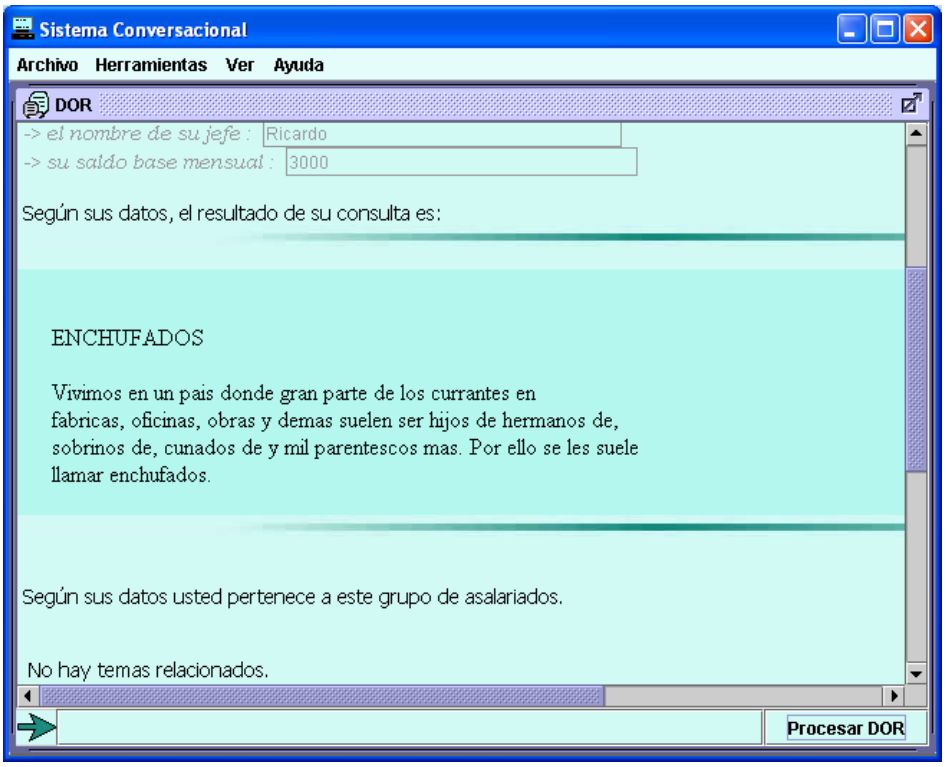
Si en algún momento, de la interacción con el sistema, el usuario no sabe en cual de las ventanas debe introducir su consulta siempre puede introducirla en la ventana COM. Esta enviará la consulta a la ventana correspondiente para ser procesada.



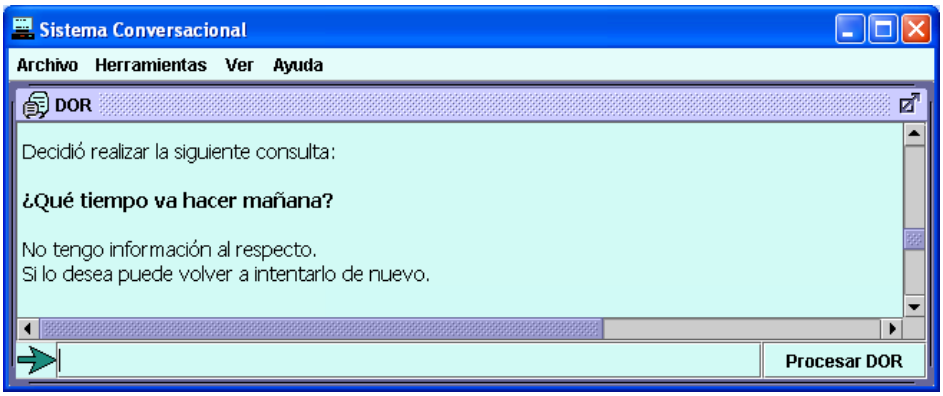
Pueden existir consultas que requieran participación por parte del usuario. Esto es debido a que algunos de los casos almacenados pueden tener el mismo identificador, diferenciándose en los valores de sus conceptos (parejas concepto-valor). Si este es el caso, el sistema le pedirá al usuario los datos que necesite saber (en caso de no tenerlos de alguna consulta o sesión anterior).



Una vez que el usuario rellene sus datos el sistema le ofrecerá la información almacenada correspondiente con su perfil.

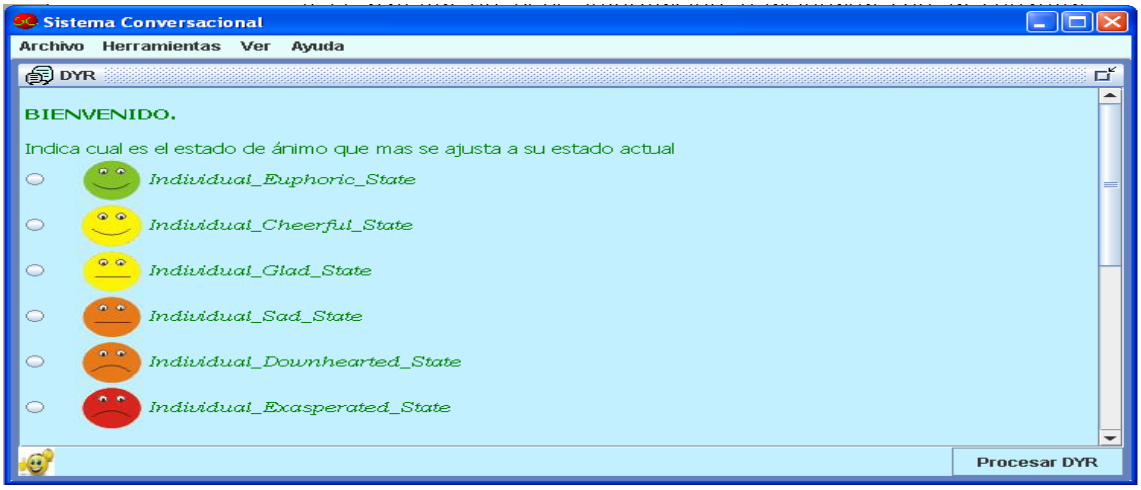


Si el sistema no tiene información relacionada con la consulta introducida, muestra un mensaje al usuario.



Ciclo del razonador DYR:

Al iniciar el razonador DYR, lo primero que hace es consulta los estados e ánimo conocidos por su ontología y muestra en su pantalla los distintos estados, representados sus caras, he invita al usuario a que le indique el estado de ánimo en el que se encuentra.



Una vez el usuario indica su estado de ánimo, eligiendo la imagen que más represente su estado, debe pulsar el botón "Procesar DYR" e inmediatamente después el razonador se activa y comienza la ejecución de un ciclo.

Un ciclo del Razonador DYR, posee tres subtareas:

- Subtarea Diagnosis: Método de *Decisión de Cambiar a otro Bucle*
  Conocimiento: Ontologías y Bases
  Razonamiento
- Subtarea Predicción: Método de *Selección del nuevo Bucle*
  Conocimiento: Ontologías y Bases
  Razonamiento
- Subtarea Planificación: Método que *Implementa el nuevo Bucle*
  Conocimiento: Ontologías y Bases
  Razonamiento

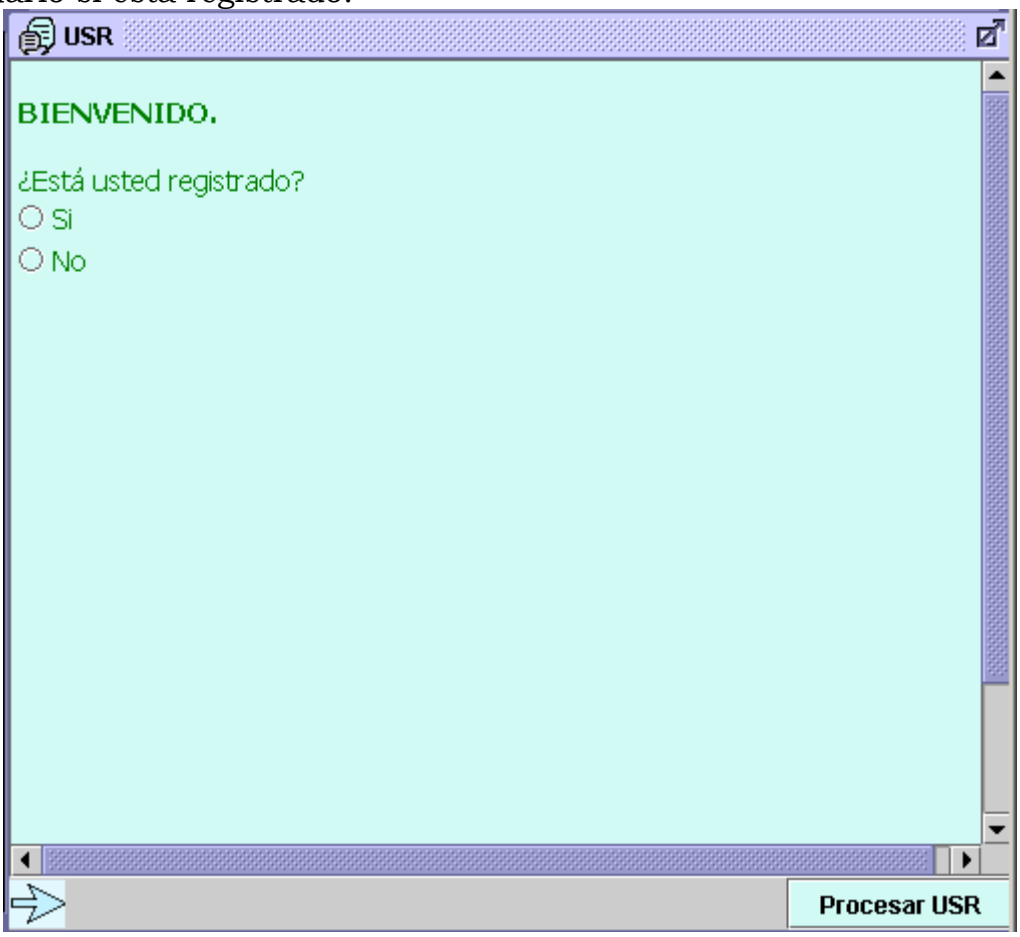
Una vez ejecutado el ciclo del razonador, muestra las sugerencias, estas suelen ser frase de ánimo para el usuario, para estimularle, con imágenes divertidas o agradables dependiendo de cómo se encuentre el usuario y de todo lo que haya pasado anteriormente.

Si el caso en el que se encuentra actualmente el usuario, tiene preguntas que hacerle al usuario, para evaluar ciertos aspecto, se le formularan la próxima vez que el usuario pulse "Procesar DYR", o para que el usuario, no se vea forzado a estar manejando este razonador, hay un proceso que controla la ejecución automática del razonador DYR, por lo que si el usuario no pulsa el botón de procesar durante una largo tiempo, la ejecución automática mostrará el siguiente paso de la ejecución del ciclo.

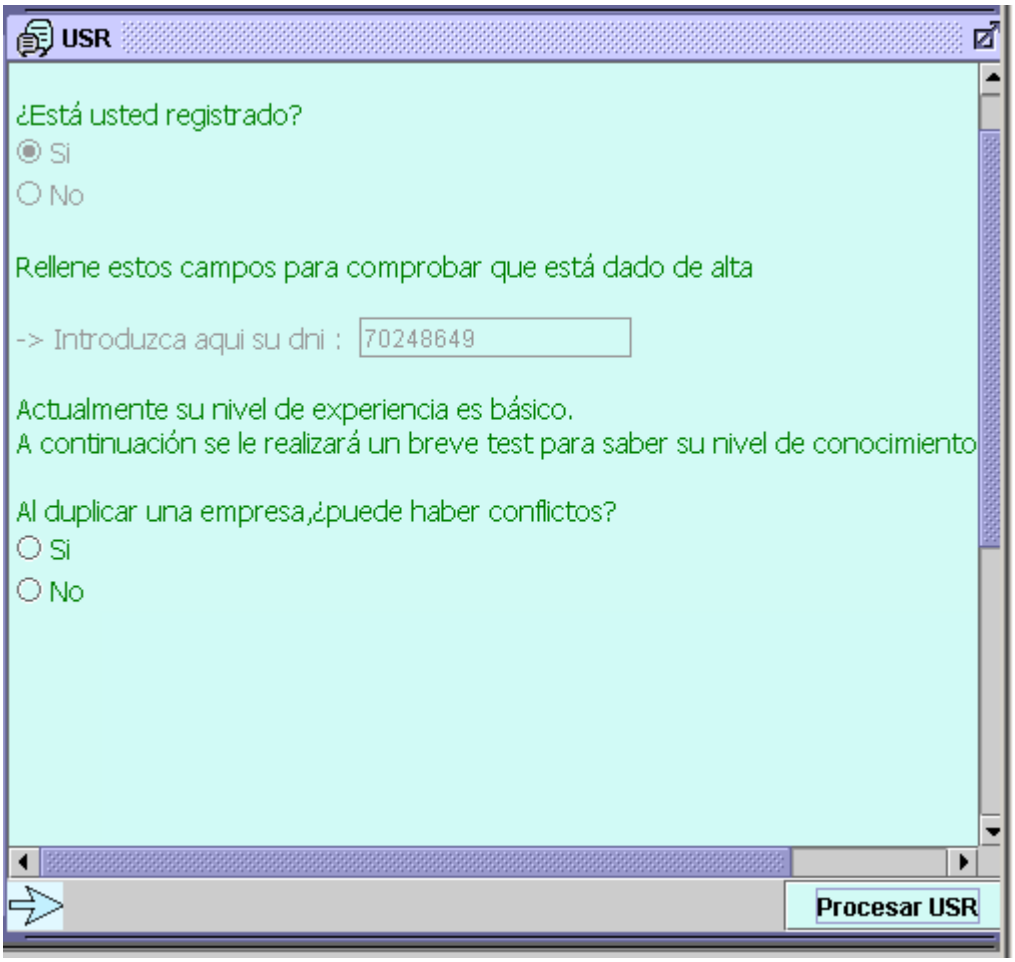
El usuario posee la libertad de contestar o no a estas preguntas, si lo hace se procesarán, si no se pasará directamente al siguiente paso de la ejecución del razonador, que es volver a mostrar los estados de ánimo. En este paso de la ejecución, es el único, donde la ejecución automática no actuará hasta que el usuario elija su estado de ánimo.

Ciclo del razonador USR:

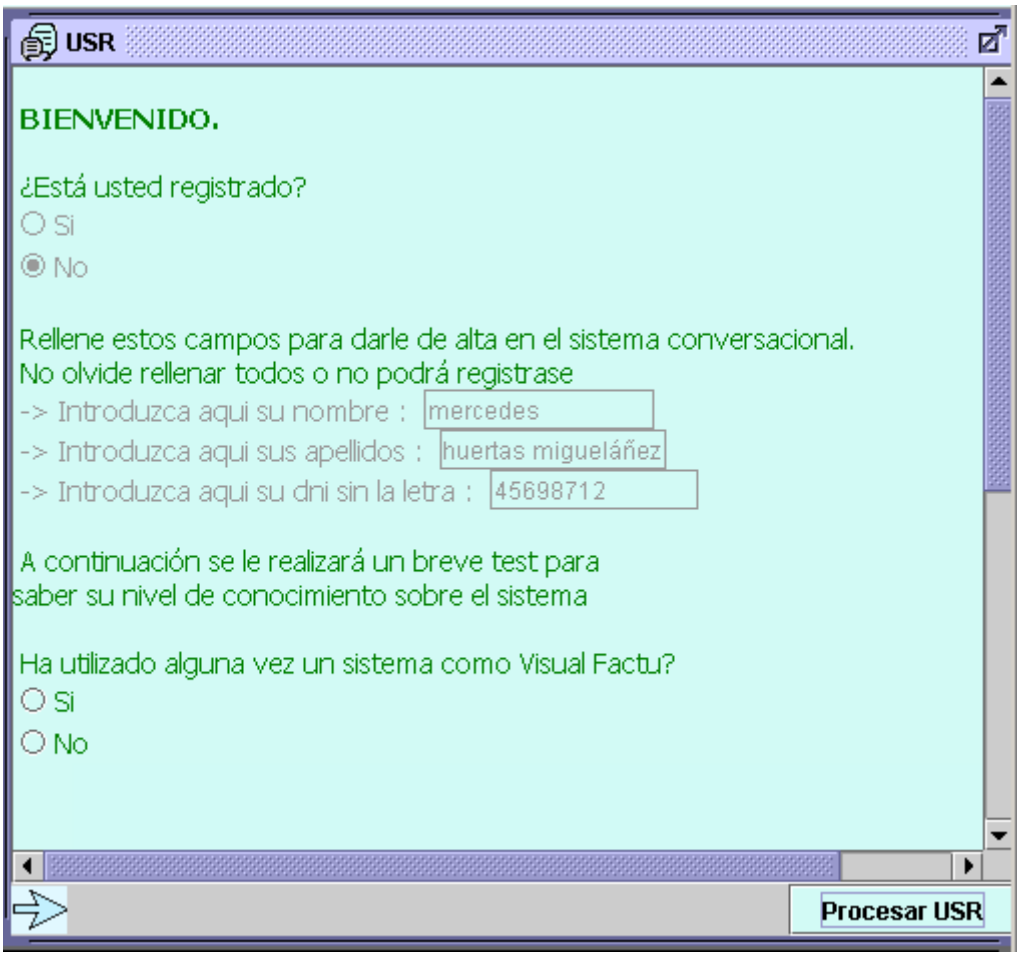
Al iniciar el razonador USR, lo primero que hace es preguntarle al uauario si está registrado.



Si lo está, le pide que introduzca el dni, para comprobar que, efectivamente, lo está. En caso de introducir un dni incorrecto, nos ofrece la posibilidad de intentarlo tres veces. Si en esas tres veces, no hemos conseguido entrar, el sistema se cerrará automáticamente. Pero, si por el contrario, el dni es correcto procederá a hacer un test para, a partir del nivel de conocimiento anterior, obtener si puede avanzar en su estado de conocimiento o no.

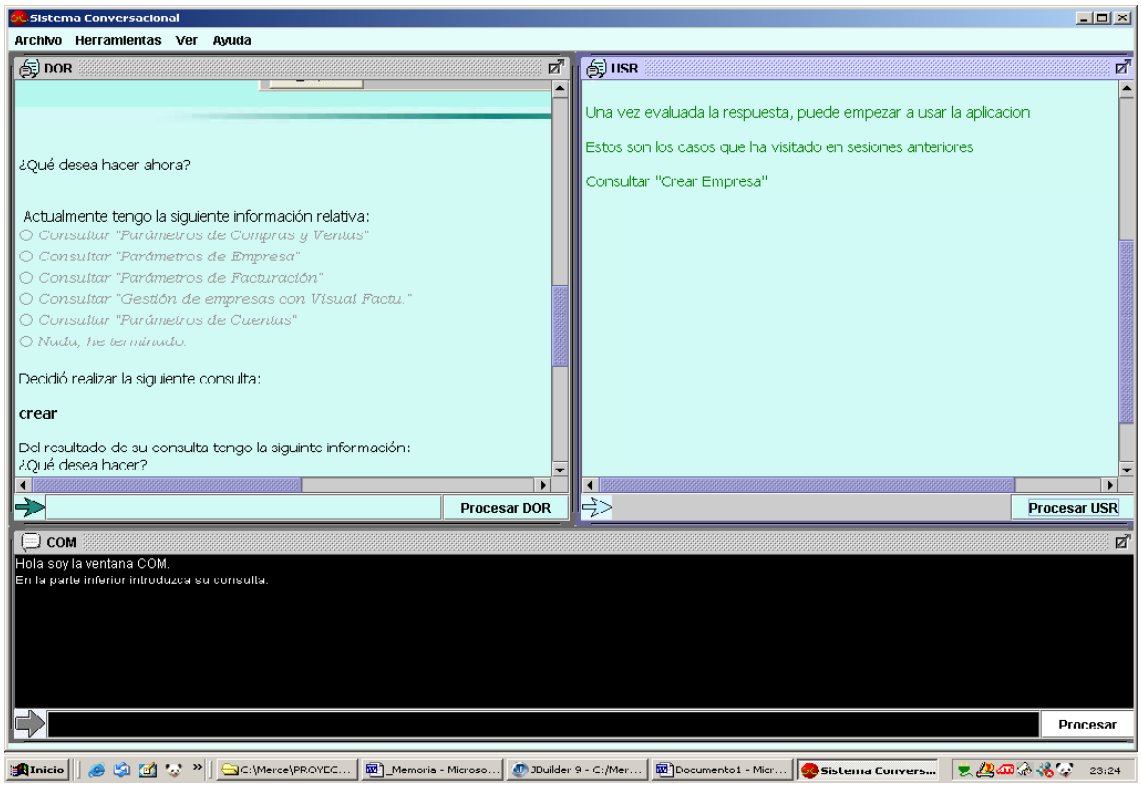


Si el usuario no está registrado, debe reellenar una serie de datos para darse de alta en la aplicación. La aplicación los almacenará comprobando previamente que es correcto y que no está repetido. También se le realiza un breve test para comprobar su nivel de concocimiento y asignarle uno.



Una vez se han realizado estos pasos, comienza la interacción con la ventana DOR.

Cuando el usuario solicite ayuda a DOR y éste le muestre las opciones que más se aproximan a su petición, USR buscará entre las sesiones que tiene asociadas el usuario, si alguno de los casos a mostrar ya ha sido visitados. Si alguno ya ha sido visitado, le mostrará al usuario un mensaje indicándole los casos por los que ha pasado.



Cuando el usuario elija uno de esos casos, se alamcenará en un array, para que al acabar la sesión se puedan guardar.

Si pulsamos el botón salir o elige la opción “Nada he terminado”, se guardan todos los datos de la sesión en la ontología.

4.3- ESTADO DE LA IMPLEMENTACIÓN:

PROBLEMAS DETECTADOS

Se han detectado algunos problemas a la hora de consultar a las ontologías. Algunas veces, y de manera aleatoria, que no se carga correctamente la ontología y cuando se consulta por ciertos individuos, clases o propiedades de ella, no devuelve nada o devuelve algo incorrecto. Esto se ha solucionado controlando los errores y reiniciando la consulta a la ontología. De esta manera, al realizar otra vez una nueva consulta se obtiene el objeto deseado.

Pero aún así, como algunas de las ontologías guardan datos, el sistema se vuelve inconsistente y no recupera bien los datos almacenados en ella.

También se ha observado, como algunas veces, al llevar un cierto tiempo de ejecución de la aplicación, se produce un error de memoria. Se ha solucionado, en parte, liberando el conector cada vez que no se necesita, para que otro razonador cargue su ontología.

POSIBLES MEJORAS FUTURAS

Una de las posibles mejoras, es realizar una investigación más profunda sobre el conector Java-OWL, ya que se ha visto que existe la posibilidad de hacer un conector mucho más potente que el actual, incluyendo alguna parte para realizar consultas RDQL sobre Pellet.

5. DOCUMENTACIÓN

5.1- MANUAL DEL USUARIO:

SISTEMA CONVERSACIONAL INTELIGENTE

MANUAL DE AYUDA

INTRODUCCIÓN

Este manual le ayudará en el manejo de la aplicación.

Es una aplicación diseñada para que funcione indiferentemente del tema a tratar.

Se trata de un sistema conversacional que ayudará al usuario en la toma de una decisión, en la realización de una tarea, o en la investigación de un tema en concreto. Para ello, le dará soporte para tomar las decisiones, podrá funcionar como un sistema de diagnóstico o le mostrará recomendaciones sobre dominio del que tenga información.

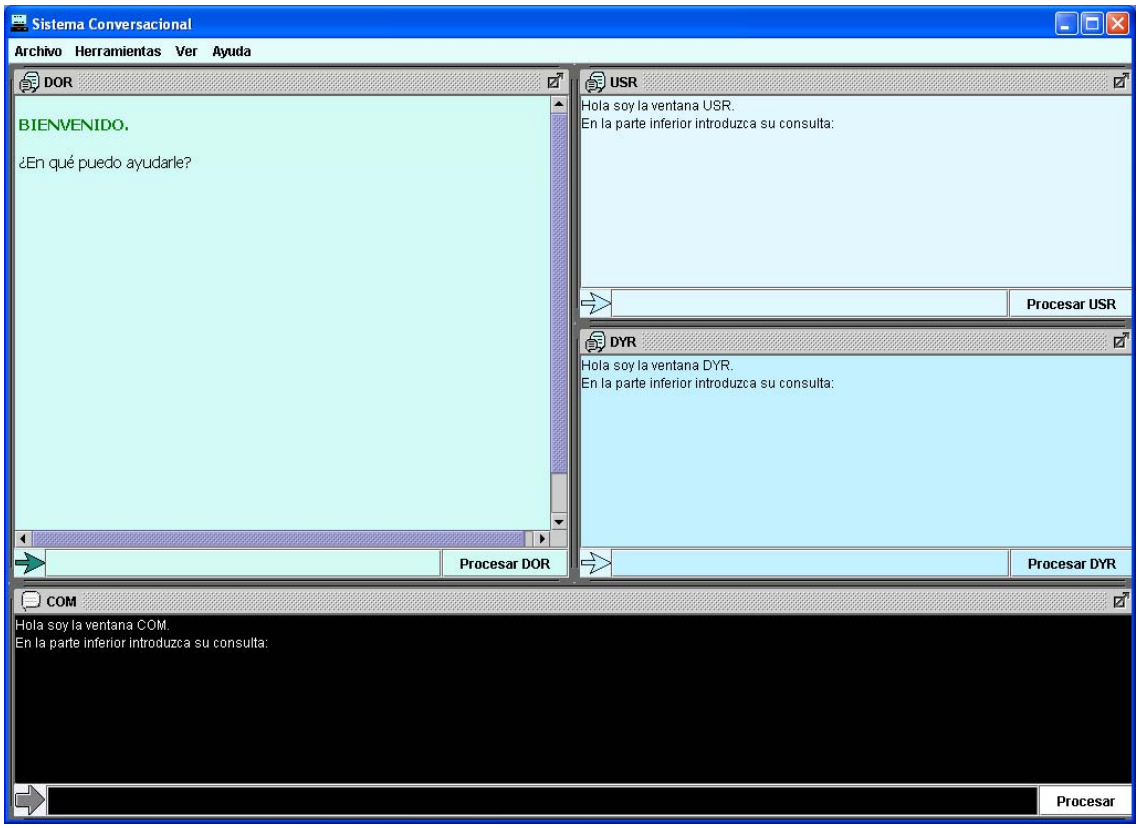
Para poder ofrecer estos servicios, está dividido en tres subtareas:

1. *Dominio*: es la que se encarga de los temas o contenidos de la aplicación. Está representada por la ventana DOR. En ella se mostrará la información al usuario, y se le guiará para llegar al caso que el usuario necesita consultar.
2. *Usuario*: adapta la respuesta global dependiendo de lo que ya sabe y su experiencia. Está representada por la ventana USR.
3. *Dinámica de la Conversación*: complementa la respuesta con un aspecto afectivo basado en el estado del usuario. Usa estrategias de la conversación. Está representada por la ventana DYR.

INICIACIÓN AL APLICATIVO

Al arrancar la aplicación, aparecerá una ventana de presentación. Para acceder simplemente debe de pulsar en el botón "Sistema Conversacional Inteligente"

Una vez dentro, el sistema le da la bienvenida y le ofrece su ayuda. Además se pueden ver las ventanas correspondientes con cada una de las tareas.



Si desea introducir una consulta, puede hacerlo en cualquiera de los campos de texto destinados a tal fin.

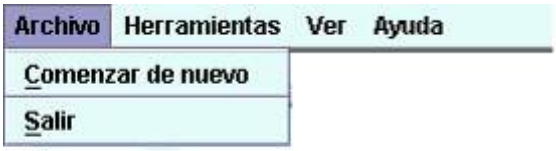
Si la consulta la introduce en el campo de la ventana COM, esta ventana la enviará a la ventana DOR para que la procese y pueda mostrarle la información al respecto.

Todas las ventanas interactúan entre si, por lo que es normal que si escribe su consulta en una de ellas, las demás le muestren información, advertencias, o le pidan su participación para poder ayudarle a encontrar lo que busca.

MENÚ DE LA APLICACIÓN

La aplicación cuenta con el siguiente menú:

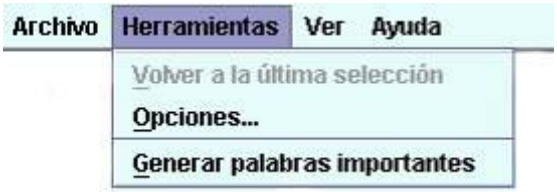
- **Archivo**:



  - Comenzar de nuevo → Seleccione esta opción de menú si desea comenzar una nueva sesión, de manera que se limpie todo lo que muestra la aplicación en ese momento.

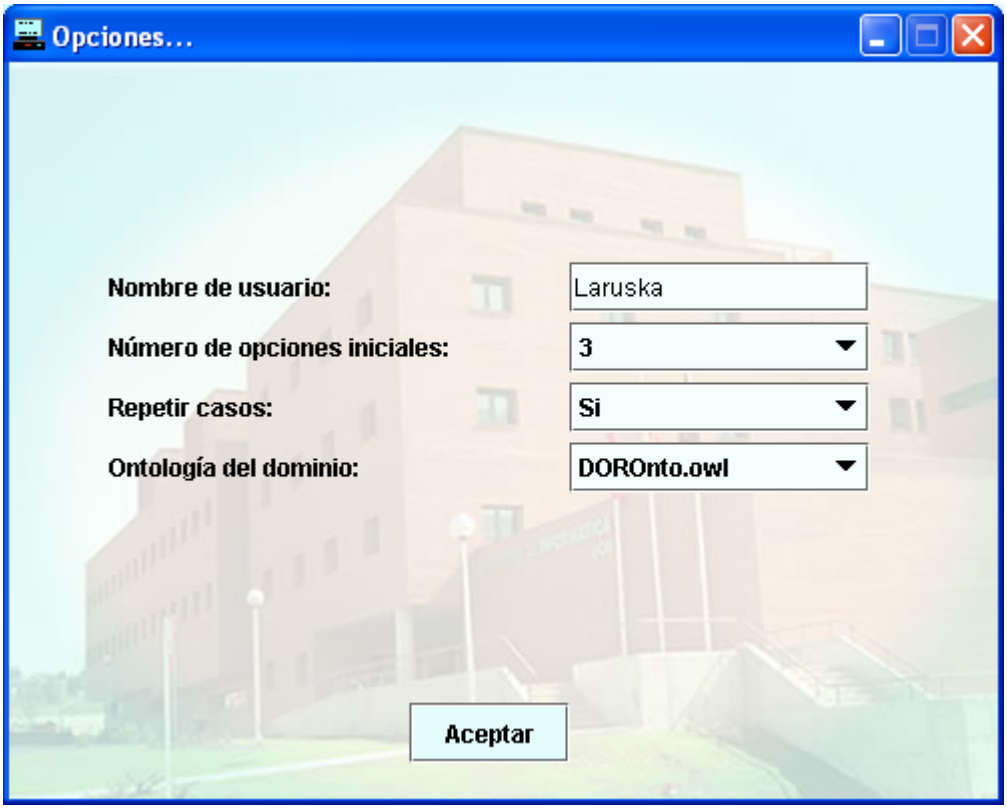
  - Salir → Esta opción le permite salir de la aplicación.

- **Herramientas**:



  - Volver a la última selección → Está opción de menú únicamente se activa cuando anteriormente se mostraron varias opciones al usuario y no se seleccionó ninguna. Eligiendo esta opción, el sistema vuelve a mostrar las alternativas anteriores, ofreciendo de nuevo, la posibilidad de elección al usuario.

- Opciones… → Si desea cambiar alguna de las opciones de la aplicación, puede hacerlo pulsando sobre esta opción de menú.



Aparecerá una ventana donde podrá cambiar:

- El nombre de usuario
- El número de opciones que la aplicación le mostrará una vez que realice una consulta.
- La ontología sobre la que la aplicación trabaja, y de la que obtiene la información.
- Su elección de querer repetir o no los casos.
  (Nótese que esta opción es debido a que la ventana USR no está en funcionamiento).

- Generar palabras importantes → Al ejecutar esta opción del menú Herramientas, usted puede obtener las palabras importantes de un archivo que se encuentre en su sistema. De manera que si el archivo está en formato XML, la aplicación produce un archivo:

  "nombreDeSuArchivo_salidaTratada.txt"

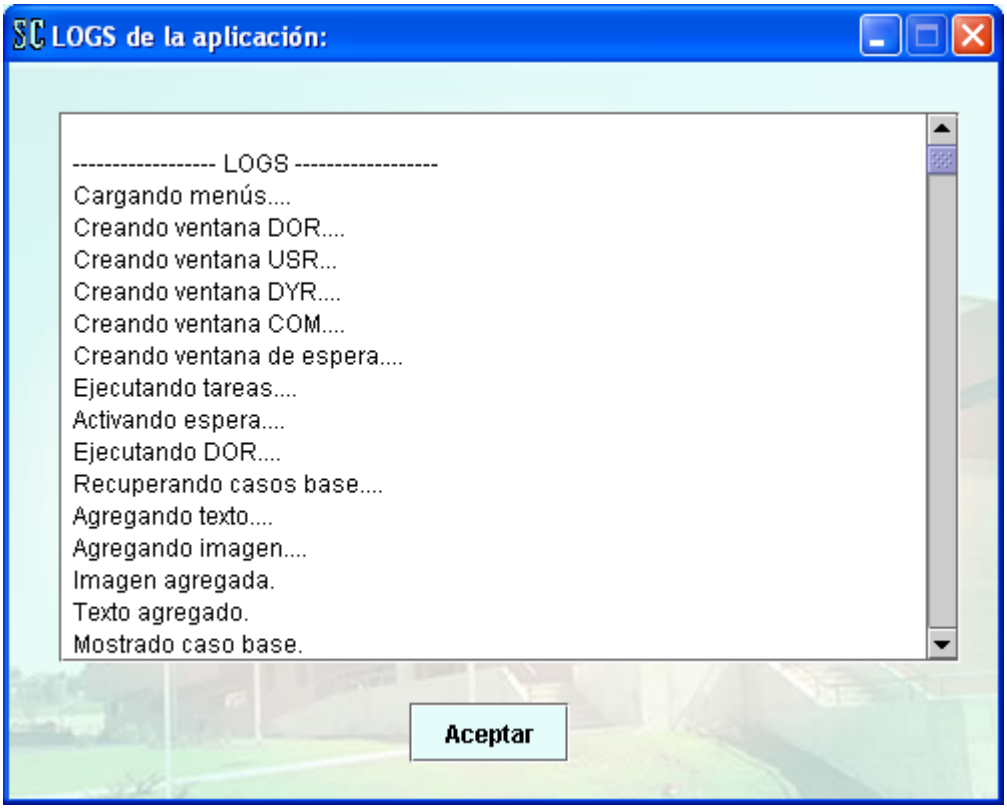
  Donde se encontrará el texto del archivo inicial, sin las marcas (etiquetas) XML.
  El archivo donde encontrará la lista de las palabras importantes será:

  "nombreDeSuArchivo_palabrasImportantes.txt"

Esta opción es útil cuando desee crear una ontología o incluir nuevos casos en una ya existente, y tenga que obtener las palabras importantes que ésta aplicación utilizará.

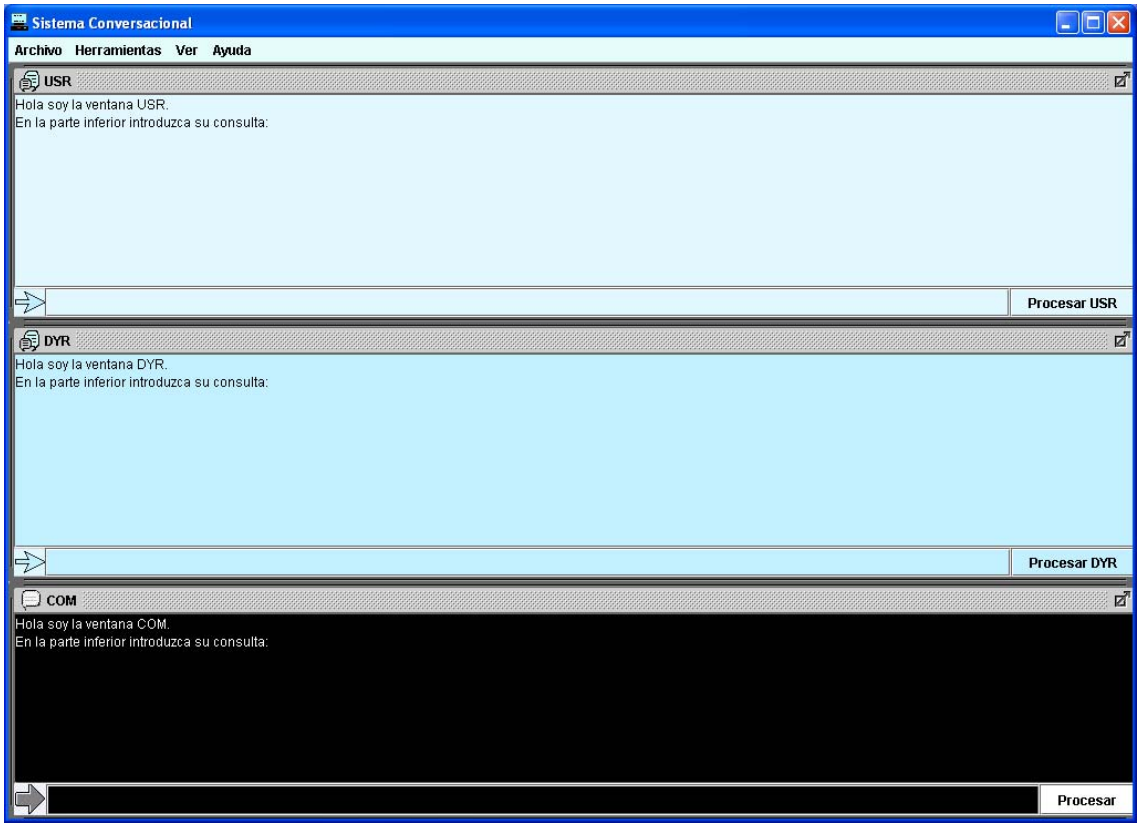
Ver:

- Ver contenido del archivo de LOGS → Esta opción le permite visualizar los LOGS que la aplicación ha ido guardando en la ejecución actual.



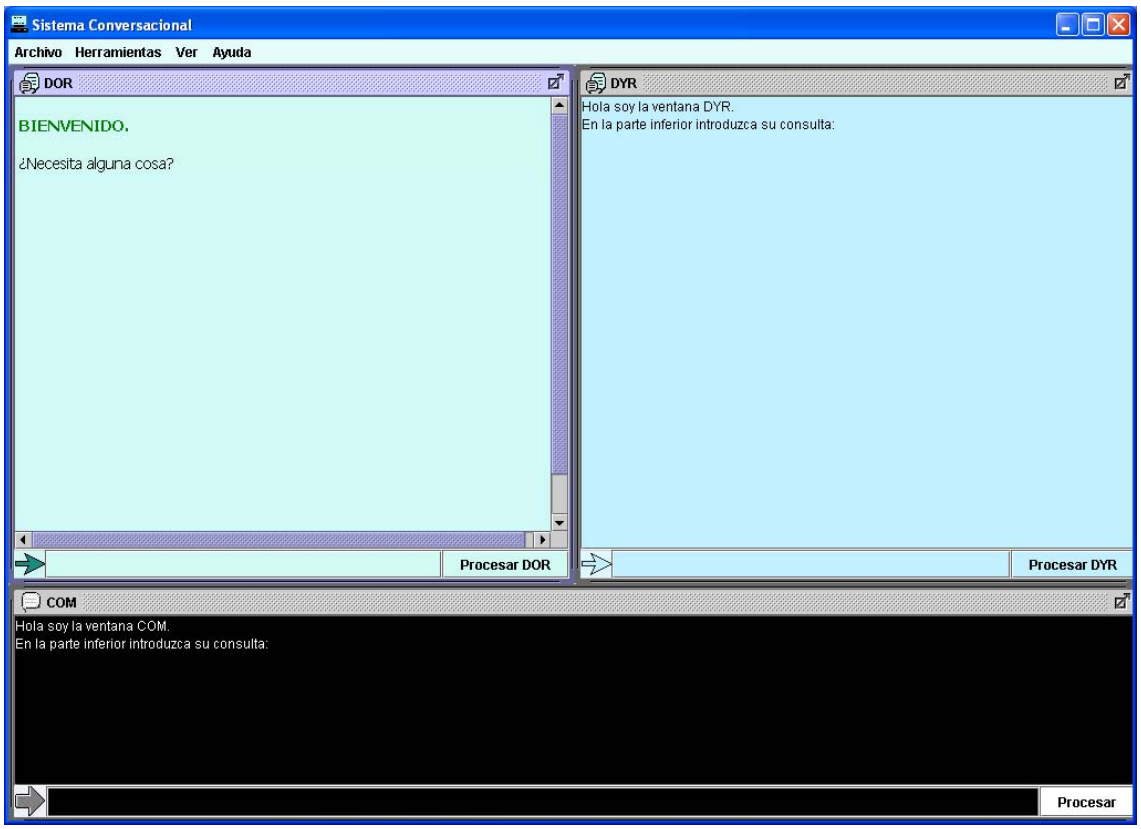
- ☑ Ver DOR → Puede seleccionar o deseleccionar esta opción, de forma que la aplicación muestra o no respectivamente, la ventana DOR. Redimensionando oportunamente las ventanas USR y DYR.

En la siguiente imagen se muestra a modo de ejemplo como quedaría la ventana de la aplicación si **no** se seleccionará esta opción de menú.



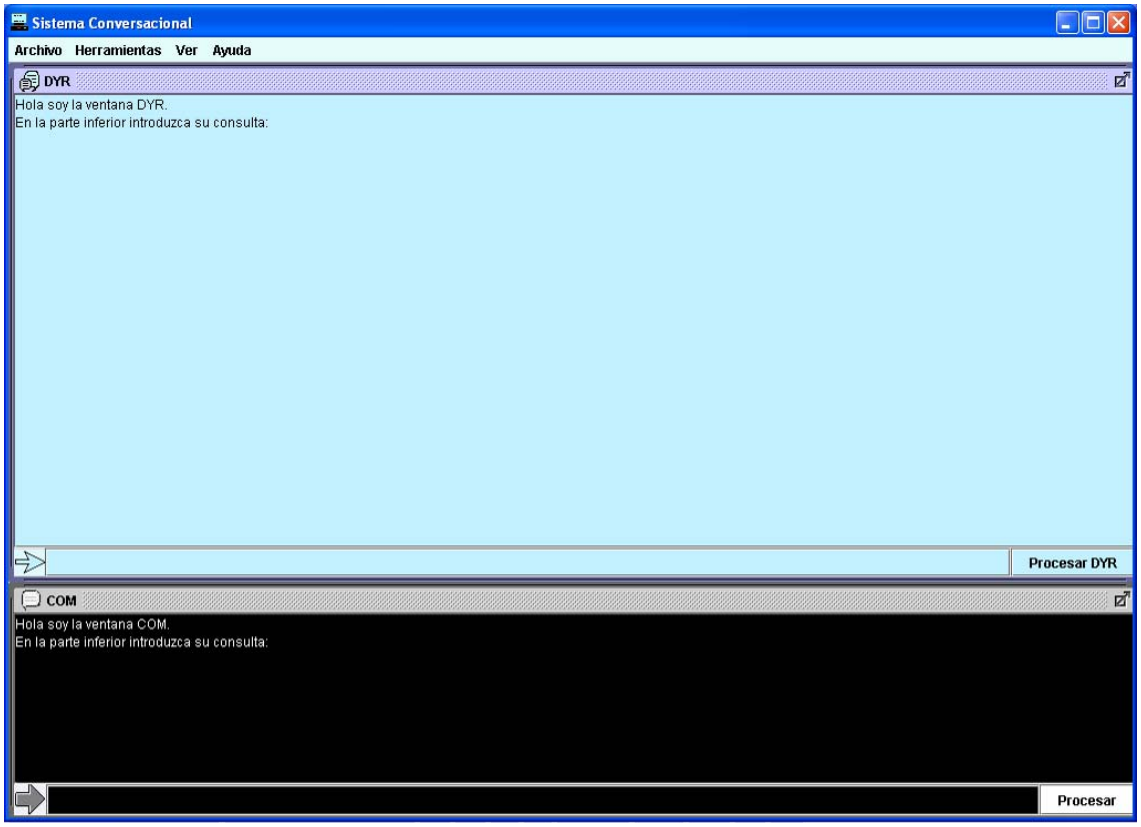
- ☑ Ver USR → Puede seleccionar o deseleccionar esta opción, de forma que la aplicación muestra o no respectivamente, la ventana USR. Redimensionando oportunamente las ventanas DOR y DYR.

En la siguiente imagen se muestra a modo de ejemplo como quedaría la ventana de la aplicación si **no** se seleccionará esta opción de menú.

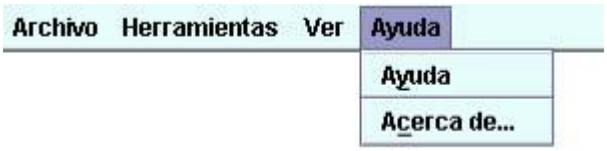


- ☑ Ver DYR → Puede seleccionar o deseleccionar esta opción, de forma que la aplicación muestra o no respectivamente, la ventana DYR. Redimensionando oportunamente las ventanas USR y DOR.

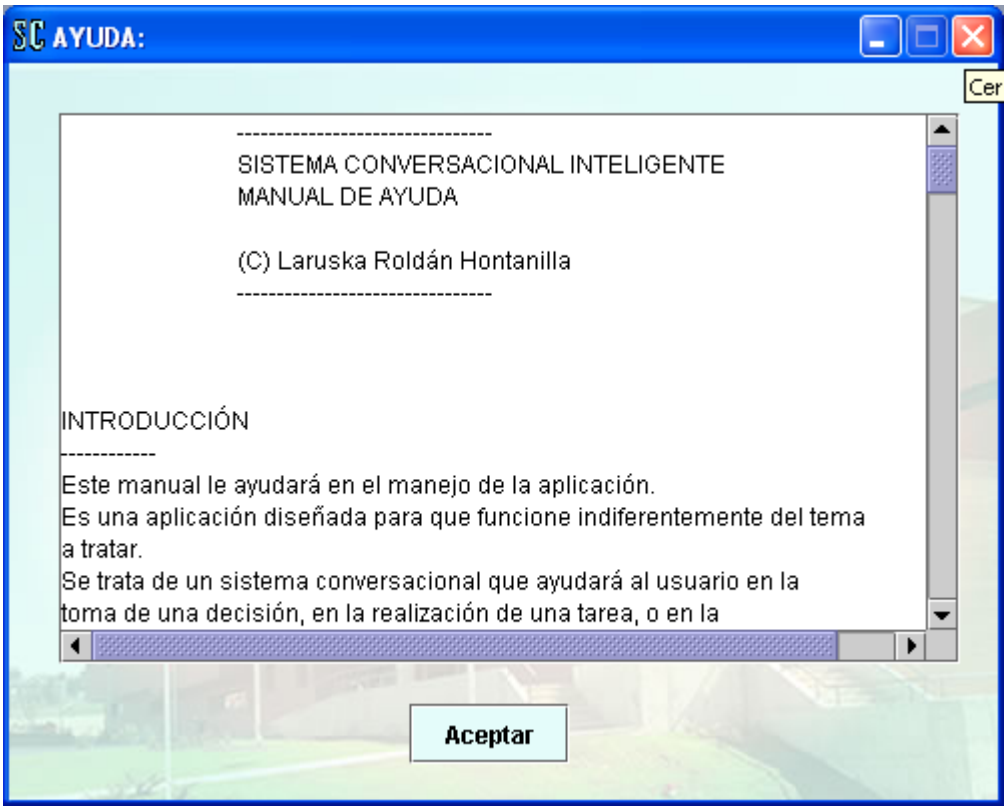
En la siguiente imagen se muestra a modo de ejemplo como quedaría la ventana de la aplicación si seleccionará esta opción de menú y no seleccionará las dos anteriores.



Ayuda:



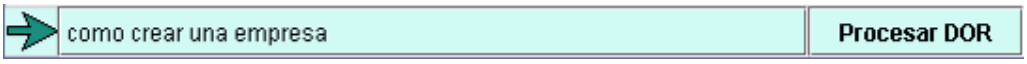
- Ayuda → Aquí es donde encontrará el manual de ayuda de la aplicación. Si su sistema tiene una aplicación para visualizar archivos en formato PDF, puede incluir la ruta de dicha aplicación en el archivo "rutaPDF.sib". Con ello, esta aplicación le mostrará esta ayuda en dicho formato. Sino, se mostrará la ayuda en formato plano.



- Acerca de… → Para ver los créditos de la aplicación.

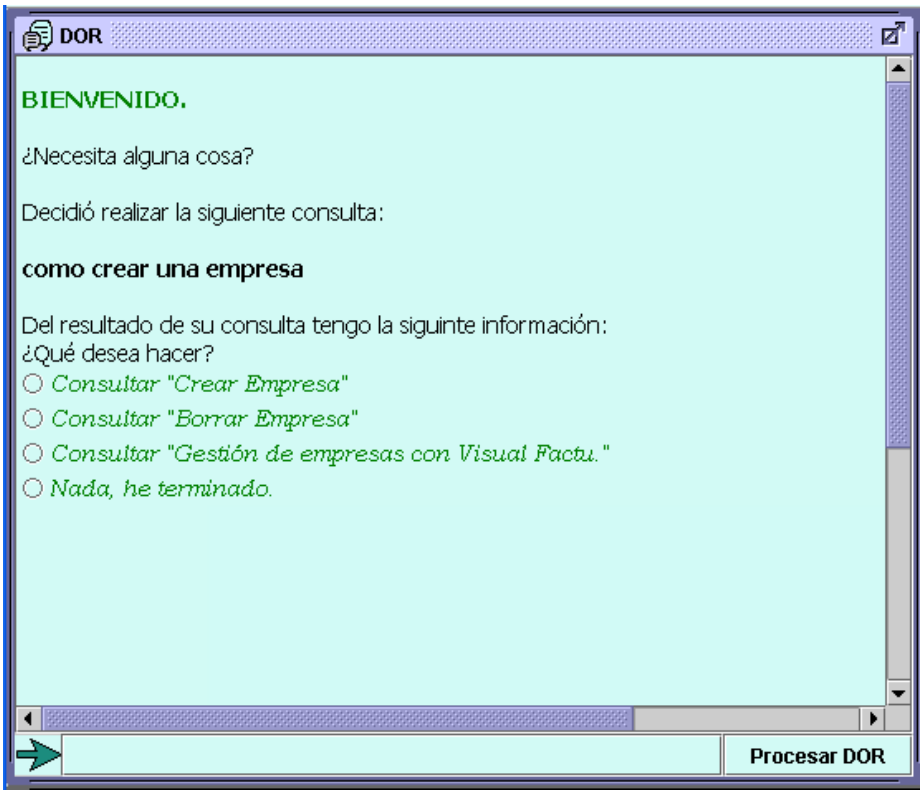
EN EJECUCIÓN

Para introducir una consulta, únicamente será necesario que la escriba en el cuadro de texto de una de las ventanas de la aplicación.



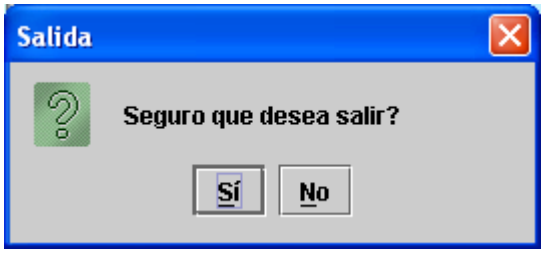
Una vez que haya escrito su consulta, pulse sobre el botón inmediatamente a la derecha del cuadro de texto donde la introdujo. Para hacerle más fácil el trabajo, también tiene la posibilidad de pulsar la tecla ENTER.

Durante la ejecución de la aplicación, el sistema puede ofrecerle la selección de una de un caso:

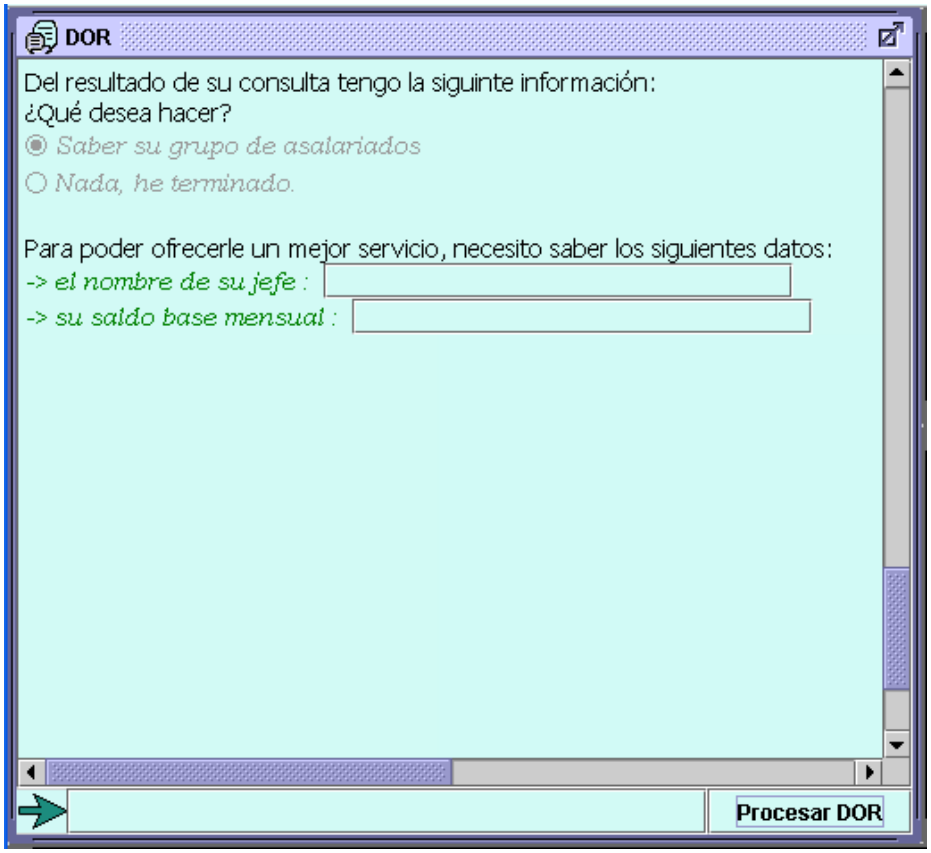


Para elegir uno de ellos, sólo debe pulsar en el que desee, de manera que el botón redondo de su izquierda se seleccione. Una vez hecho esto, pulse sobre el botón “Procesar” de la ventana en la que se encuentre. La aplicación continuará su proceso.

Si selecciona “Nada, he terminado”, la aplicación le pedirá su confirmación.



En alguna de sus consultas, puede ser que el aplicativo necesite que rellene algunos datos:



Rellene los campos con sus datos. Una vez hecho esto, pulse sobre el botón “Procesar” de la ventana en la que se encuentre. La aplicación continuará su proceso.

5.2- MANUAL DEL SISTEMA:

REQUISITOS DEL SISTEMA

Para poder arrancar Eclipse, los requerimientos necesarios es tener instalado el JDK 1.3 o superior y preferiblemente tener 256 MB de memoria. Esta IDE puede funcionar en Windows, Linux, Solaris, etc.
Para el funcionamiento de Pellet se requiere de una JVM compatible con la versión 1.4 de Sun o superior.

INSTALACIÓN

Se ha creado un paquete de instalación automático para windows, el cual guía de manera automática al usuario en la instalación de la aplicación. De todas formas, tan solo es necesaria la extracción del archivo comprimido "SistemaConversacional.rar", en el directorio raíz del ordenador.

Para el sistema operativo, Linux, es necesario, aparte de la extracción del archivo anteriormente mencionado, editar los archivos de las ontologías, cambiando la línea de importación de ontología, por el path donde se tenga colocada la ontología CCBROnto.

Ejemplo:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.owl-ontologies.com/DYRonto.owl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:CCBROnto="http://www.owl-ontologies.com/CCBRonto.owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://www.owl-ontologies.com/DYRonto.owl">
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="file:///C:/SistemaConversacional/ontologias/CCBROnto.owl"/>
  </owl:Ontology>
.....
.....
.....
```

5.3- PARA EL CÓDIGO:

JAVADOC

La documentación del código de la aplicación se encuentra en la carpeta “doc”. También se puede acceder haciendo clic aquí.

6. BIBLIOGRAFÍA

- ARANO, Silvia. "La ontología: una zona de interacción entre la Lingüística y la Documentación". Hipertext.net, núm. 2, 2003. http://www.hipertext.net/web/pag220.htm

- Protegé
  http://protege.semanticweb.org
  http://protege.stanford.edu/

- Pellet
  http://www.mindswap.org/2003/pellet/index.shtml

- BECHLOFER, Sean. "DAML+OIL is not enough". SWW-1, Semantic Web working symposium. Stanford (CA), July 29th-August 1 st, 2001.
  http://potato.cs.man.ac.uk/papers/not-enough.pdf

- W3C. *OWL Web Ontology Language Guide.*
  http://www.w3.org/TR/owl-guide/

- Sitio Oficial de Jena 2
  http://jena.sourceforge.net/

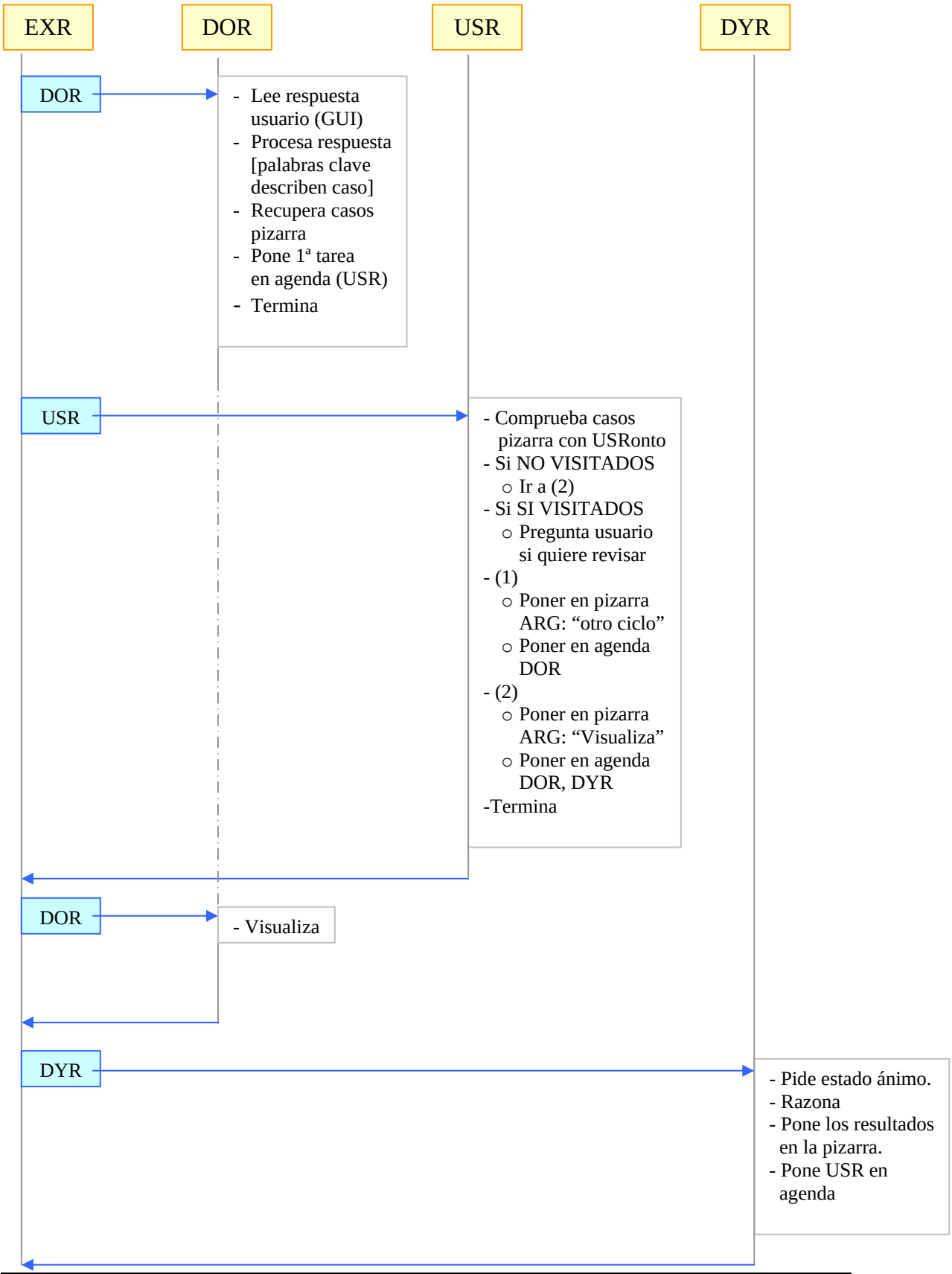
- Jena 2 Ontology API
  http://jena.sourceforge.net/ontology/

- XML, Servicios Web y Web Semántica
  http://www.di.uniovi.es/~labra/cursos/ext04/pres/SemWeb.pdf

- Instalación de un entorno de desarrollo Eclipse
  http://www.desarrolloweb.com/articulos/1692.php

ANEXO 1

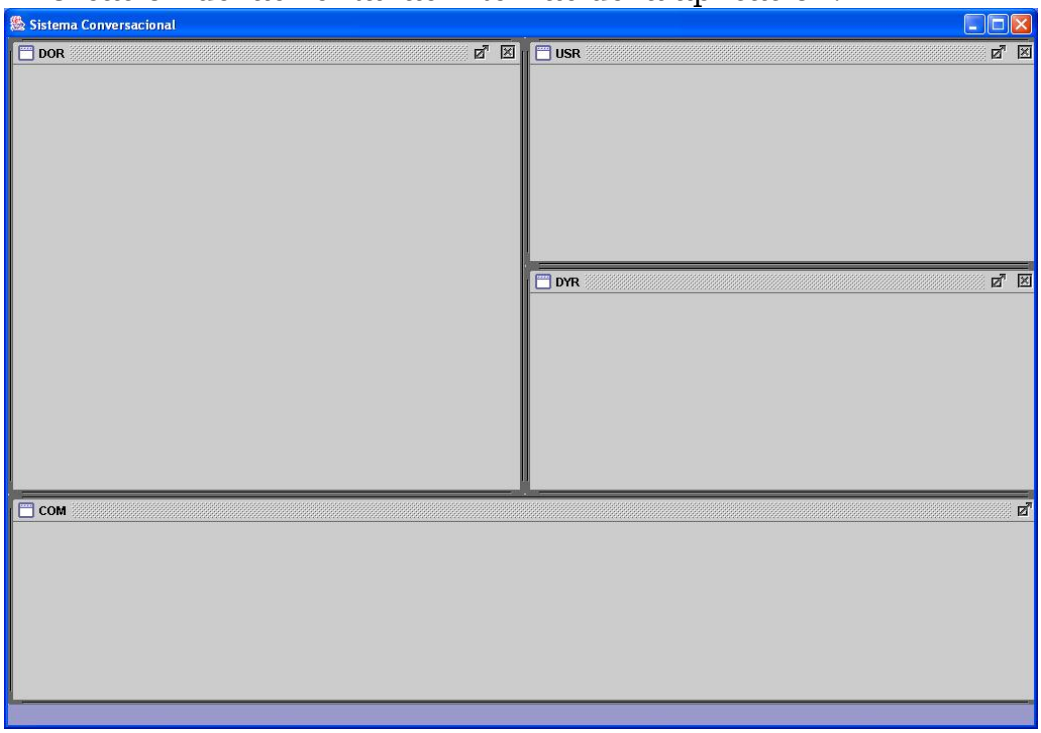
EJEMPLO CICLO GLOBAL DE EJECUCIÓN



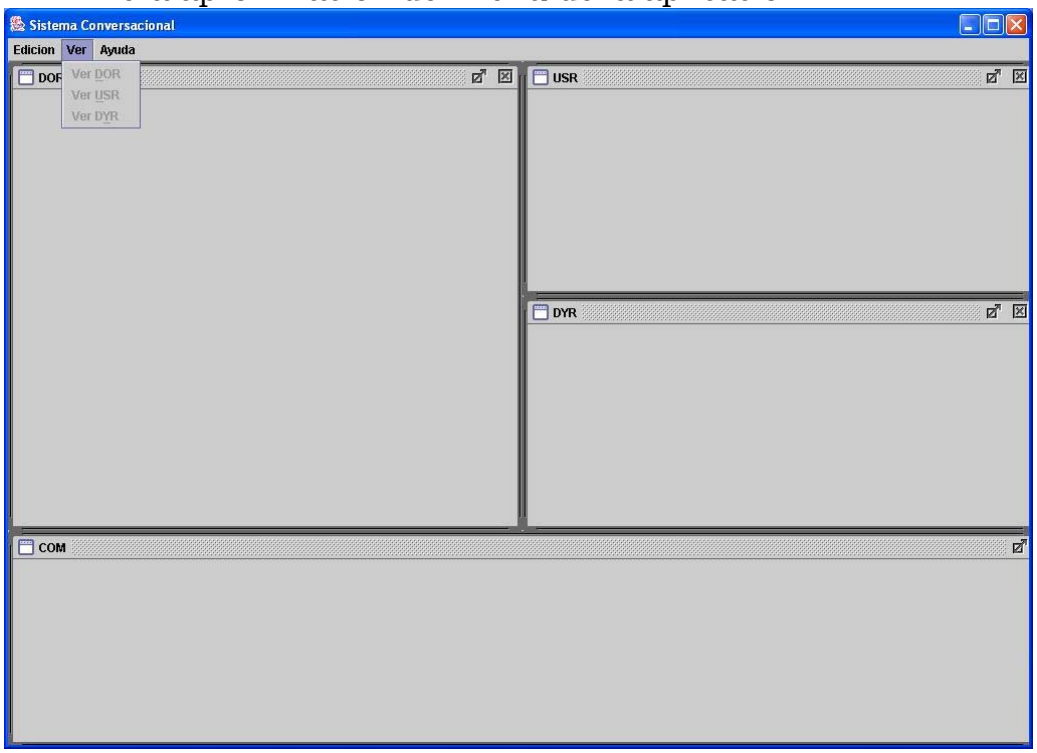
ANEXO 2

CRECIMIENTO INICIAL DE LA INTERFAZ GRÁFICA

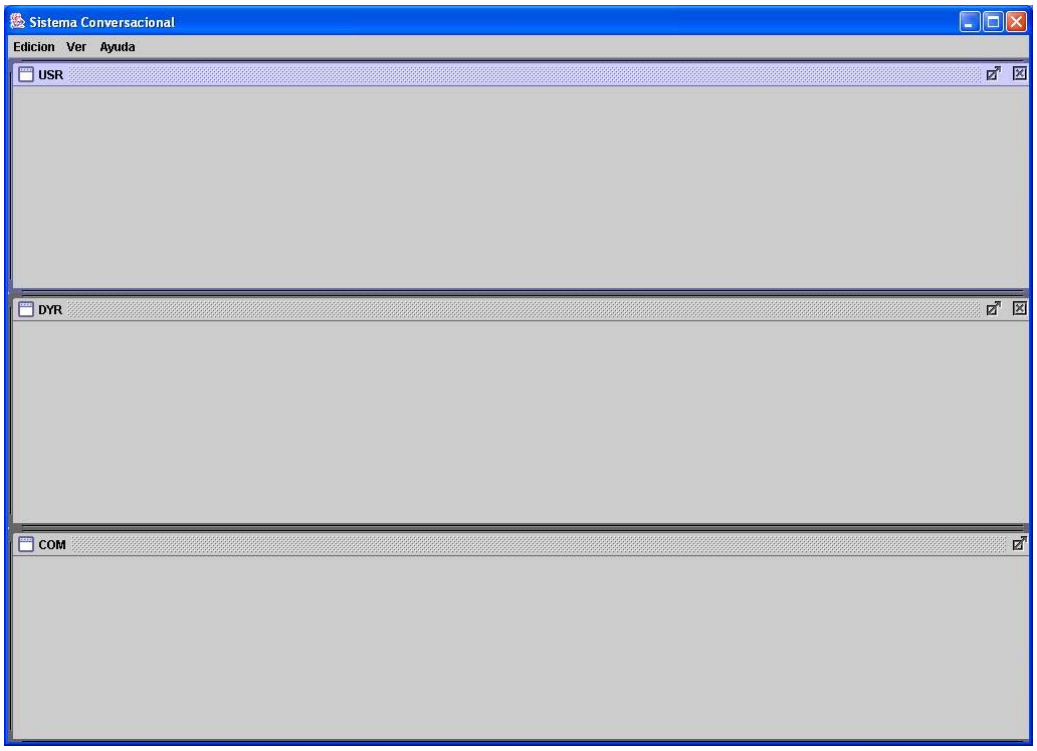
→ Creación de las ventanas internas de la aplicación.



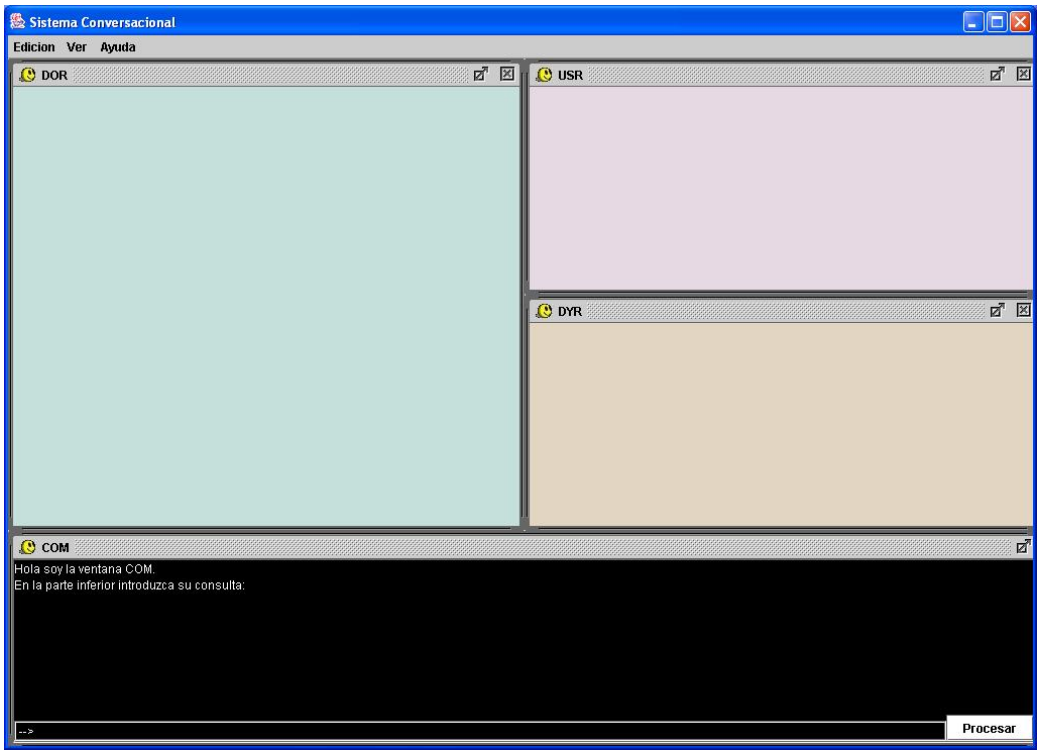
→ Primera aproximación del menú de la aplicación



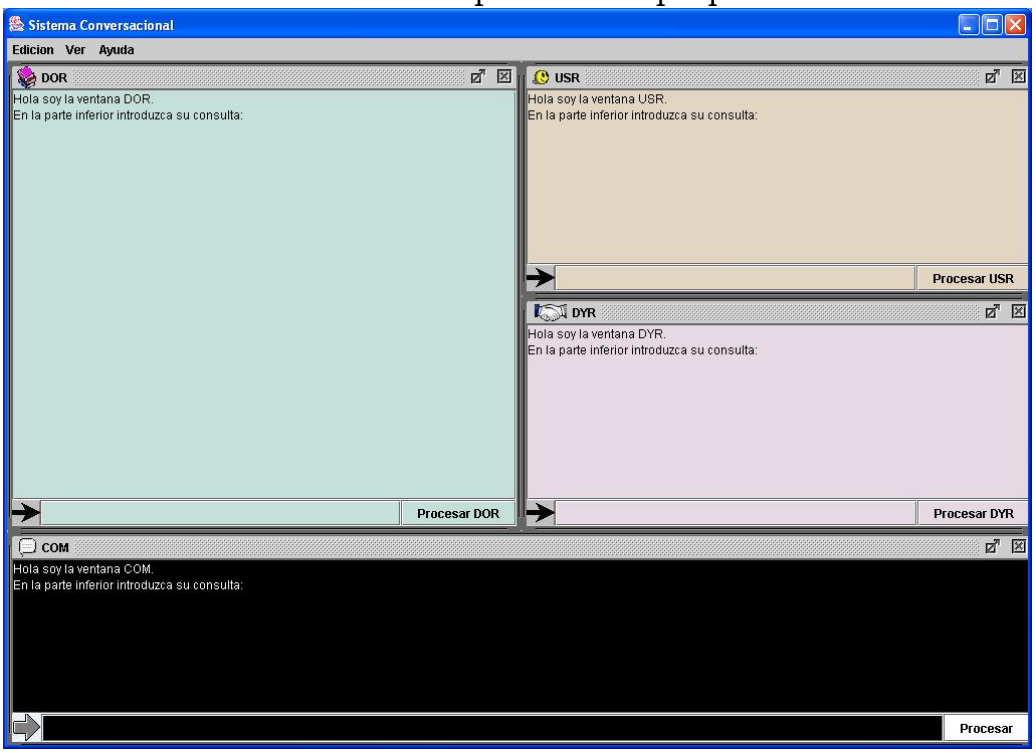
→ Redimensionamiento de las ventanas



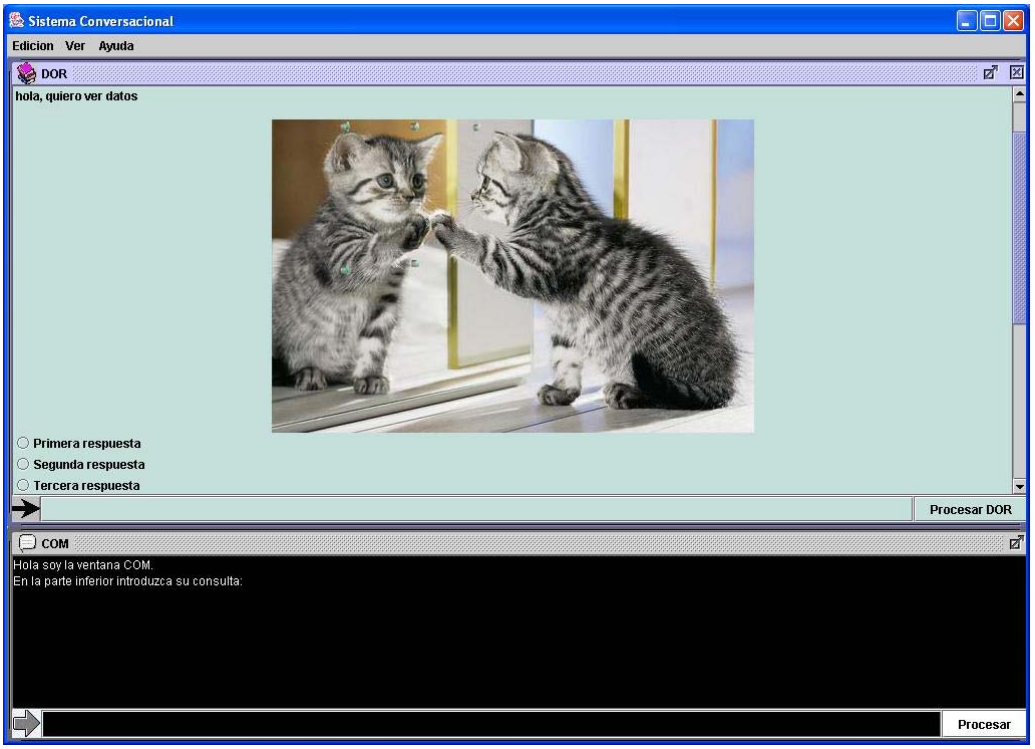
→ Cada ventana tiene una visión diferente



→ En cada una existe un campo de texto propio.

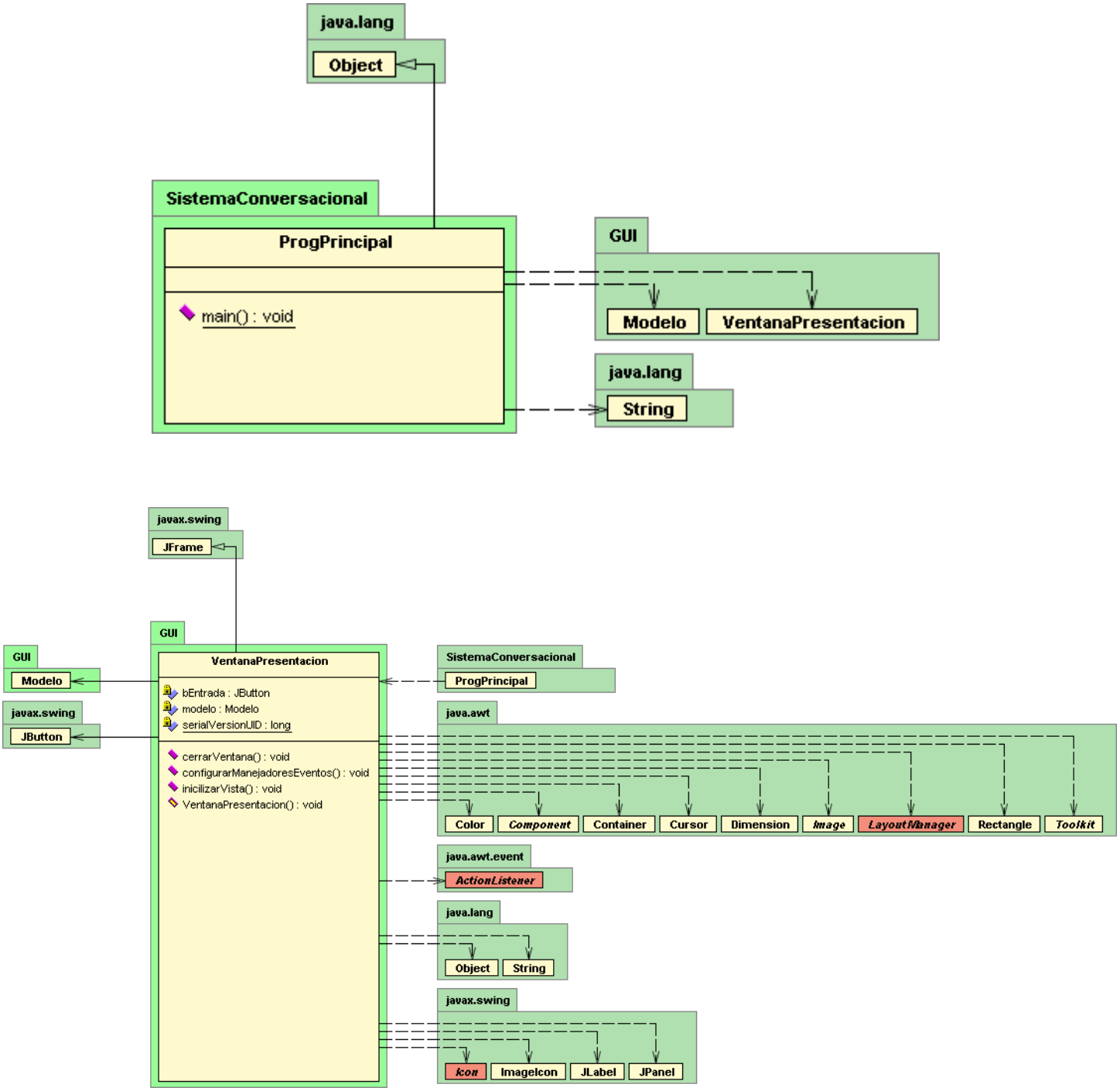


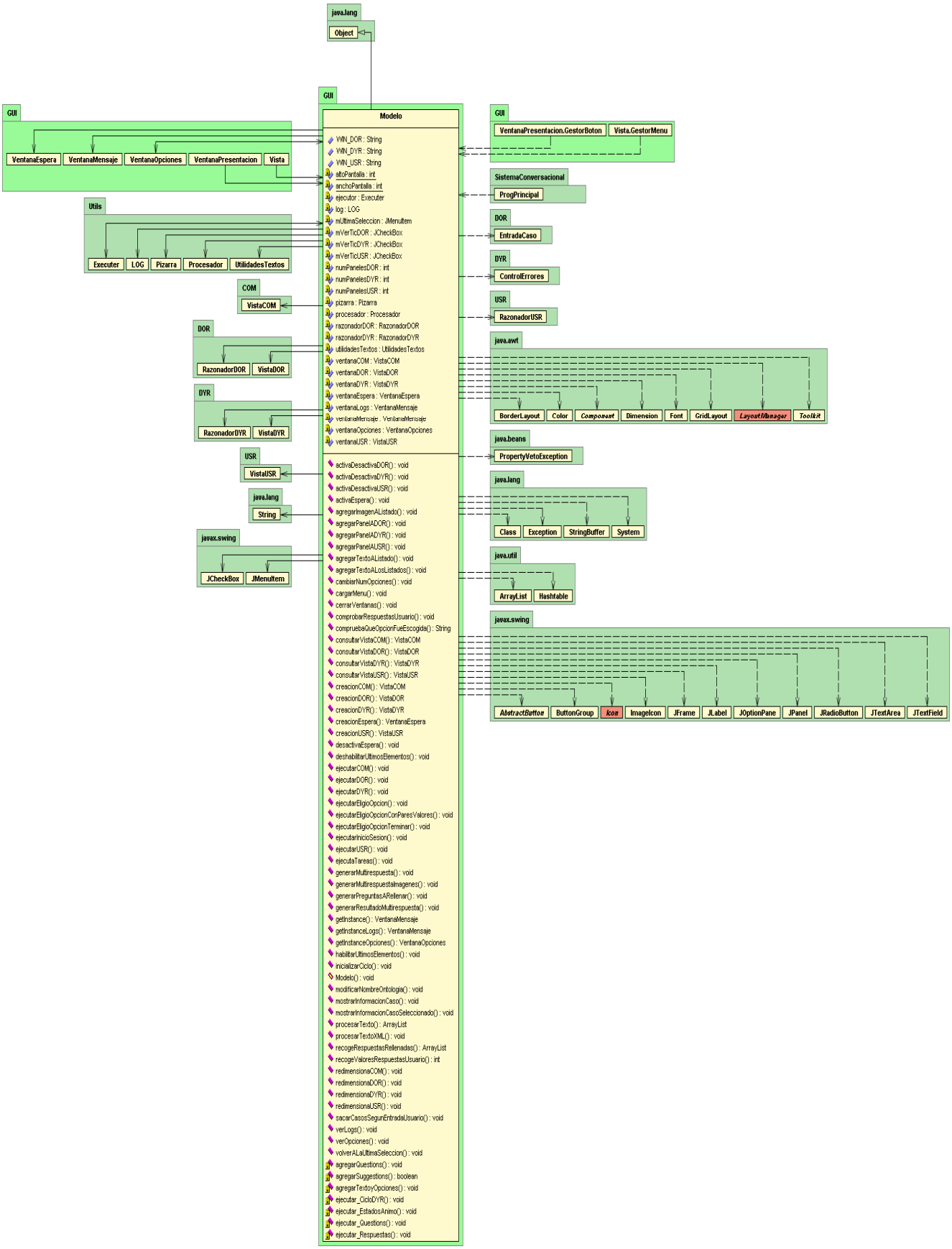
→ Se incluye la posibilidad de mostrar imágenes y múltiples selecciones.

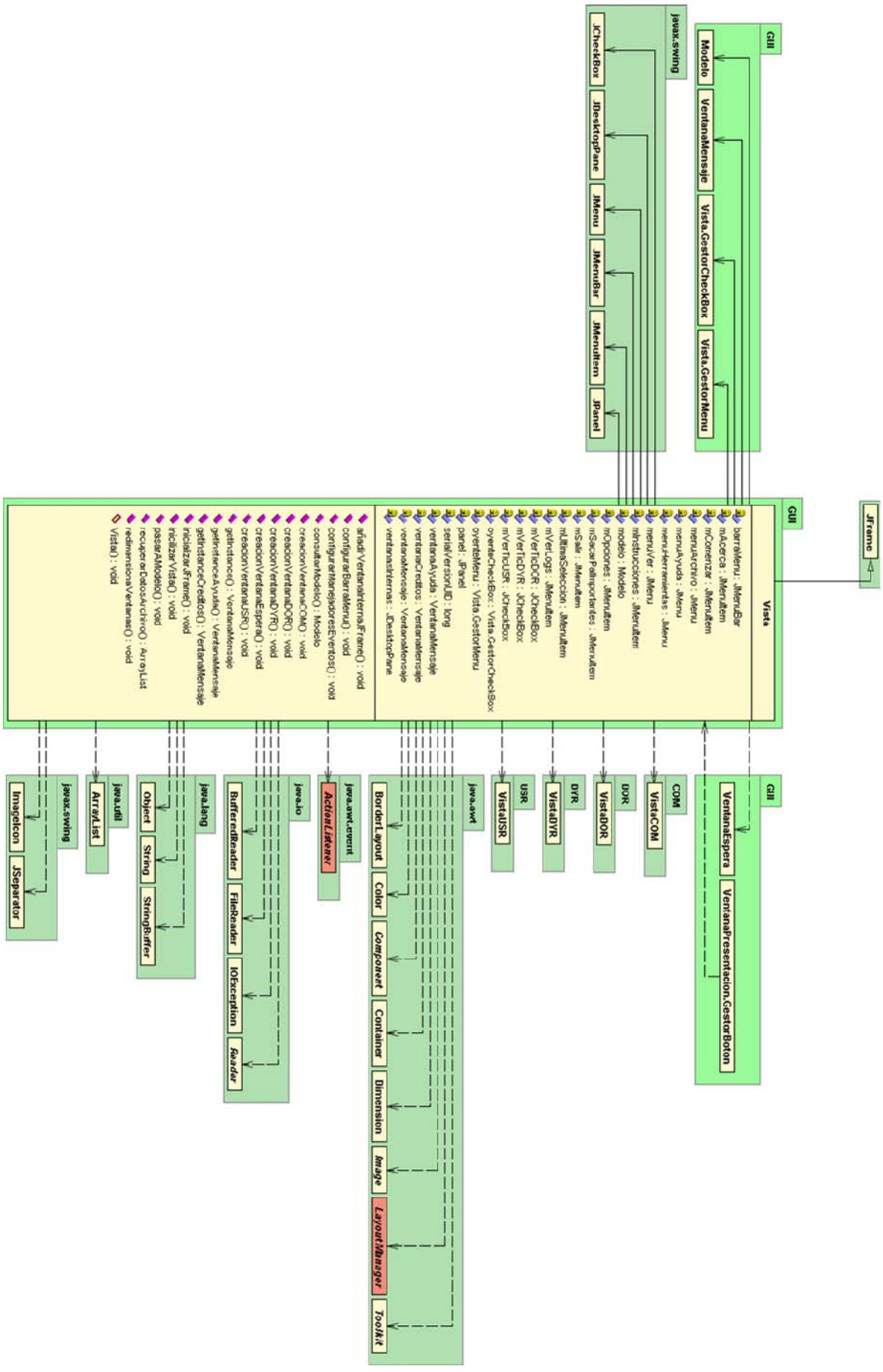


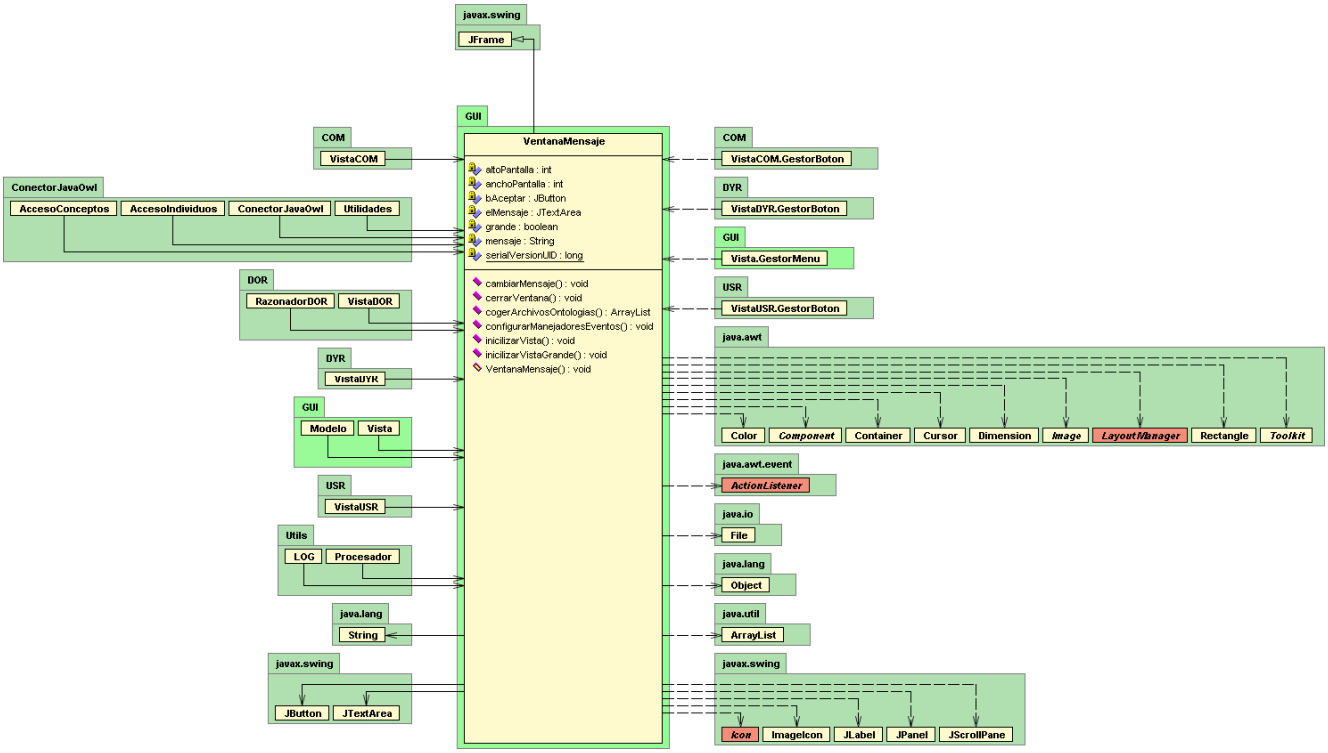
ANEXO 3

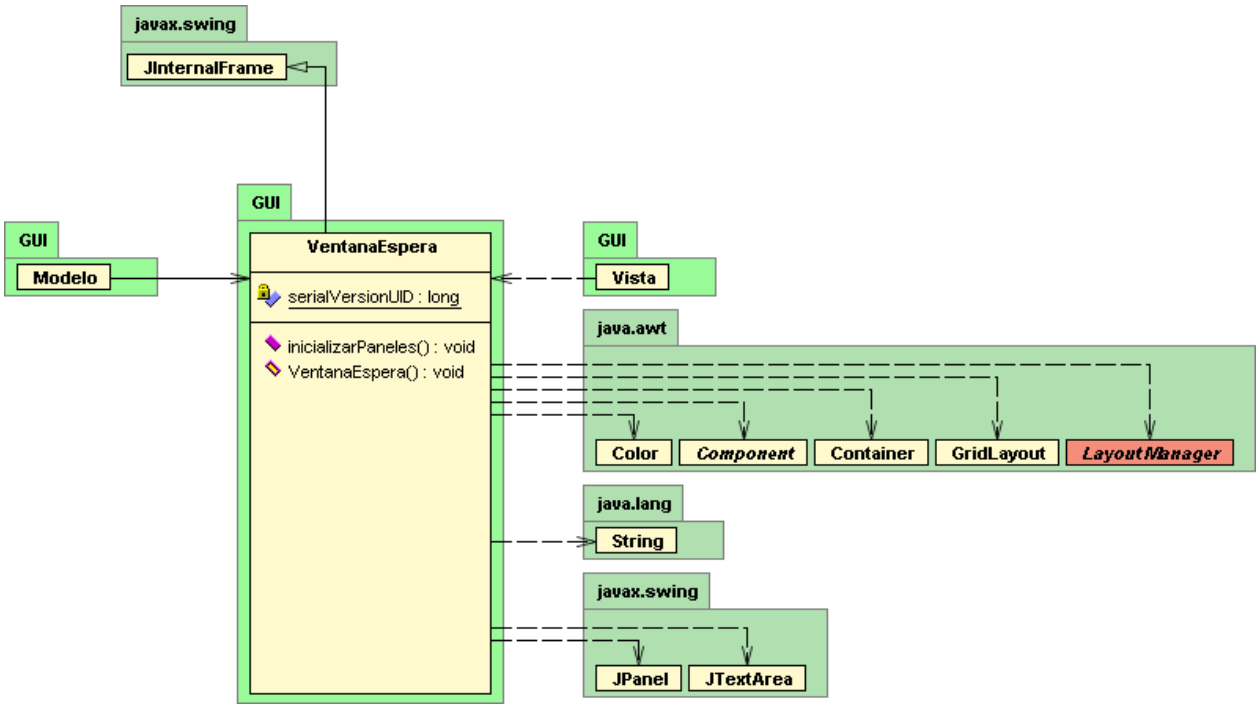
DIAGRAMAS UML

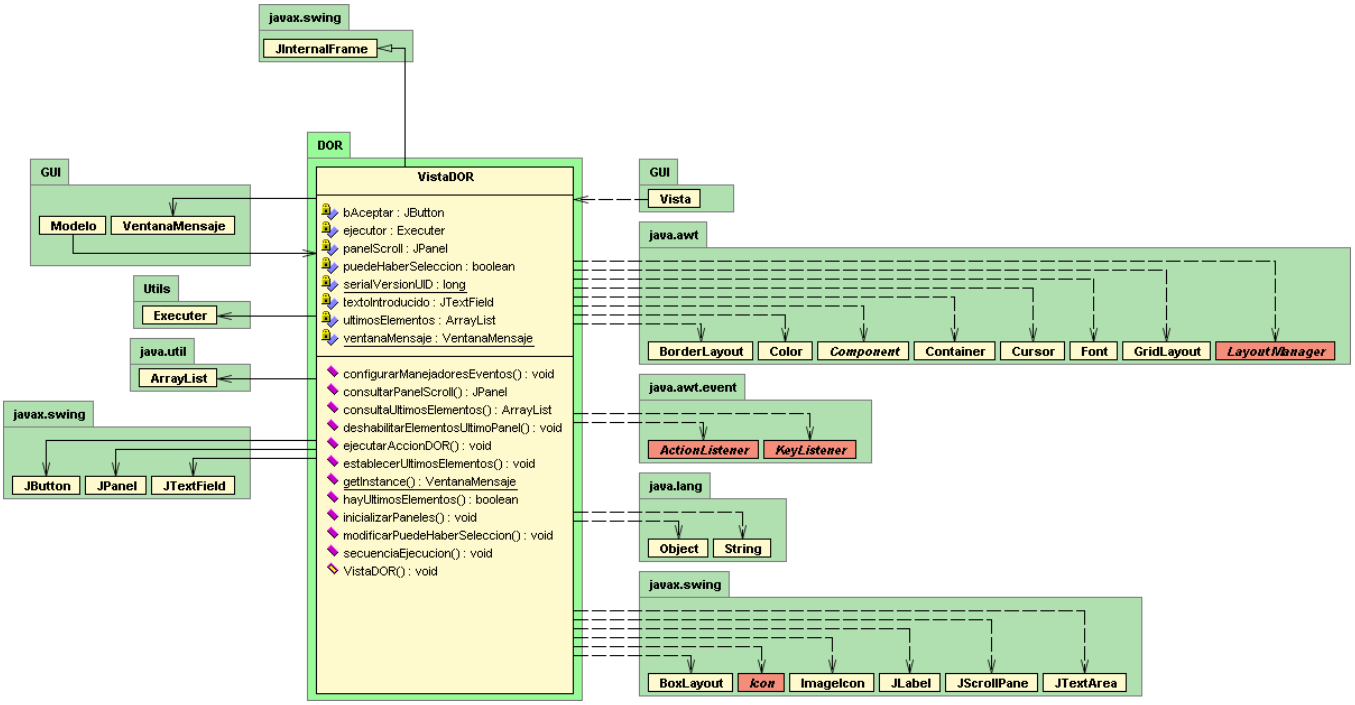


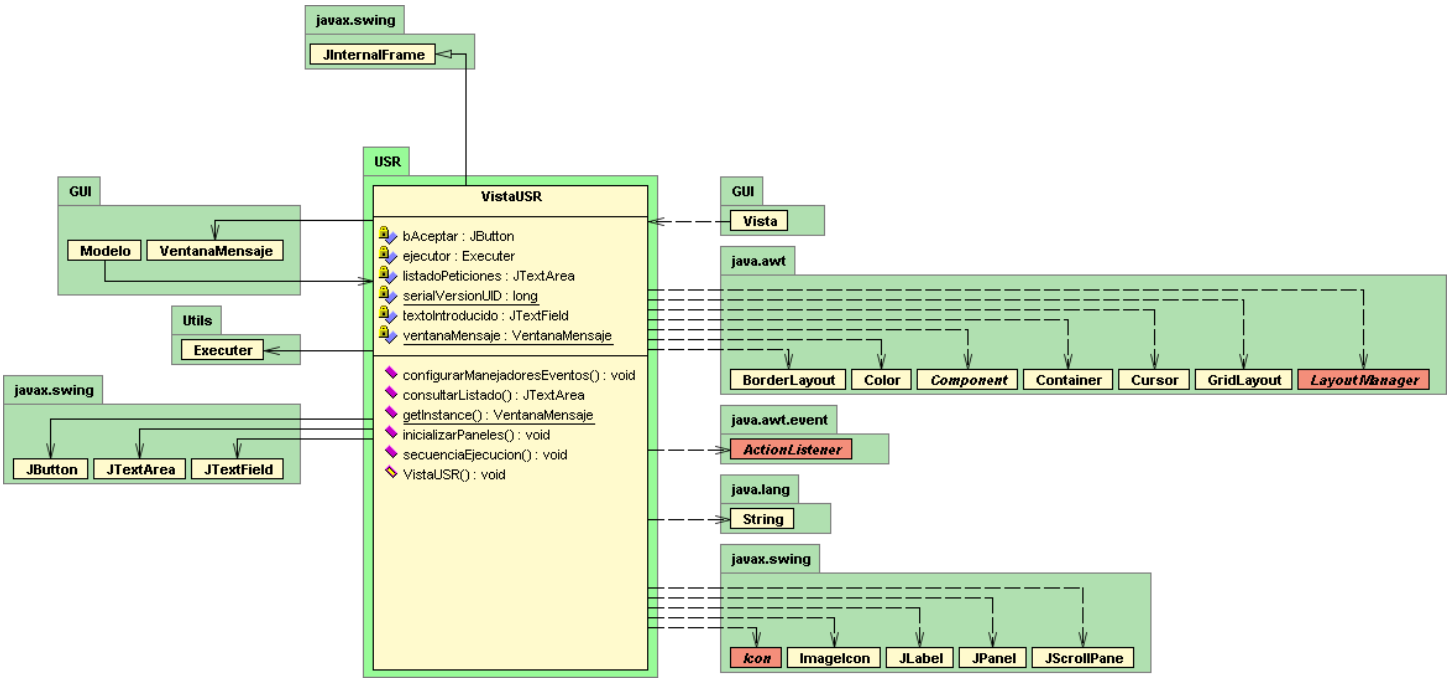

java.lang
Object
GUI
Modelo
WN_DOR : String
WN_DYR : String
WN_USR : String
altoPantalla : int
anchoPantalla : int
ejecutor : Executer
log : LOG
miUltimaSeleccion : JMenuItem
mVerTicDOR : JCheckBox
mVerTicDYR : JCheckBox
mVerTicUSR : JCheckBox
numPanelesDOR : int
numPanelesDYR : int
numPanelesUSR : int
pizarra : Pizarra
procesador : Procesador
razonadorDOR : RazonadorDOR
razonadorDYR : RazonadorDYR
utilidadesTextos : UtilidadesTextos
ventanaCOM : VistaCOM
ventanaDOR : VistaDOR
ventanaDYR : VistaDYR
ventanaEspera : VentanaEspera
ventanaLogs : VentanaMensaje
ventanaMensaje : VentanaMensaje
ventanaOpciones : VentanaOpciones
ventanaUSR : VistaUSR
activaDesactivaDOR() : void
activaDesactivaDYR() : void
activaDesactivaUSR() : void
activaEspera() : void
agregarImagenAListado() : void
agregarPanelADOR() : void
agregarPanelADYR() : void
agregarPanelAUSR() : void
agregarTextoAListado() : void
agregarTextoALosListados() : void
cambiarNumOpciones() : void
cargarMenu() : void
cerrarVentanas() : void
comprobarRespuestasUsuario() : void
compruebaQueOpcionFueEscogida() : String
consultarVistaCOM() : VistaCOM
consultarVistaDOR() : VistaDOR
consultarVistaDYR() : VistaDYR
consultarVistaUSR() : VistaUSR
creacionCOM() : VistaCOM
creacionDOR() : VistaDOR
creacionDYR() : VistaDYR
creacionEspera() : VentanaEspera
creacionUSR() : VistaUSR
desactivaEspera() : void
deshabilitarUltimosElementos() : void
ejecutarCOM() : void
ejecutarDOR() : void
ejecutarDYR() : void
ejecutarElgioOpcion() : void
ejecutarElgioOpcionConParesValores() : void
ejecutarElgioOpcionTerminar() : void
ejecutarInicioSesion() : void
ejecutarUSR() : void
ejecutaTareas() : void
generarMultirespuesta() : void
generarMultirespuestaImagenes() : void
generarPreguntasARellenar() : void
generarResultadoMultirespuesta() : void
getInstance() : VentanaMensaje
getInstanceLogs() : VentanaMensaje
getInstanceOpciones() : VentanaOpciones
habilitarUltimosElementos() : void
inicializarCiclo() : void
Modelo() : void
modificarNombreOntologia() : void
mostrarInformacionCaso() : void
mostrarInformacionCasoSeleccionado() : void
procesarTexto() : ArrayList
procesarTextoXML() : void
recogeRespuestasRellenadas() : ArrayList
recogeValoresRespuestasUsuario() : int
redimensionaCOM() : void
redimensionaDOR() : void
redimensionaDYR() : void
redimensionaUSR() : void
sacarCasosSegunEntradaUsuario() : void
verLogs() : void
verOpciones() : void
volverALaUltimaSeleccion() : void
agregarQuestions() : void
agregarSuggestions() : boolean
agregarTextoyOpciones() : void
ejecutar_CicloDYR() : void
ejecutar_EstadosAnimo() : void
ejecutar_Questions() : void
ejecutar_Respuestas() : void
GUI
VentanaEspera
VentanaMensaje
VentanaOpciones
VentanaPresentacion
Vista
Utils
Executer
LOG
Pizarra
Procesador
UtilidadesTextos
COM
VistaCOM
DOR
RazonadorDOR
VistaDOR
DYR
RazonadorDYR
VistaDYR
USR
VistaUSR
java.lang
String
javax.swing
JCheckBox
JMenuItem
GUI
VentanaPresentacion.GestorBoton
Vista.GestorMenu
SistemaConversacional
ProgPrincipal
DOR
EntradaCaso
DYR
ControlErrores
USR
RazonadorUSR
java.awt
BorderLayout
Color
Component
Dimension
Font
GridLayout
LayoutManager
Toolkit
java.beans
PropertyVetoException
java.lang
Class
Exception
StringBuffer
System
java.util
ArrayList
Hashtable
javax.swing
AbstractButton
ButtonGroup
Icon
ImageIcon
JFrame
JLabel
JOptionPane
JPanel
JRadioButton
JTextArea
JTextField

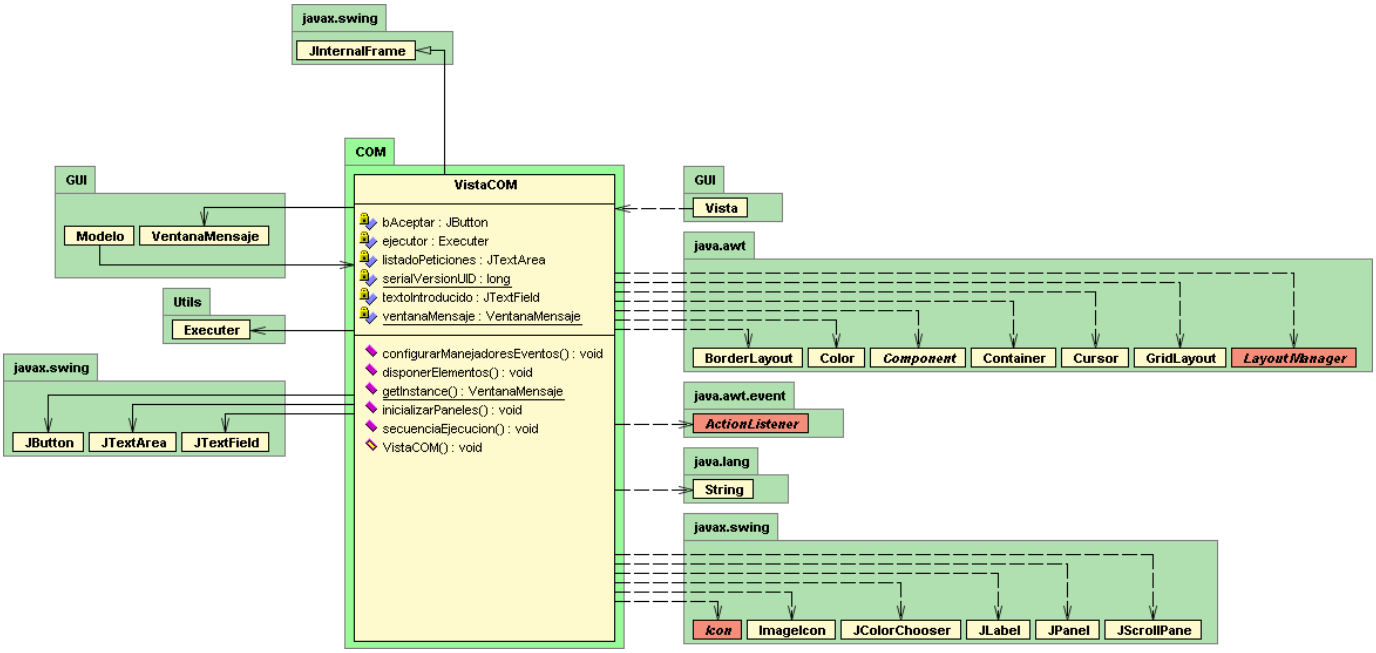

GUI
Modelo
VentanaMensaje
Vista.GestorCheckBox
Vista.GestorMenu
javax.swing
JCheckBox
JDesktopPane
JMenu
JMenuBar
JMenuItem
JPanel
JFrame
Vista
barraMenu : JMenuBar
mAcerca : JMenuItem
mComenzar : JMenuItem
menuArchivo : JMenu
menuAyuda : JMenu
menuHerramientas : JMenu
menuVer : JMenu
mInstrucciones : JMenuItem
modelo : Modelo
mOpciones : JMenuItem
mSacarPalmpuntos : JMenuItem
mSalir : JMenuItem
mUltimaSeleccion : JMenuItem
mVerLogs : JMenuItem
mVerTicDOR : JCheckBox
mVerTicDYR : JCheckBox
mVerTicUSR : JCheckBox
oyenteCheckBox : Vista.GestorCheckBox
oyenteMenu : Vista.GestorMenu
panel : JPanel
serialVersionUID : long
ventanaAyuda : VentanaMensaje
ventanaCreditos : VentanaMensaje
ventanaMensaje : VentanaMensaje
ventanasInternas : JDesktopPane
añadirVentanasInternasJFrame() : void
configurarBarraMenu() : void
configurarManejoporcesEventos() : void
consultarModelo() : Modelo
creacionVentanaCOM() : void
creacionVentanaDOR() : void
creacionVentanaDYR() : void
creacionVentanaEspera() : void
creacionVentanaUSR() : void
getInstance() : VentanaMensaje
getInstanceAyuda() : VentanaMensaje
getInstanceCreditos() : VentanaMensaje
inicializarJFrame() : void
inicializarVista() : void
pasarAModelo() : void
recuperarDatosArchivo() : ArrayList
redimensionaVentanas() : void
Vista() : void
GUI
VentanaEspera
VentanaPresentacion.GestorBoton
COM
VistaCOM
DOR
VistaDOR
DYR
VistaDYR
USR
VistaUSR
java.awt
BorderLayout
Color
Component
Container
Dimension
Image
LayoutManager
Toolkit
java.awt.event
ActionListener
java.io
BufferedReader
FileReader
IOException
Reader
java.lang
Object
String
StringBuffer
java.util
ArrayList
javax.swing
ImageIcon
JSeparator

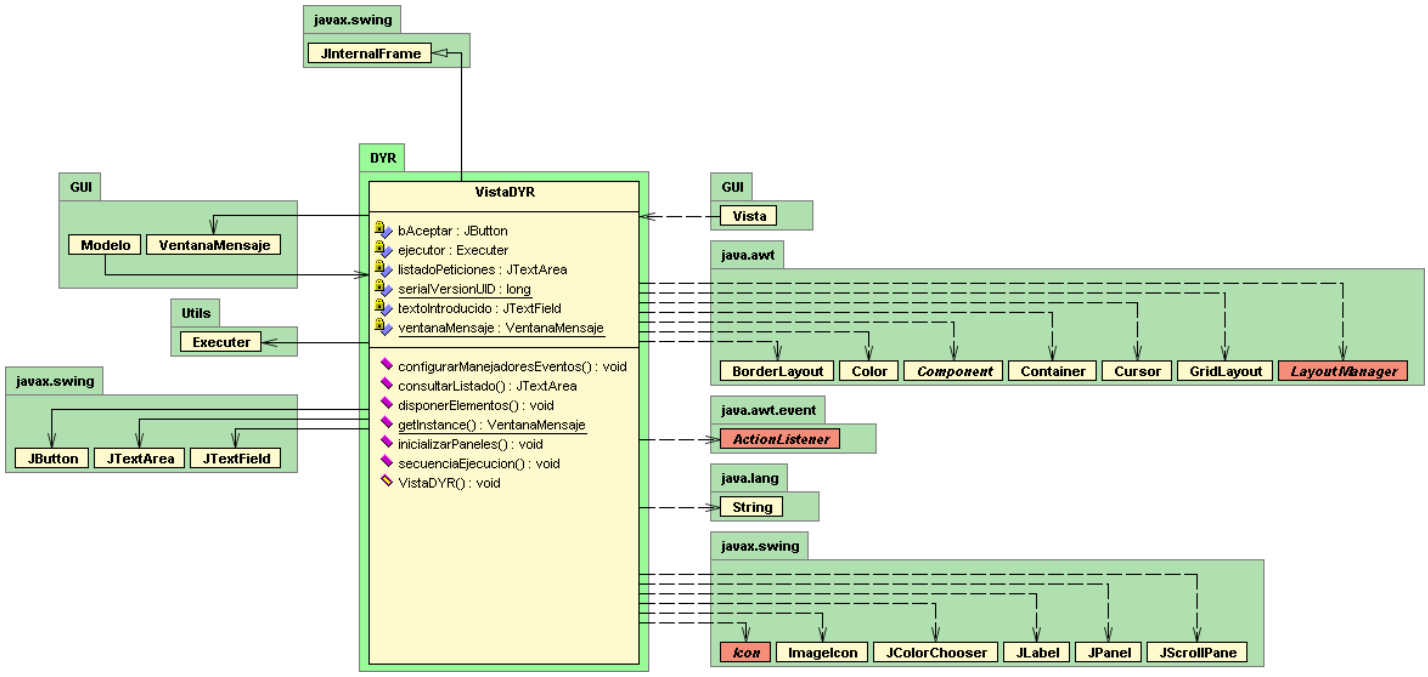

javax.swing
JFrame
GUI
VentanaMensaje
altoPantalla : int
anchoPantalla : int
bAceptar : JButton
elMensaje : JTextArea
grande : boolean
mensaje : String
serialVersionUID : long
cambiarMensaje() : void
cerrarVentana() : void
cogerArchivosOntologias() : ArrayList
configurarManejadoresEventos() : void
inicilizarVista() : void
inicilizarVistaGrande() : void
VentanaMensaje() : void
COM
VistaCOM
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
ConectorJavaOwl
Utilidades
DOR
RazonadorDOR
VistaDOR
DYR
VistaDYR
GUI
Modelo
Vista
USR
VistaUSR
Utils
LOG
Procesador
java.lang
String
javax.swing
JButton
JTextArea
COM
VistaCOM.GestorBoton
DYR
VistaDYR.GestorBoton
GUI
Vista.GestorMenu
USR
VistaUSR.GestorBoton
java.awt
Color
Component
Container
Cursor
Dimension
Image
LayoutManager
Rectangle
Toolkit
java.awt.event
ActionListener
java.io
File
java.lang
Object
java.util
ArrayList
javax.swing
Icon
ImageIcon
JLabel
JPanel
JScrollPane

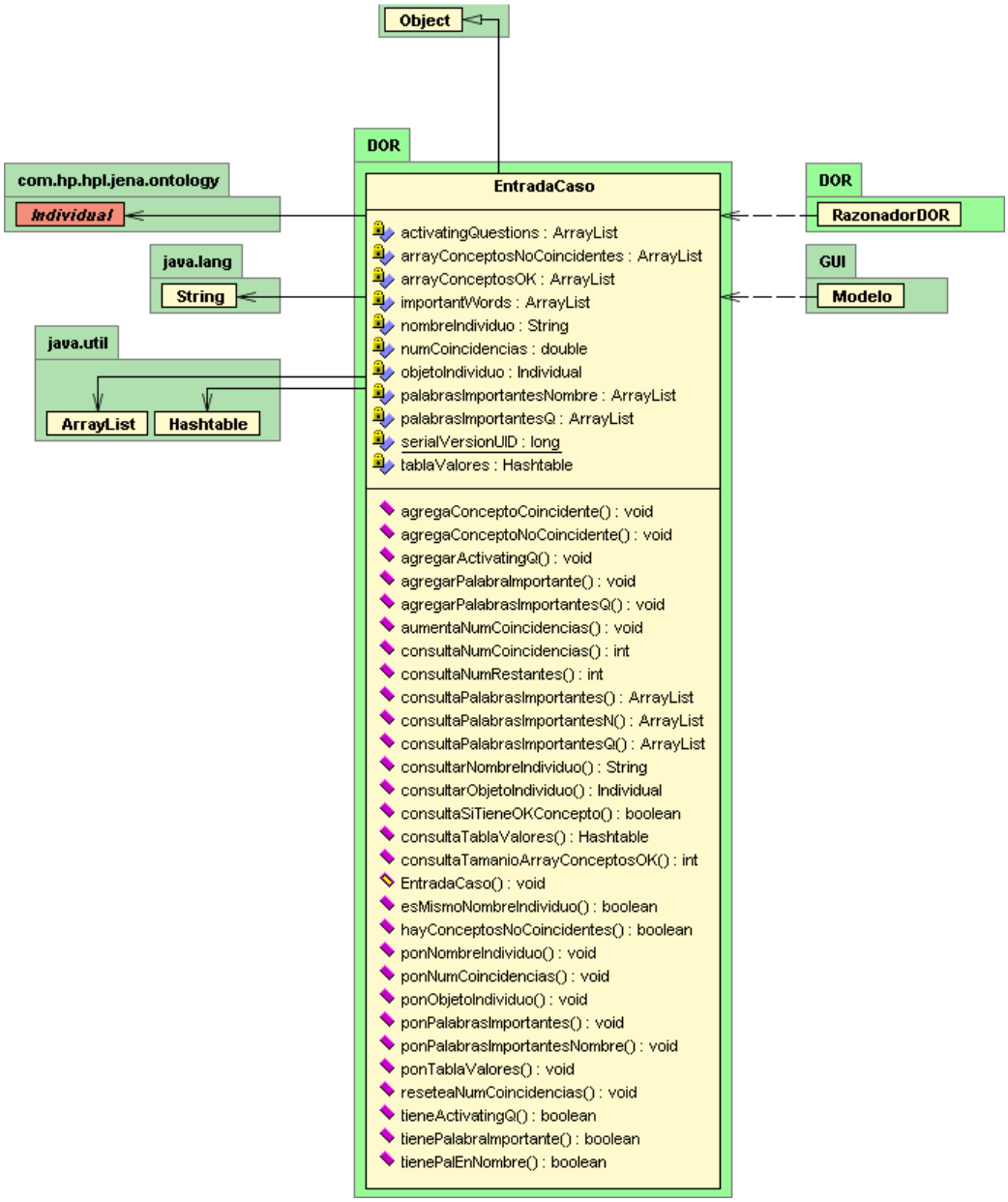

javax.swing
JInternalFrame
GUI
VentanaEspera
serialVersionUID : long
inicializarPaneles() : void
VentanaEspera() : void
GUI
Modelo
GUI
Vista
java.awt
Color
Component
Container
GridLayout
LayoutManager
java.lang
String
javax.swing
JPanel
JTextArea

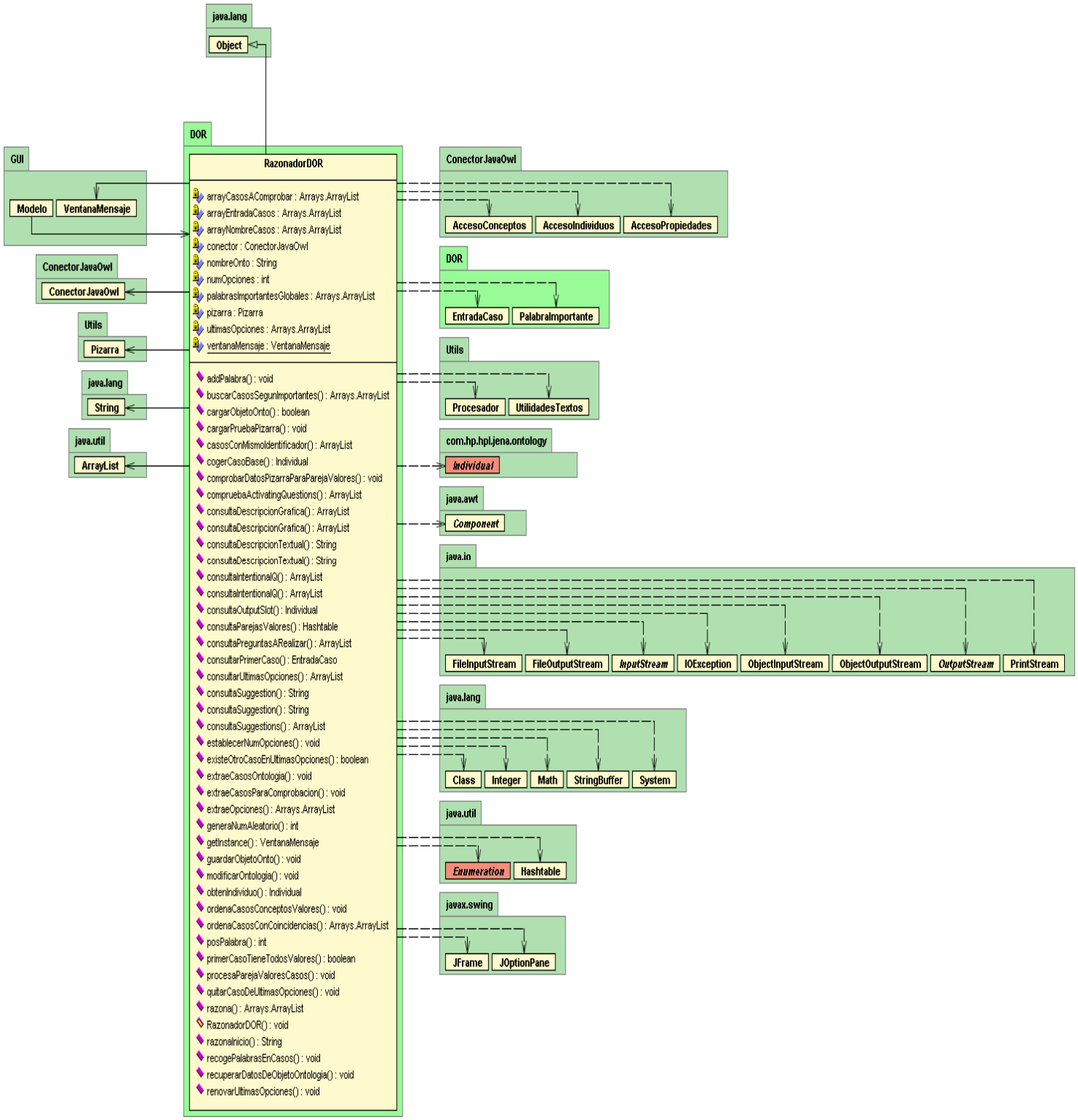

javax.swing
JInternalFrame
DOR
VistaDOR
bAceptar : JButton
ejecutor : Executer
panelScroll : JPanel
puedeHaberSeleccion : boolean
serialVersionUID : long
textoIntroducido : JTextField
ultimosElementos : ArrayList
ventanaMensaje : VentanaMensaje
configurarManejadoresEventos() : void
consultarPanelScroll() : JPanel
consultaUltimosElementos() : ArrayList
deshabilitarElementosUltimoPanel() : void
ejecutarAccionDOR() : void
establecerUltimosElementos() : void
getInstance() : VentanaMensaje
hayUltimosElementos() : boolean
inicializarPaneles() : void
modificarPuedeHaberSeleccion() : void
secuenciaEjecucion() : void
VistaDOR() : void
GUI
Modelo
VentanaMensaje
Utils
Executer
java.util
ArrayList
javax.swing
JButton
JPanel
JTextField
GUI
Vista
java.awt
BorderLayout
Color
Component
Container
Cursor
Font
GridLayout
LayoutManager
java.awt.event
ActionListener
KeyListener
java.lang
Object
String
javax.swing
BoxLayout
Icon
ImageIcon
JLabel
JScrollPane
JTextArea

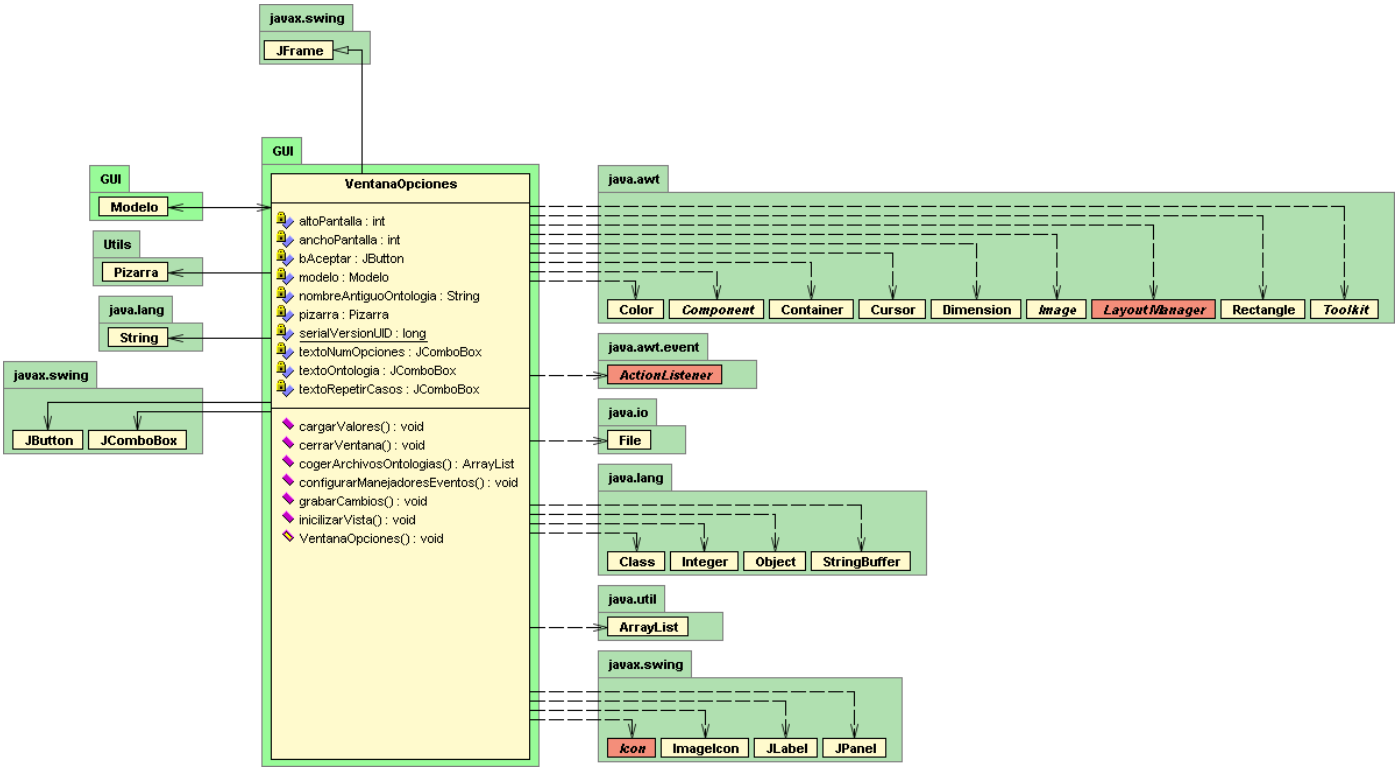

javax.swing
JInternalFrame
USR
VistaUSR
bAceptar : JButton
ejecutor : Executer
listadoPeticiones : JTextArea
serialVersionUID : long
textoIntroducido : JTextField
ventanaMensaje : VentanaMensaje
configurarManejadoresEventos() : void
consultarListado() : JTextArea
getInstance() : VentanaMensaje
inicializarPaneles() : void
secuenciaEjecucion() : void
VistaUSR() : void
GUI
Modelo
VentanaMensaje
Utils
Executer
javax.swing
JButton
JTextArea
JTextField
GUI
Vista
java.awt
BorderLayout
Color
Component
Container
Cursor
GridLayout
LayoutManager
java.awt.event
ActionListener
java.lang
String
javax.swing
Icon
ImageIcon
JLabel
JPanel
JScrollPane

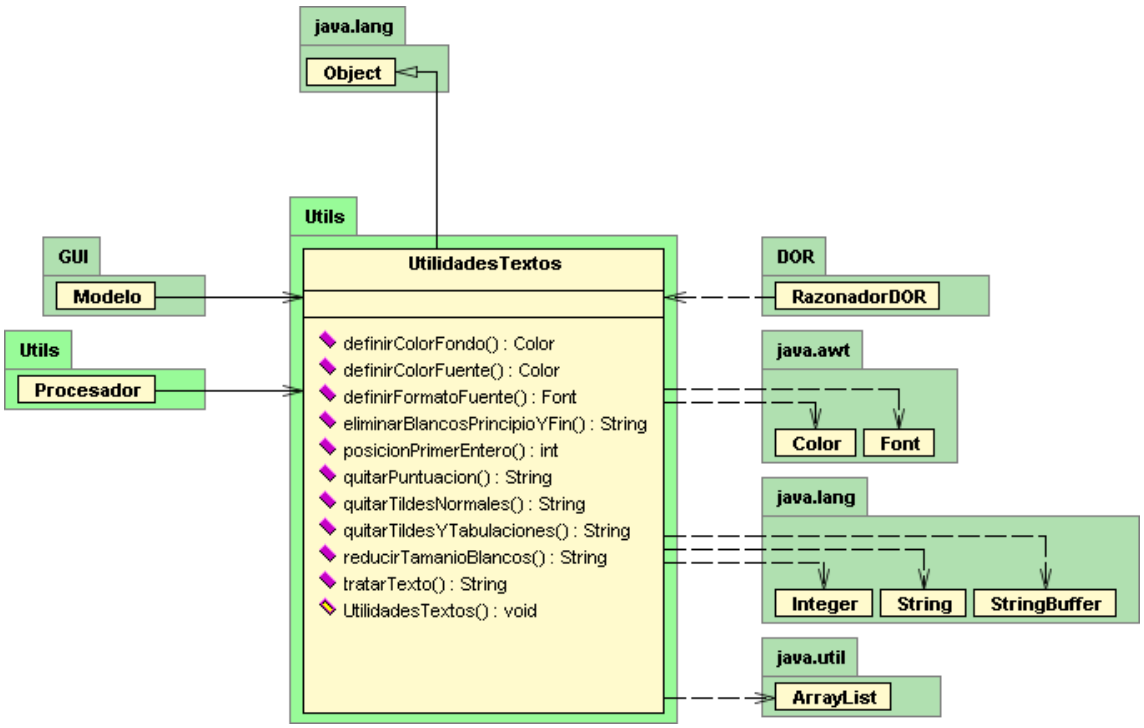

javax.swing
JInternalFrame
COM
VistaCOM
bAceptar : JButton
ejecutor : Executer
listadoPeticiones : JTextArea
serialVersionUID : long
textoIntroducido : JTextField
ventanaMensaje : VentanaMensaje
configurarManejadoresEventos() : void
disponerElementos() : void
getInstance() : VentanaMensaje
inicializarPaneles() : void
secuenciaEjecucion() : void
VistaCOM() : void
GUI
Modelo
VentanaMensaje
Utils
Executer
javax.swing
JButton
JTextArea
JTextField
GUI
Vista
java.awt
BorderLayout
Color
Component
Container
Cursor
GridLayout
LayoutManager
java.awt.event
ActionListener
java.lang
String
javax.swing
Icon
ImageIcon
JColorChooser
JLabel
JPanel
JScrollPane

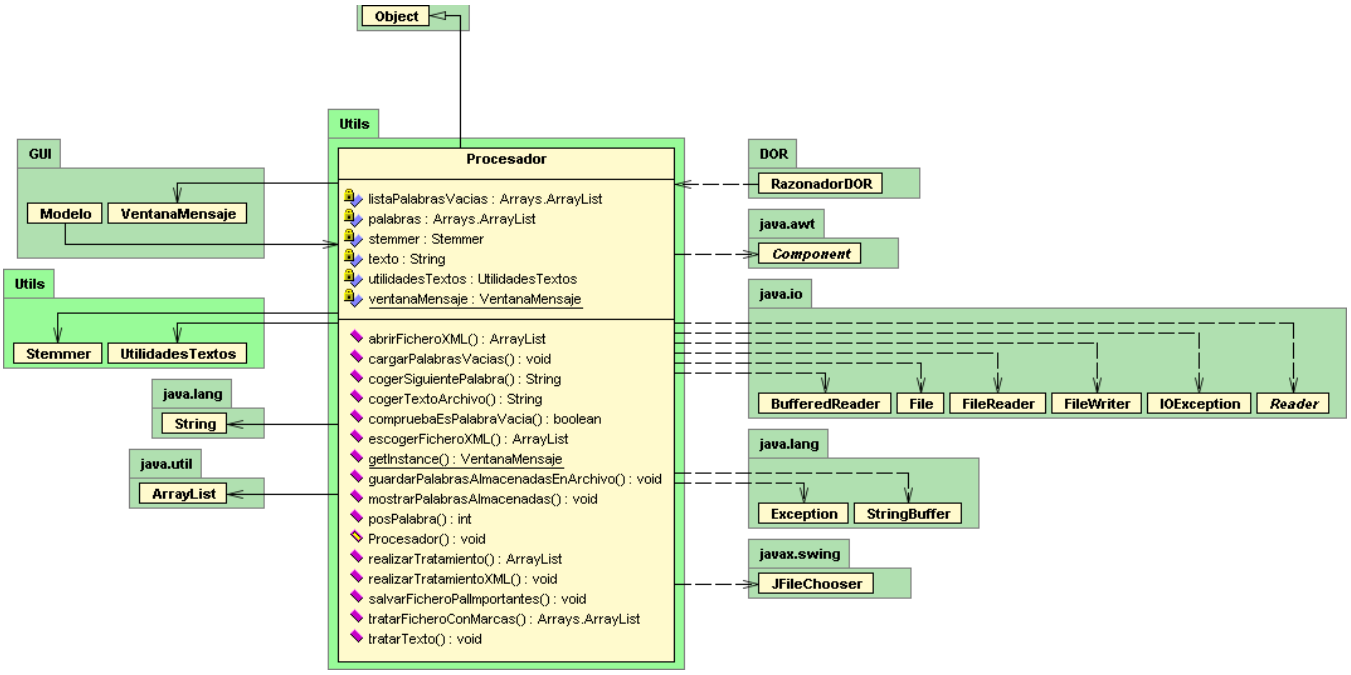

javax.swing
JInternalFrame
DYR
VistaDYR
bAceptar : JButton
ejecutor : Executer
listadoPeticiones : JTextArea
serialVersionUID : long
textoIntroducido : JTextField
ventanaMensaje : VentanaMensaje
configurarManejadoresEventos() : void
consultarListado() : JTextArea
disponerElementos() : void
getInstance() : VentanaMensaje
inicializarPaneles() : void
secuenciaEjecucion() : void
VistaDYR() : void
GUI
Modelo
VentanaMensaje
Utils
Executer
javax.swing
JButton
JTextArea
JTextField
GUI
Vista
java.awt
BorderLayout
Color
Component
Container
Cursor
GridLayout
LayoutManager
java.awt.event
ActionListener
java.lang
String
javax.swing
Icon
ImageIcon
JColorChooser
JLabel
JPanel
JScrollPane

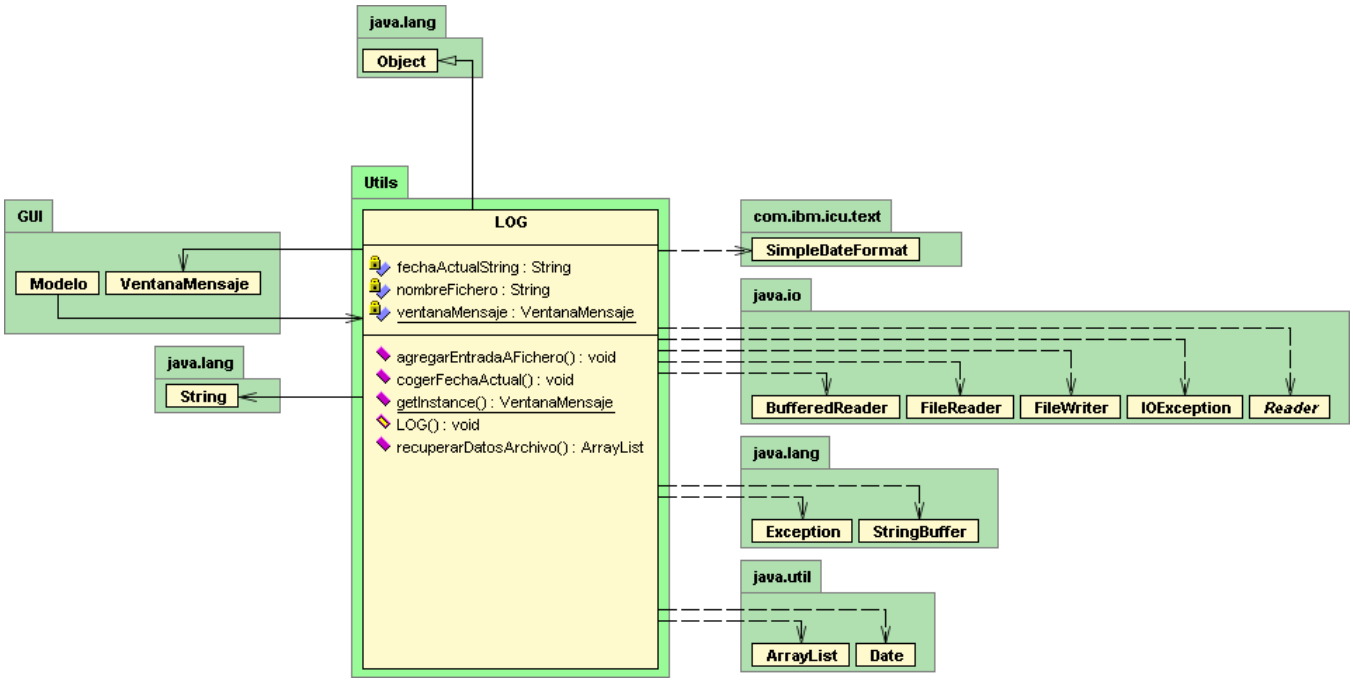

Object
DOR
EntradaCaso
activatingQuestions : ArrayList
arrayConceptosNoCoincidentes : ArrayList
arrayConceptosOK : ArrayList
importantWords : ArrayList
nombreIndividuo : String
numCoincidencias : double
objetoIndividuo : Individual
palabrasImportantesNombre : ArrayList
palabrasImportantesQ : ArrayList
serialVersionUID : long
tablaValores : Hashtable
agregaConceptoCoincidente() : void
agregaConceptoNoCoincidente() : void
agregarActivatingQ() : void
agregarPalabraImportante() : void
agregarPalabrasImportantesQ() : void
aumentaNumCoincidencias() : void
consultaNumCoincidencias() : int
consultaNumRestantes() : int
consultaPalabrasImportantes() : ArrayList
consultaPalabrasImportantesN() : ArrayList
consultaPalabrasImportantesQ() : ArrayList
consultarNombreIndividuo() : String
consultarObjetoIndividuo() : Individual
consultaSiTieneOKConcepto() : boolean
consultaTablaValores() : Hashtable
consultaTamanioArrayConceptosOK() : int
EntradaCaso() : void
esMismoNombreIndividuo() : boolean
hayConceptosNoCoincidentes() : boolean
ponNombreIndividuo() : void
ponNumCoincidencias() : void
ponObjetoIndividuo() : void
ponPalabrasImportantes() : void
ponPalabrasImportantesNombre() : void
ponTablaValores() : void
reseteaNumCoincidencias() : void
tieneActivatingQ() : boolean
tienePalabraImportante() : boolean
tienePalEnNombre() : boolean
com.hp.hpl.jena.ontology
Individual
java.lang
String
java.util
ArrayList
Hashtable
DOR
RazonadorDOR
GUI
Modelo
DOR
PalabraImportante
casosPalabra : Hashtable
serialVersionUID : long
aumentarFrecuenciaEnCaso() : void
calculaPeso() : double
PalabraImportante() : PalabraImportante
palabra : String

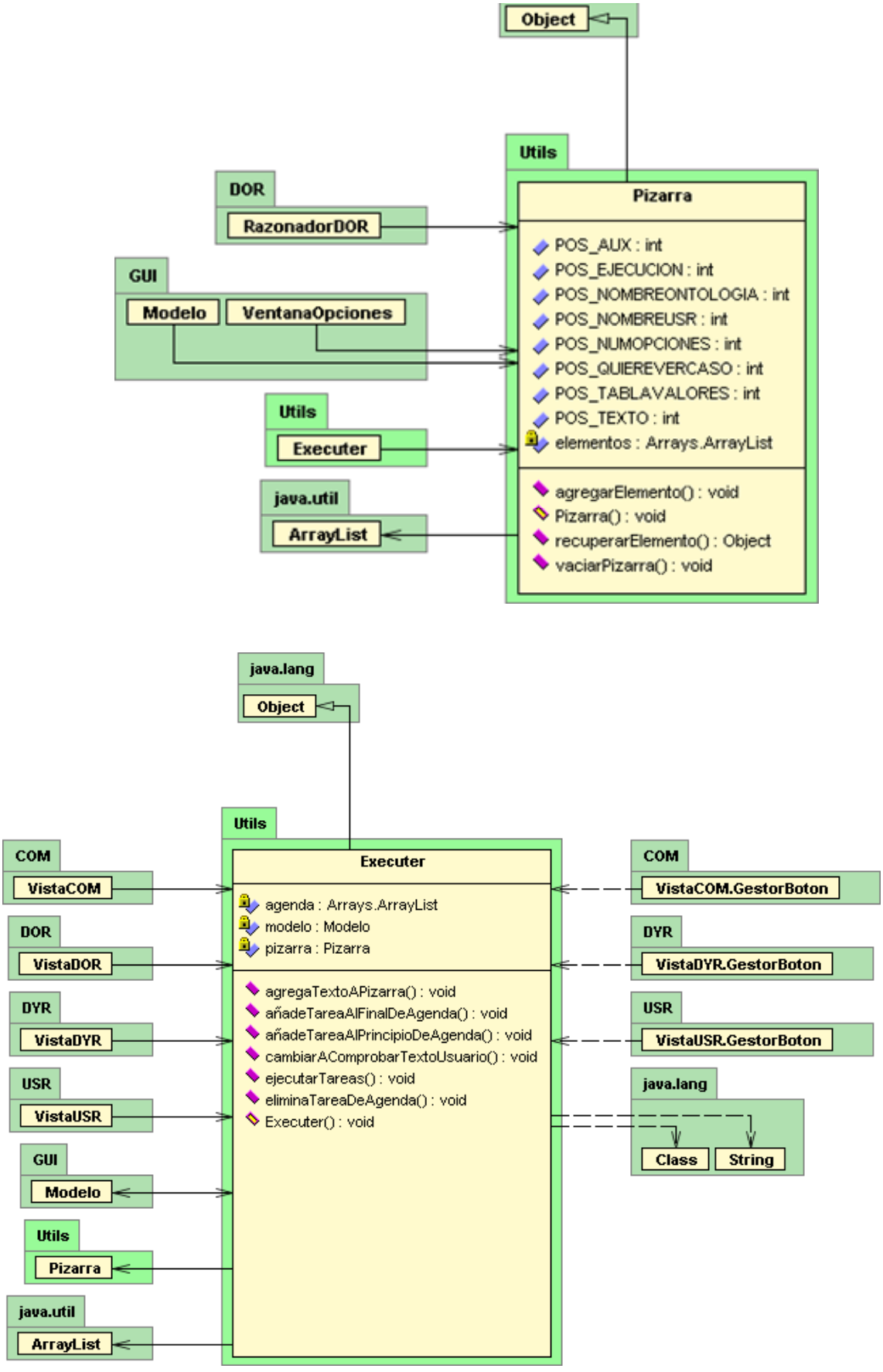

java.lang
Object
DOR
RazonadorDOR
GUI
Modelo
VentanaMensaje
ConectorJavaOwl
ConectorJavaOwl
Utils
Pizarra
java.lang
String
java.util
ArrayList
arrayCasosAComprobar : Arrays.ArrayList
arrayEntradaCasos : Arrays.ArrayList
arrayNombreCasos : Arrays.ArrayList
conector : ConectorJavaOwl
nombreOnto : String
numOpciones : int
palabrasImportantesGlobales : Arrays.ArrayList
pizarra : Pizarra
ultimasOpciones : Arrays.ArrayList
ventanaMensaje : VentanaMensaje
addPalabra() : void
buscarCasosSegunImportantes() : Arrays.ArrayList
cargarObjetoOnto() : boolean
cargarPruebaPizarra() : void
casosConMismoIdentificador() : ArrayList
cogerCasoBase() : Individual
comprobarDatosPizarraParaParejaValores() : void
compruebaActivatingQuestions() : ArrayList
consultaDescripcionGrafica() : ArrayList
consultaDescripcionGrafica() : ArrayList
consultaDescripcionTextual() : String
consultaDescripcionTextual() : String
consultaIntentionalQ() : ArrayList
consultaIntentionalQ() : ArrayList
consultaOutputSlot() : Individual
consultaParejasValores() : Hashtable
consultaPreguntasARealizar() : ArrayList
consultarPrimerCaso() : EntradaCaso
consultarUltimasOpciones() : ArrayList
consultaSuggestion() : String
consultaSuggestion() : String
consultaSuggestions() : ArrayList
establecerNumOpciones() : void
existeOtroCasoEnUltimasOpciones() : boolean
extraeCasosOntologia() : void
extraeCasosParaComprobacion() : void
extraeOpciones() : Arrays.ArrayList
generaNumAleatorio() : int
getInstance() : VentanaMensaje
guardarObjetoOnto() : void
modificarOntologia() : void
obtenIndividuo() : Individual
ordenaCasosConceptosValores() : void
ordenaCasosConCoincidencias() : Arrays.ArrayList
posPalabra() : int
primerCasoTieneTodosValores() : boolean
procesaParejaValoresCasos() : void
quitarCasoDeUltimasOpciones() : void
razona() : Arrays.ArrayList
RazonadorDOR() : void
razonaInicio() : String
recogePalabrasEnCasos() : void
recuperarDatosDeObjetoOntologia() : void
renovarUltimasOpciones() : void
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
AccesoPropiedades
DOR
EntradaCaso
PalabraImportante
Utils
Procesador
UtilidadesTextos
com.hp.hpl.jena.ontology
Individual
java.awt
Component
java.io
FileInputStream
FileOutputStream
InputStream
IOException
ObjectInputStream
ObjectOutputStream
OutputStream
PrintStream
java.lang
Class
Integer
Math
StringBuffer
System
java.util
Enumeration
Hashtable
javax.swing
JFrame
JOptionPane

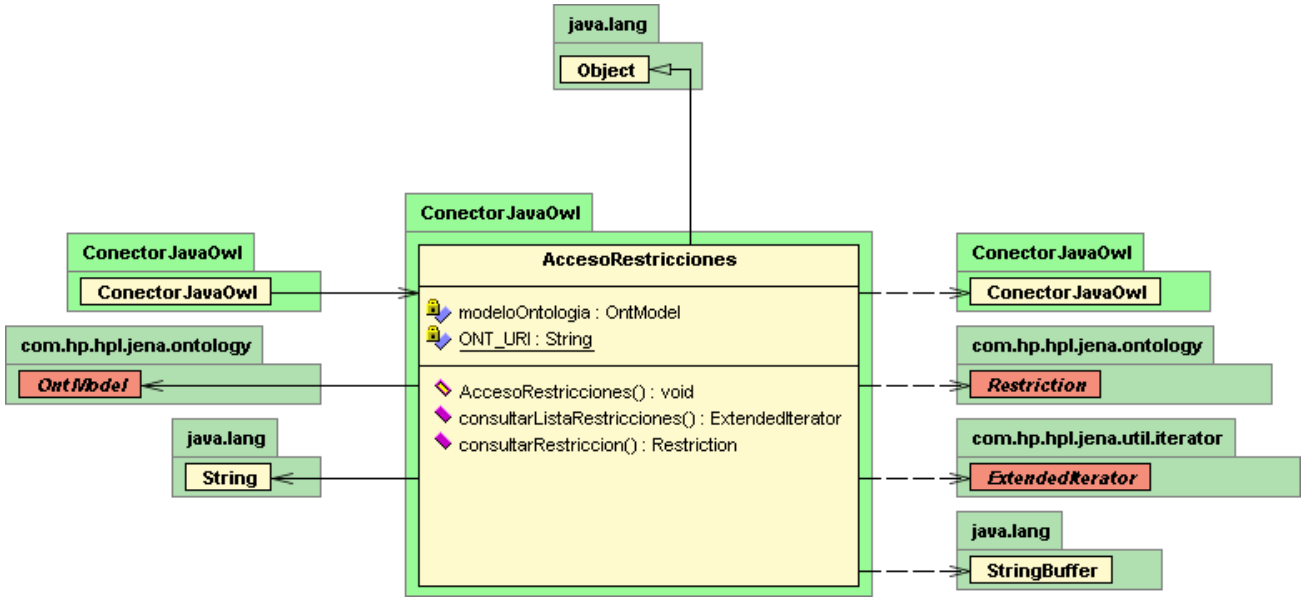

javax.swing
JFrame
GUI
VentanaOpciones
altoPantalla : int
anchoPantalla : int
bAceptar : JButton
modelo : Modelo
nombreAntiguoOntologia : String
pizarra : Pizarra
serialVersionUID : long
textoNumOpciones : JComboBox
textoOntologia : JComboBox
textoRepetirCasos : JComboBox
cargarValores() : void
cerrarVentana() : void
cogerArchivosOntologias() : ArrayList
configurarManejadoresEventos() : void
grabarCambios() : void
inicilizarVista() : void
VentanaOpciones() : void
GUI
Modelo
Utils
Pizarra
java.lang
String
javax.swing
JButton
JComboBox
java.awt
Color
Component
Container
Cursor
Dimension
Image
LayoutManager
Rectangle
Toolkit
java.awt.event
ActionListener
java.io
File
java.lang
Class
Integer
Object
StringBuffer
java.util
ArrayList
javax.swing
Icon
ImageIcon
JLabel
JPanel

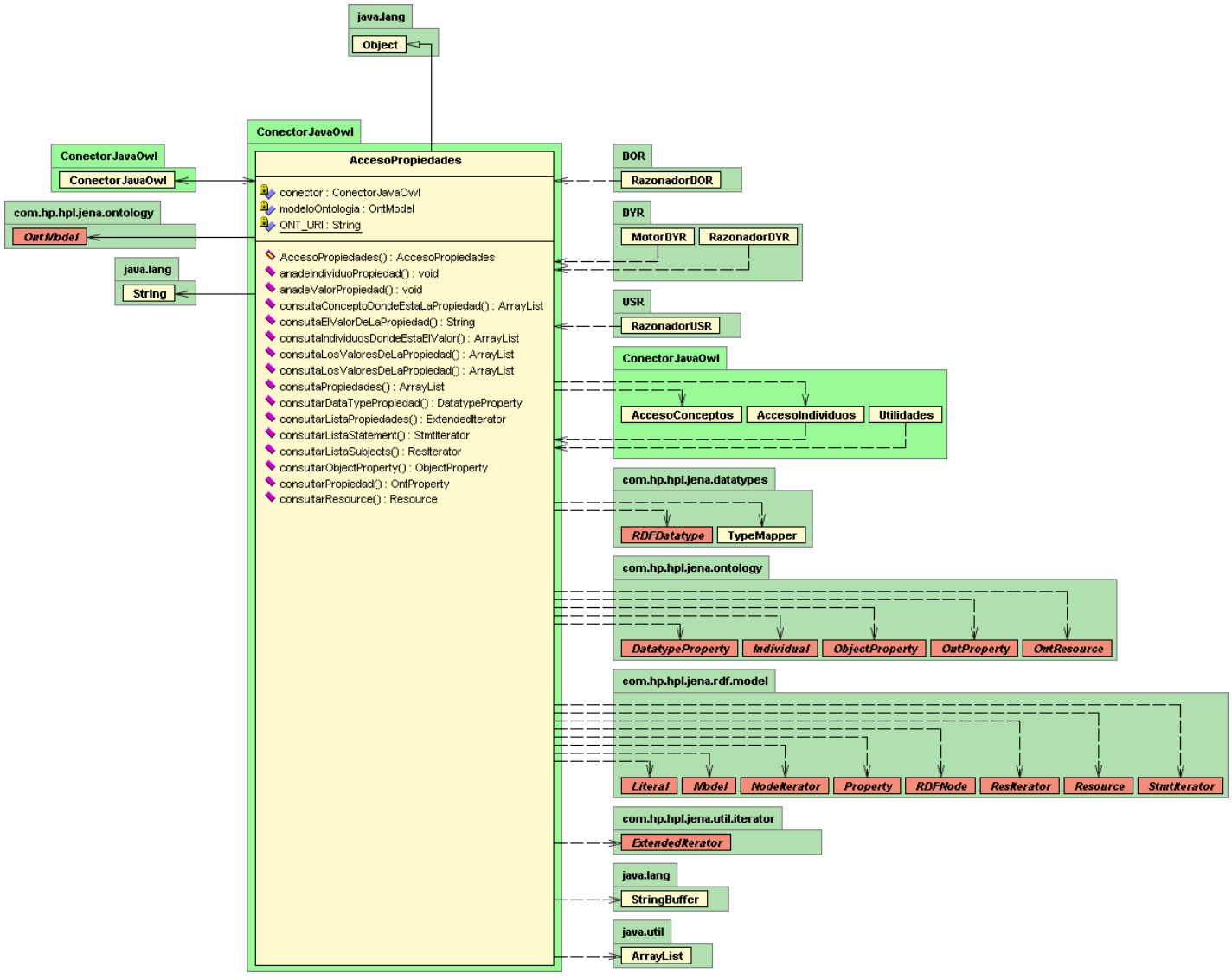

java.lang
Object
Utils
UtilidadesTextos
definirColorFondo() : Color
definirColorFuente() : Color
definirFormatoFuente() : Font
eliminarBlancosPrincipioYFin() : String
posicionPrimerEntero() : int
quitarPuntuacion() : String
quitarTildesNormales() : String
quitarTildesYTabulaciones() : String
reducirTamanioBlancos() : String
tratarTexto() : String
UtilidadesTextos() : void
GUI
Modelo
Utils
Procesador
DOR
RazonadorDOR
java.awt
Color
Font
java.lang
Integer
String
StringBuffer
java.util
ArrayList

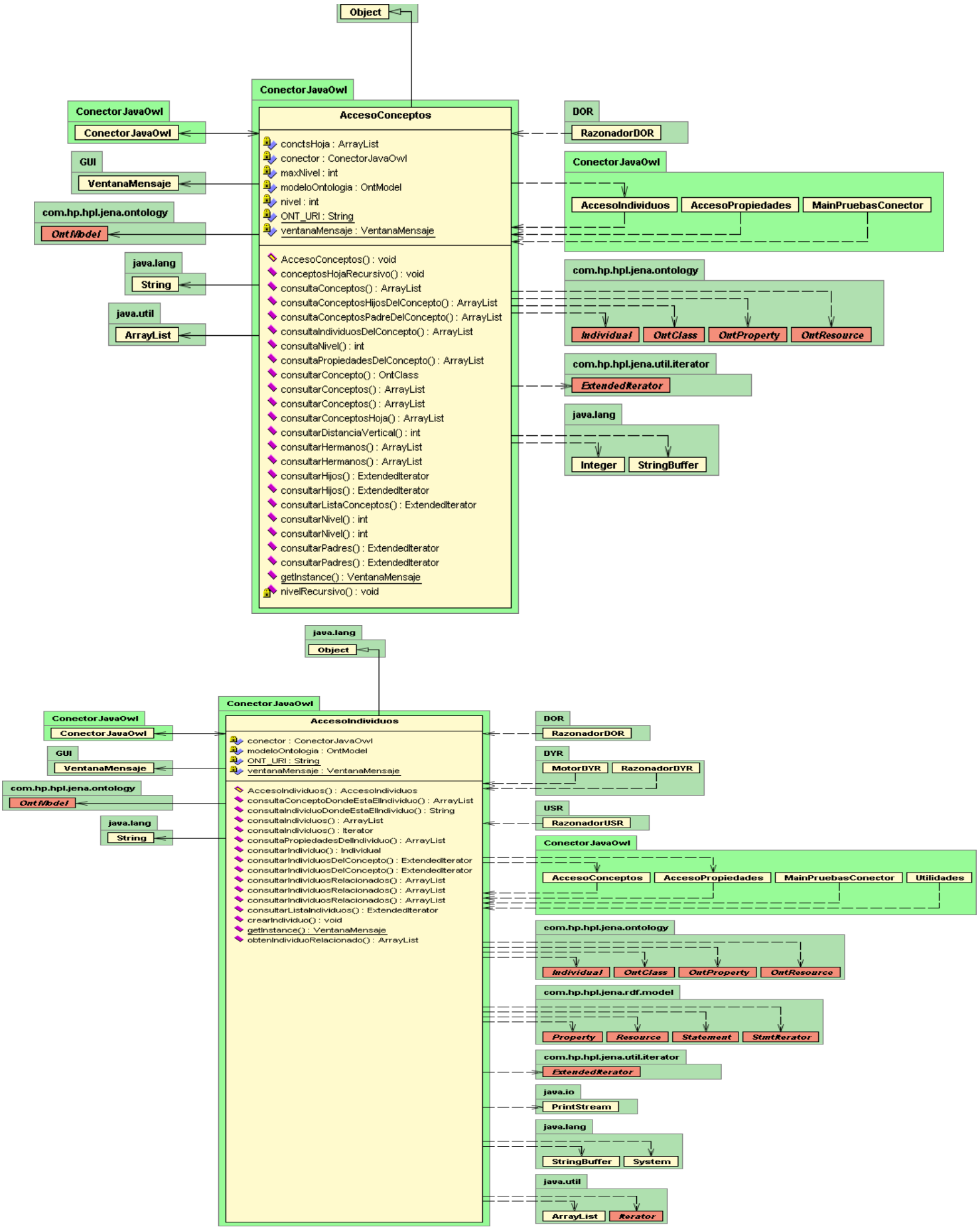

Object
Utils
Procesador
listaPalabrasVacias : Arrays.ArrayList
palabras : Arrays.ArrayList
stemmer : Stemmer
texto : String
utilidadesTextos : UtilidadesTextos
ventanaMensaje : VentanaMensaje
abrirFicheroXML() : ArrayList
cargarPalabrasVacias() : void
cogerSiguientePalabra() : String
cogerTextoArchivo() : String
compruebaEsPalabraVacia() : boolean
escogerFicheroXML() : ArrayList
getInstance() : VentanaMensaje
guardarPalabrasAlmacenadasEnArchivo() : void
mostrarPalabrasAlmacenadas() : void
posPalabra() : int
Procesador() : void
realizarTratamiento() : ArrayList
realizarTratamientoXML() : void
salvarFicheroPalImportantes() : void
tratarFicheroConMarcas() : Arrays.ArrayList
tratarTexto() : void
GUI
Modelo
VentanaMensaje
Utils
Stemmer
UtilidadesTextos
java.lang
String
java.util
ArrayList
DOR
RazonadorDOR
java.awt
Component
java.io
BufferedReader
File
FileReader
FileWriter
IOException
Reader
java.lang
Exception
StringBuffer
javax.swing
JFileChooser

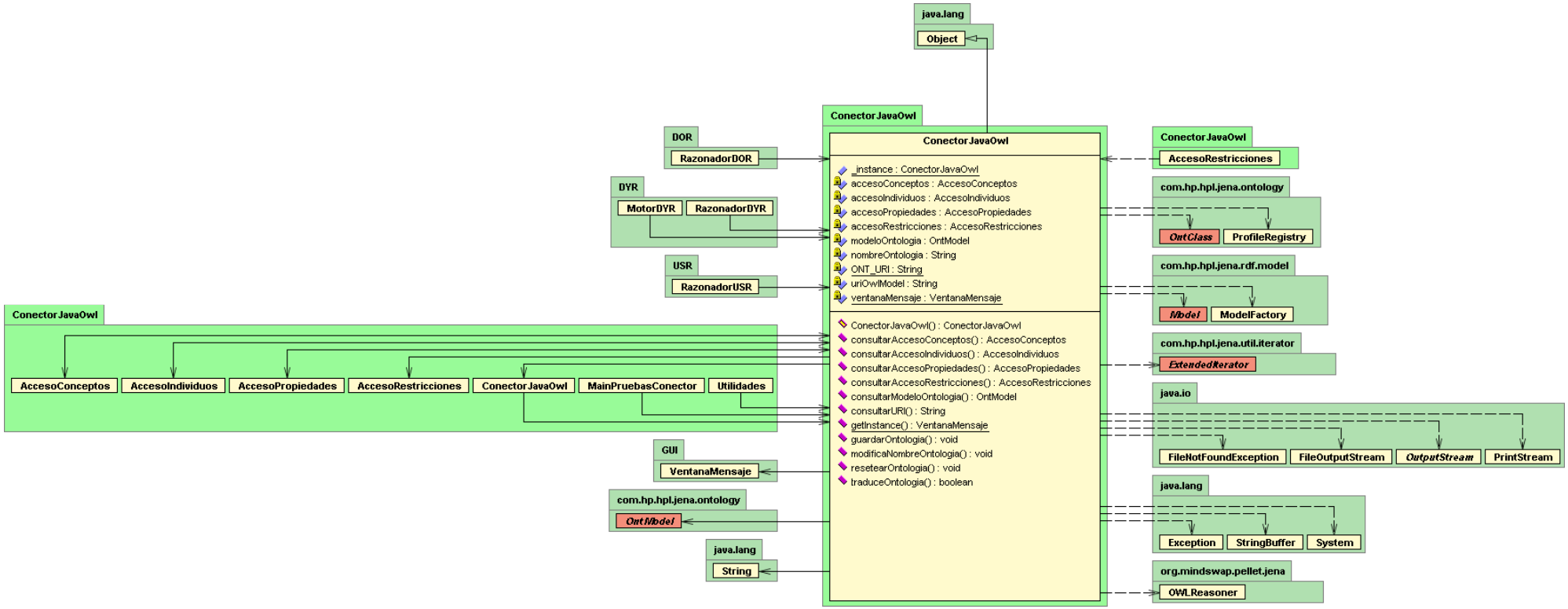

java.lang
Object
Utils
LOG
fechaActualString : String
nombreFichero : String
ventanaMensaje : VentanaMensaje
agregarEntradaAFichero() : void
cogerFechaActual() : void
getInstance() : VentanaMensaje
LOG() : void
recuperarDatosArchivo() : ArrayList
GUI
Modelo
VentanaMensaje
java.lang
String
com.ibm.icu.text
SimpleDateFormat
java.io
BufferedReader
FileReader
FileWriter
IOException
Reader
java.lang
Exception
StringBuffer
java.util
ArrayList
Date

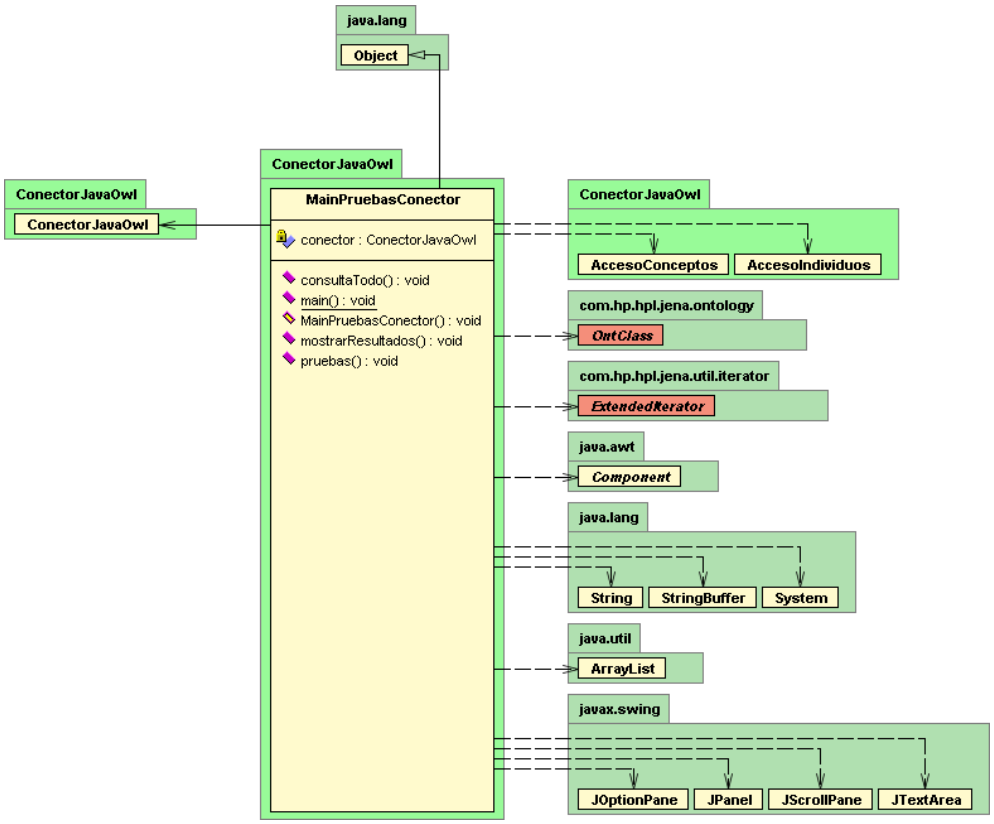

Object
Utils
Pizarra
DOR
RazonadorDOR
GUI
Modelo
VentanaOpciones
Utils
Executer
java.util
ArrayList
POS_AUX : int
POS_EJECUCION : int
POS_NOMBREONTOLOGIA : int
POS_NOMBREUSR : int
POS_NUMOPCIONES : int
POS_QUIEREVERCASO : int
POS_TABLAVALORES : int
POS_TEXTO : int
elementos : Arrays.ArrayList
agregarElemento() : void
Pizarra() : void
recuperarElemento() : Object
vaciarPizarra() : void
java.lang
Object
Utils
Executer
COM
VistaCOM
DOR
VistaDOR
DYR
VistaDYR
USR
VistaUSR
GUI
Modelo
Utils
Pizarra
java.util
ArrayList
agenda : Arrays.ArrayList
modelo : Modelo
pizarra : Pizarra
agregaTextoAPizarra() : void
añadeTareaAlFinalDeAgenda() : void
añadeTareaAlPrincipioDeAgenda() : void
cambiarAComprobarTextoUsuario() : void
ejecutarTareas() : void
eliminaTareaDeAgenda() : void
Executer() : void
COM
VistaCOM.GestorBoton
DYR
VistaDYR.GestorBoton
USR
VistaUSR.GestorBoton
java.lang
Class
String

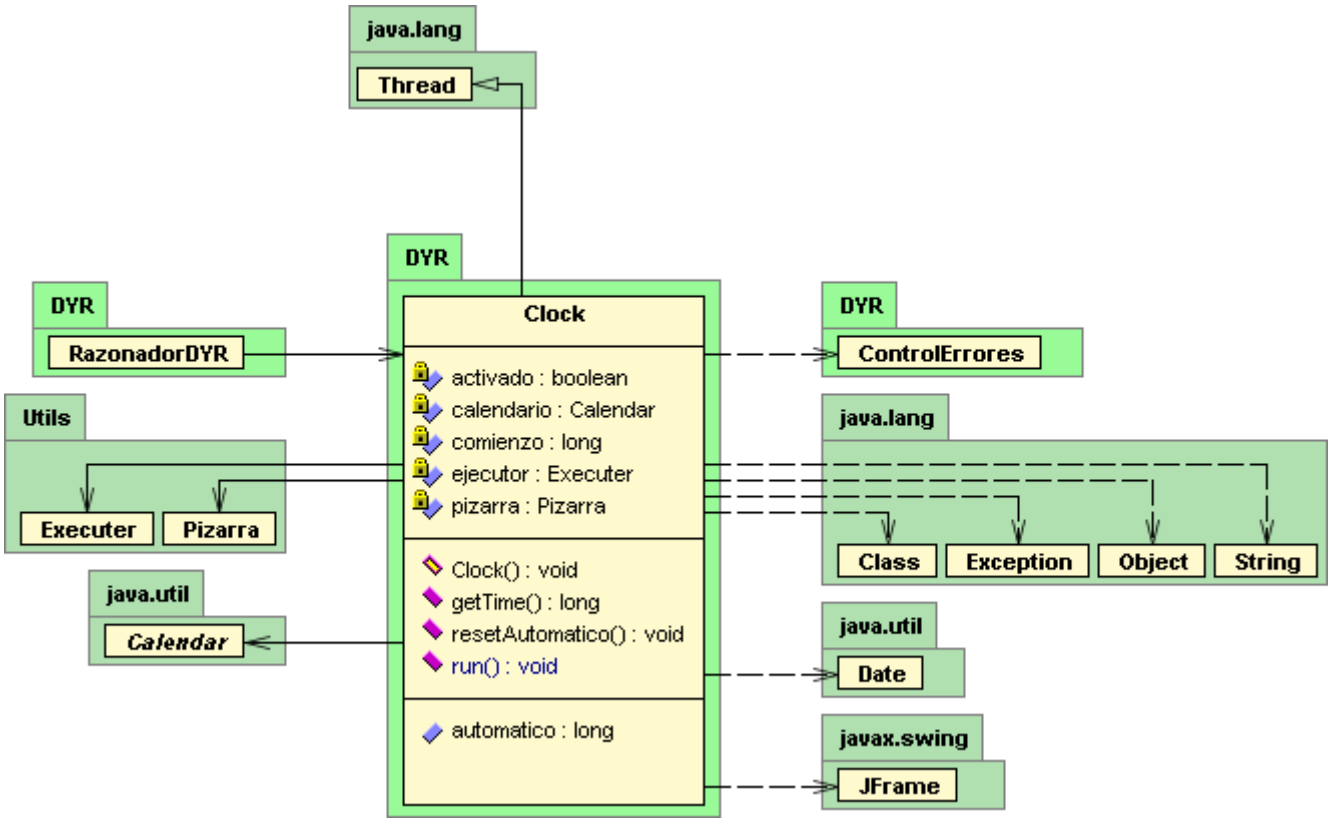

java.lang
Object
ConectorJavaOwl
AccesoRestricciones
modeloOntologia : OntModel
ONT_URI : String
AccesoRestricciones() : void
consultarListaRestricciones() : ExtendedIterator
consultarRestriccion() : Restriction
ConectorJavaOwl
ConectorJavaOwl
com.hp.hpl.jena.ontology
OntModel
java.lang
String
ConectorJavaOwl
ConectorJavaOwl
com.hp.hpl.jena.ontology
Restriction
com.hp.hpl.jena.util.iterator
ExtendedIterator
java.lang
StringBuffer

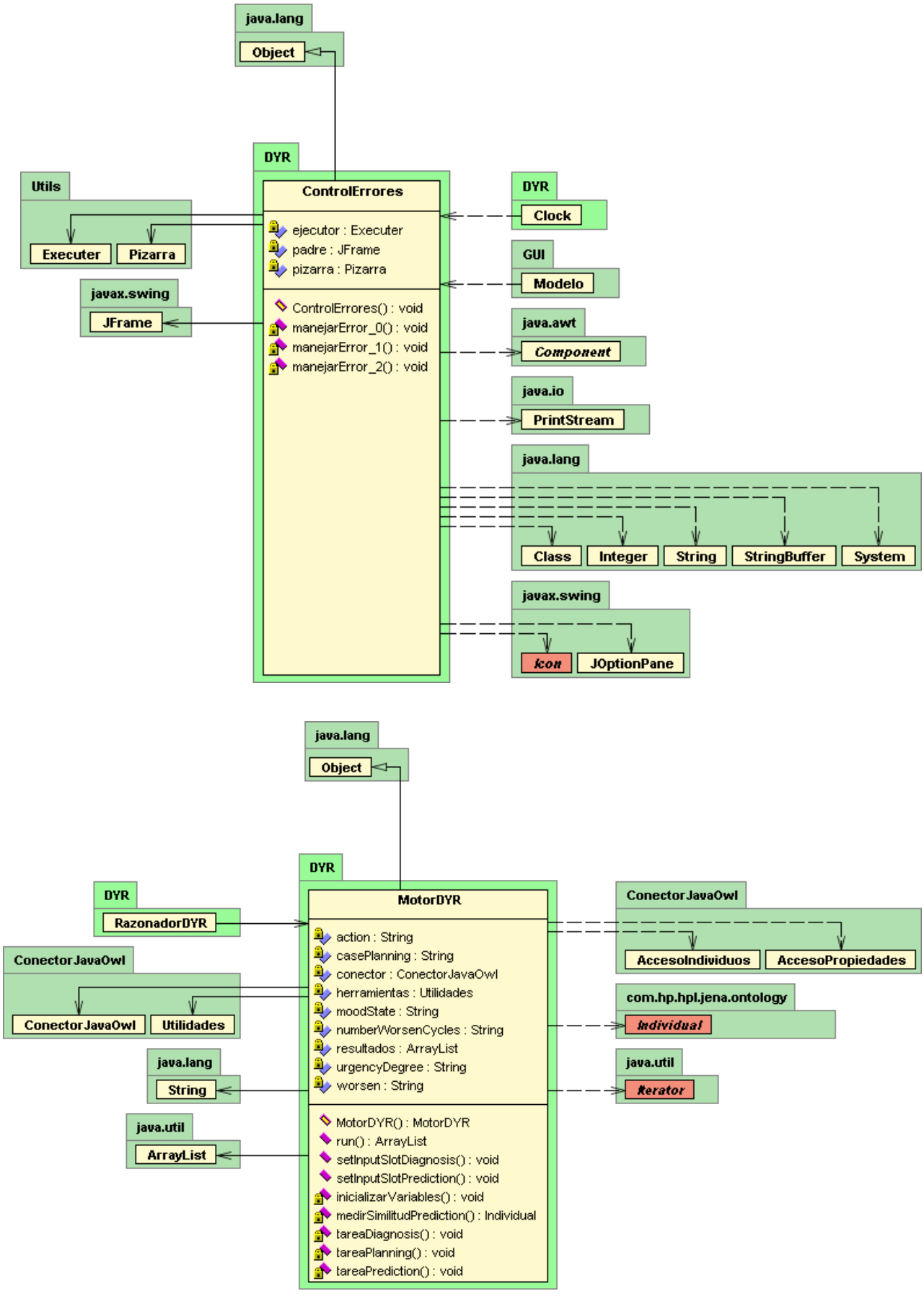

java.lang
Object
ConectorJavaOwl
AccesoPropiedades
conector : ConectorJavaOwl
modeloOntologia : OntModel
ONT_URI : String
AccesoPropiedades() : AccesoPropiedades
anadeIndividuoPropiedad() : void
anadeValorPropiedad() : void
consultaConceptoDondeEstaLaPropiedad() : ArrayList
consultaElValorDeLaPropiedad() : String
consultaIndividuosDondeEstaElValor() : ArrayList
consultaLosValoresDeLaPropiedad() : ArrayList
consultaLosValoresDeLaPropiedad() : ArrayList
consultaPropiedades() : ArrayList
consultarDataTypePropiedad() : DatatypeProperty
consultarListaPropiedades() : ExtendedIterator
consultarListaStatement() : StmtIterator
consultarListaSubjects() : ResIterator
consultarObjectProperty() : ObjectProperty
consultarPropiedad() : OntProperty
consultarResource() : Resource
ConectorJavaOwl
ConectorJavaOwl
com.hp.hpl.jena.ontology
OntModel
java.lang
String
DOR
RazonadorDOR
DYR
MotorDYR
RazonadorDYR
USR
RazonadorUSR
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
Utilidades
com.hp.hpl.jena.datatypes
RDFDatatype
TypeMapper
com.hp.hpl.jena.ontology
DatatypeProperty
Individual
ObjectProperty
OntProperty
OntResource
com.hp.hpl.jena.rdf.model
Literal
Model
NodeIterator
Property
RDFNode
ResIterator
Resource
StmtIterator
com.hp.hpl.jena.util.iterator
ExtendedIterator
java.lang
StringBuffer
java.util
ArrayList

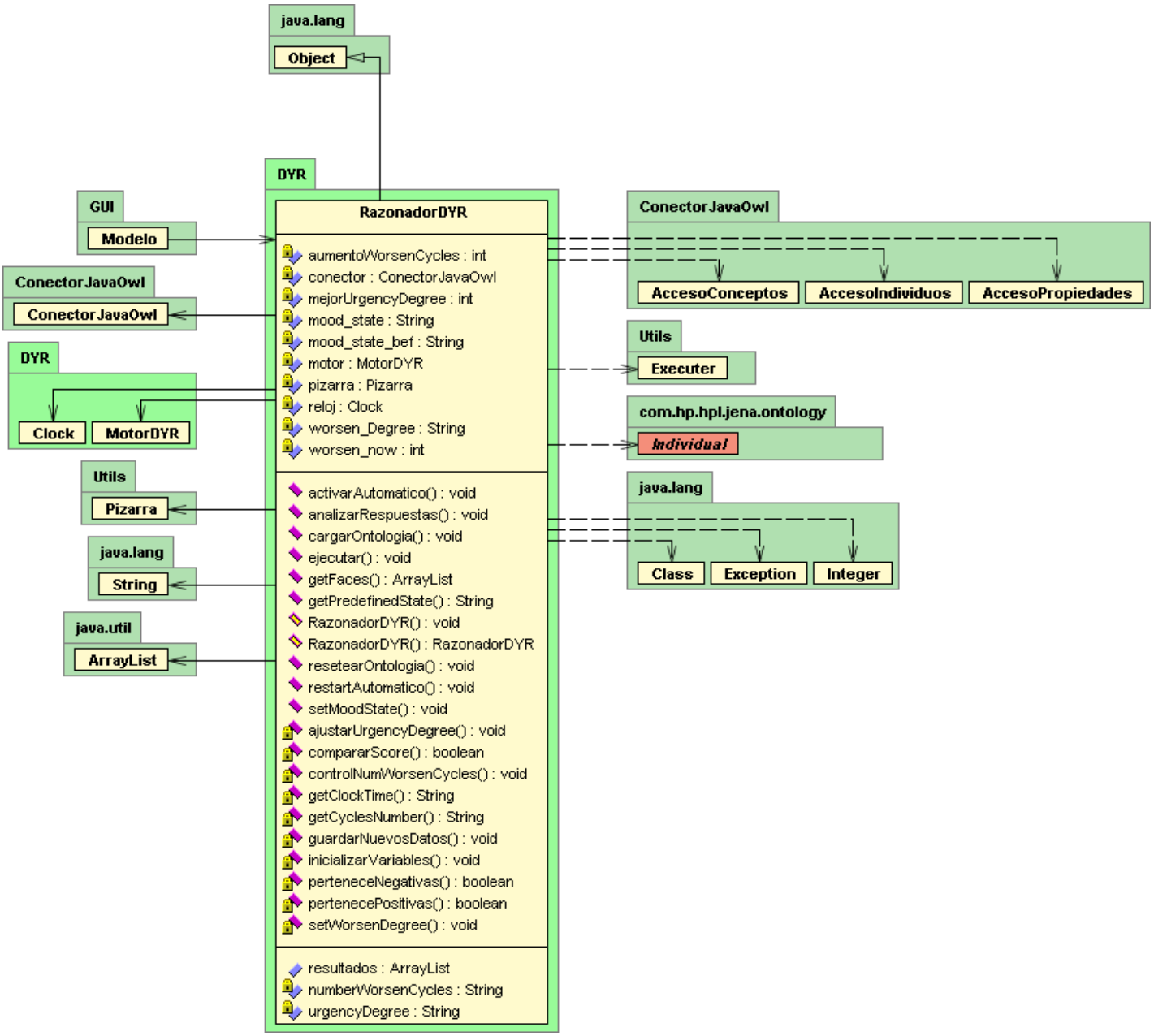

Object
ConectorJavaOwl
AccesoConceptos
conctsHoja : ArrayList
conector : ConectorJavaOwl
maxNivel : int
modeloOntologia : OntModel
nivel : int
ONT_URI : String
ventanaMensaje : VentanaMensaje
AccesoConceptos() : void
conceptosHojaRecursivo() : void
consultaConceptos() : ArrayList
consultaConceptosHijosDelConcepto() : ArrayList
consultaConceptosPadreDelConcepto() : ArrayList
consultaIndividuosDelConcepto() : ArrayList
consultaNivel() : int
consultaPropiedadesDelConcepto() : ArrayList
consultarConcepto() : OntClass
consultarConceptos() : ArrayList
consultarConceptos() : ArrayList
consultarConceptosHoja() : ArrayList
consultarDistanciaVertical() : int
consultarHermanos() : ArrayList
consultarHermanos() : ArrayList
consultarHijos() : ExtendedIterator
consultarHijos() : ExtendedIterator
consultarListaConceptos() : ExtendedIterator
consultarNivel() : int
consultarNivel() : int
consultarPadres() : ExtendedIterator
consultarPadres() : ExtendedIterator
getInstance() : VentanaMensaje
nivelRecursivo() : void
ConectorJavaOwl
ConectorJavaOwl
GUI
VentanaMensaje
com.hp.hpl.jena.ontology
OntModel
java.lang
String
java.util
ArrayList
DOR
RazonadorDOR
ConectorJavaOwl
AccesoIndividuos
AccesoPropiedades
MainPruebasConector
com.hp.hpl.jena.ontology
Individual
OntClass
OntProperty
OntResource
com.hp.hpl.jena.util.iterator
ExtendedIterator
java.lang
Integer
StringBuffer
java.lang
Object
ConectorJavaOwl
AccesoIndividuos
conector : ConectorJavaOwl
modeloOntologia : OntModel
ONT_URI : String
ventanaMensaje : VentanaMensaje
AccesoIndividuos() : AccesoIndividuos
consultaConceptoDondeEstaElIndividuo() : ArrayList
consultaIndividuoDondeEstaElIndividuo() : String
consultaIndividuos() : ArrayList
consultaIndividuos() : Iterator
consultaPropiedadesDelIndividuo() : ArrayList
consultarIndividuo() : Individual
consultarIndividuosDelConcepto() : ExtendedIterator
consultarIndividuosDelConcepto() : ExtendedIterator
consultarIndividuosRelacionados() : ArrayList
consultarIndividuosRelacionados() : ArrayList
consultarIndividuosRelacionados() : ArrayList
consultarListaIndividuos() : ExtendedIterator
crearIndividuo() : void
getInstance() : VentanaMensaje
obtenIndividuoRelacionado() : ArrayList
ConectorJavaOwl
ConectorJavaOwl
GUI
VentanaMensaje
com.hp.hpl.jena.ontology
OntModel
java.lang
String
DOR
RazonadorDOR
DYR
MotorDYR
RazonadorDYR
USR
RazonadorUSR
ConectorJavaOwl
AccesoConceptos
AccesoPropiedades
MainPruebasConector
Utilidades
com.hp.hpl.jena.ontology
Individual
OntClass
OntProperty
OntResource
com.hp.hpl.jena.rdf.model
Property
Resource
Statement
StmtIterator
com.hp.hpl.jena.util.iterator
ExtendedIterator
java.io
PrintStream
java.lang
StringBuffer
System
java.util
ArrayList
Iterator

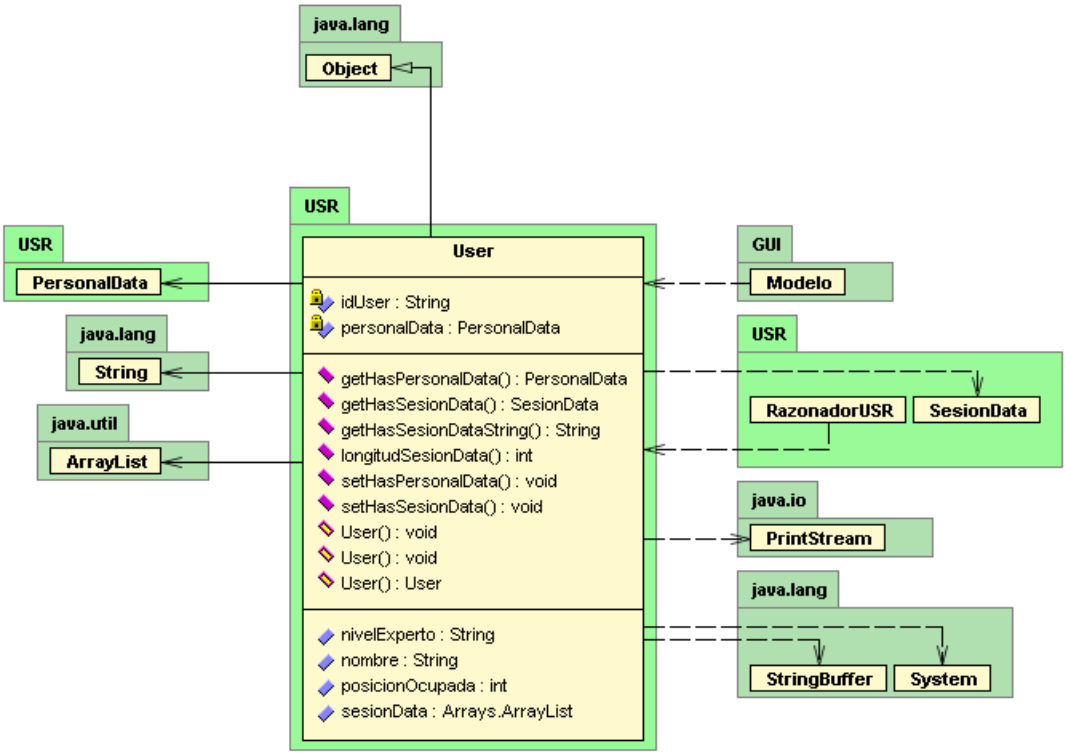

java.lang
Object
ConectorJavaOwl
ConectorJavaOwl
DOR
RazonadorDOR
DYR
MotorDYR
RazonadorDYR
USR
RazonadorUSR
_instance : ConectorJavaOwl
accesoConceptos : AccesoConceptos
accesoIndividuos : AccesoIndividuos
accesoPropiedades : AccesoPropiedades
accesoRestricciones : AccesoRestricciones
modeloOntologia : OntModel
nombreOntologia : String
ONT_URI : String
uriOwlModel : String
ventanaMensaje : VentanaMensaje
ConectorJavaOwl() : ConectorJavaOwl
consultarAccesoConceptos() : AccesoConceptos
consultarAccesoIndividuos() : AccesoIndividuos
consultarAccesoPropiedades() : AccesoPropiedades
consultarAccesoRestricciones() : AccesoRestricciones
consultarModeloOntologia() : OntModel
consultarURI() : String
getInstance() : VentanaMensaje
guardarOntologia() : void
modificaNombreOntologia() : void
resetearOntologia() : void
traduceOntologia() : boolean
ConectorJavaOwl
AccesoRestricciones
com.hp.hpl.jena.ontology
OntClass
ProfileRegistry
com.hp.hpl.jena.rdf.model
Model
ModelFactory
com.hp.hpl.jena.util.iterator
ExtendedIterator
java.io
FileNotFoundException
FileOutputStream
OutputStream
PrintStream
java.lang
Exception
StringBuffer
System
org.mindswap.pellet.jena
OWLReasoner
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
AccesoPropiedades
AccesoRestricciones
ConectorJavaOwl
MainPruebasConector
Utilidades
GUI
VentanaMensaje
com.hp.hpl.jena.ontology
OntModel
java.lang
String

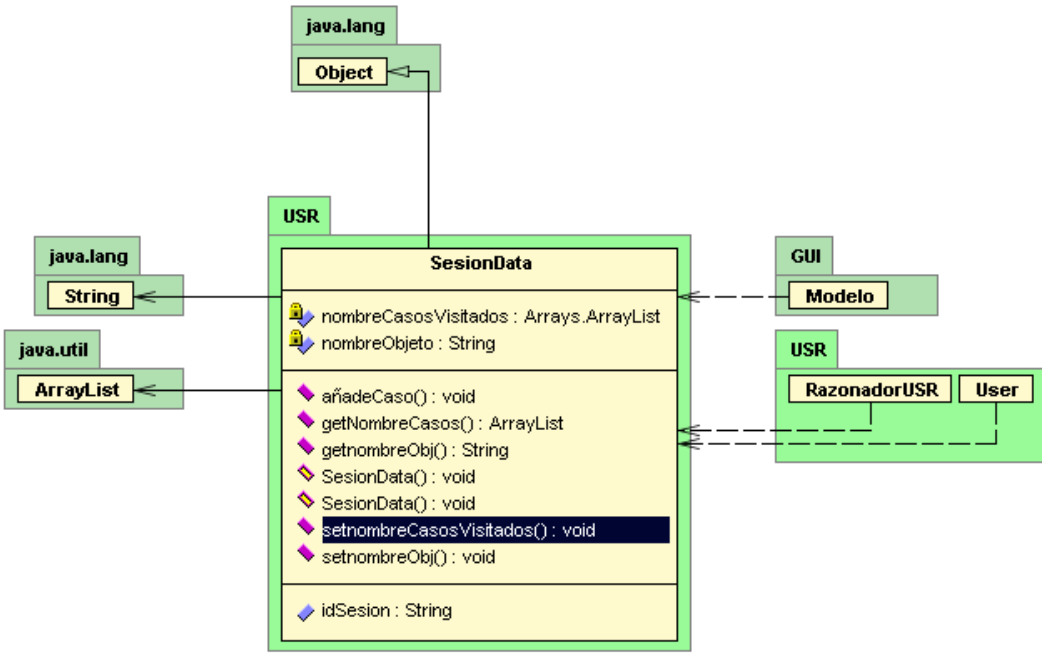

java.lang
Object
ConectorJavaOwl
ConectorJavaOwl
ConectorJavaOwl
ConectorJavaOwl
MainPruebasConector
conector : ConectorJavaOwl
consultaTodo() : void
main() : void
MainPruebasConector() : void
mostrarResultados() : void
pruebas() : void
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
com.hp.hpl.jena.ontology
OntClass
com.hp.hpl.jena.util.iterator
ExtendedIterator
java.awt
Component
java.lang
String
StringBuffer
System
java.util
ArrayList
javax.swing
JOptionPane
JPanel
JScrollPane
JTextArea

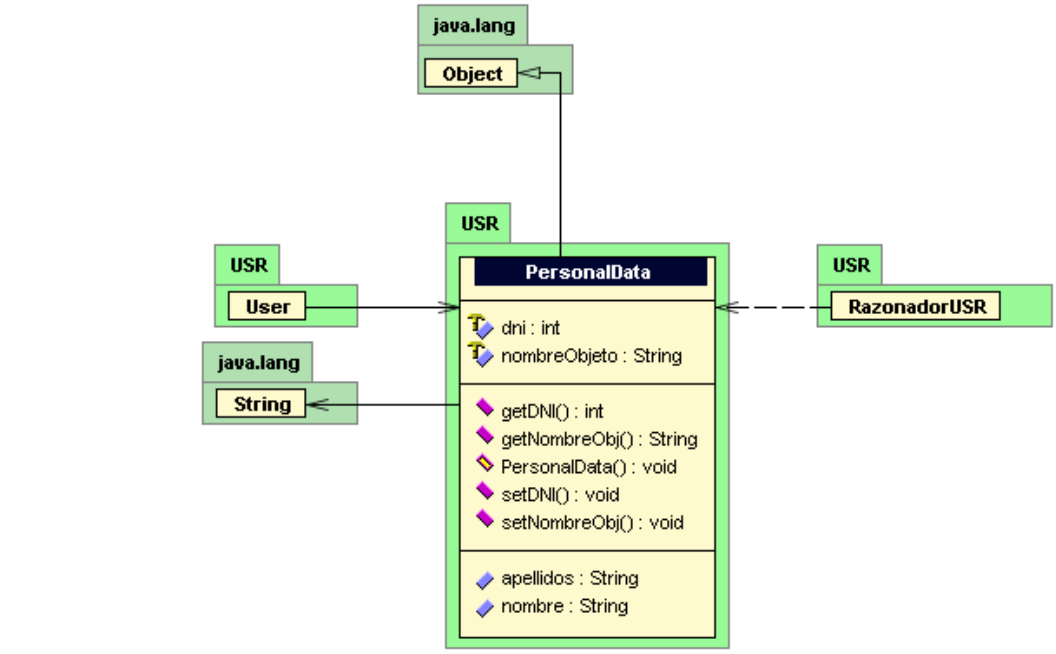

java.lang
Thread
DYR
DYR
RazonadorDYR
Clock
DYR
ControlErrores
Utils
Executer
Pizarra
activado : boolean
calendario : Calendar
comienzo : long
ejecutor : Executer
pizarra : Pizarra
java.lang
Class
Exception
Object
String
java.util
Calendar
Clock() : void
getTime() : long
resetAutomatico() : void
run() : void
java.util
Date
automatico : long
javax.swing
JFrame

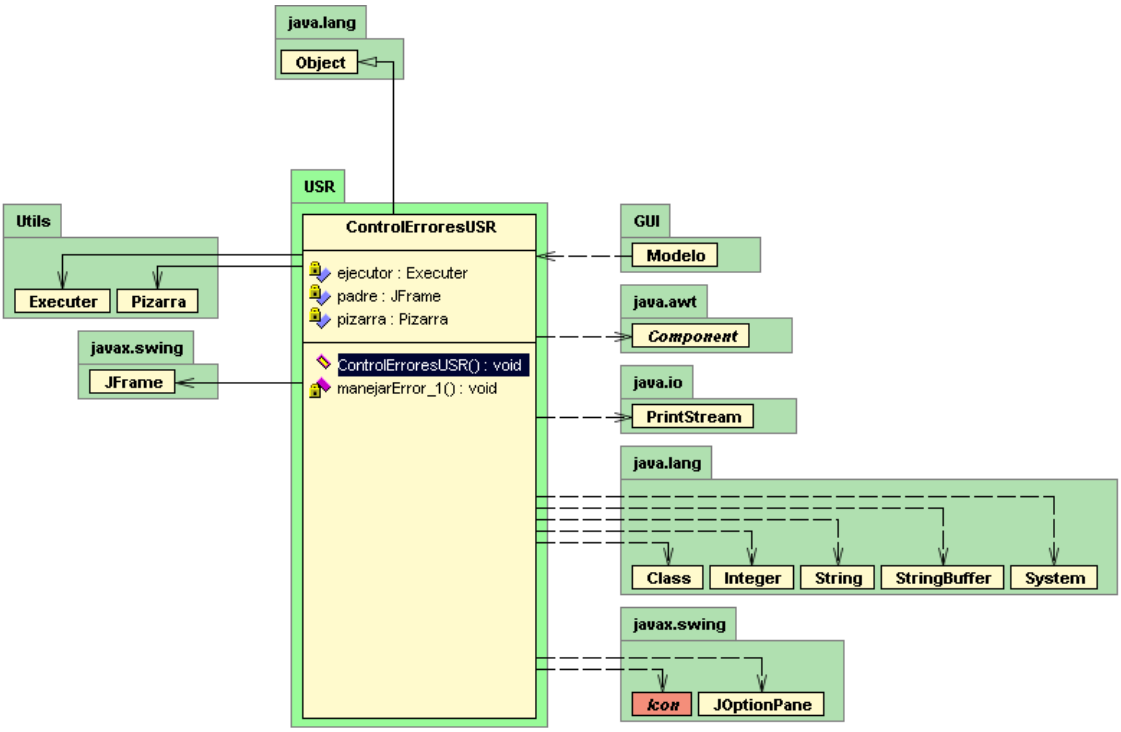

java.lang
Object
DYR
ControlErrores
ejecutor : Executer
padre : JFrame
pizarra : Pizarra
ControlErrores() : void
manejarError_0() : void
manejarError_1() : void
manejarError_2() : void
Utils
Executer
Pizarra
javax.swing
JFrame
DYR
Clock
GUI
Modelo
java.awt
Component
java.io
PrintStream
java.lang
Class
Integer
String
StringBuffer
System
javax.swing
Icon
JOptionPane
java.lang
Object
DYR
MotorDYR
action : String
casePlanning : String
conector : ConectorJavaOwl
herramientas : Utilidades
moodState : String
numberWorsenCycles : String
resultados : ArrayList
urgencyDegree : String
worsen : String
MotorDYR() : MotorDYR
run() : ArrayList
setInputSlotDiagnosis() : void
setInputSlotPrediction() : void
inicializarVariables() : void
medirSimilitudPrediction() : Individual
tareaDiagnosis() : void
tareaPlanning() : void
tareaPrediction() : void
DYR
RazonadorDYR
ConectorJavaOwl
ConectorJavaOwl
Utilidades
java.lang
String
java.util
ArrayList
ConectorJavaOwl
AccesoIndividuos
AccesoPropiedades
com.hp.hpl.jena.ontology
Individual
java.util
Iterator

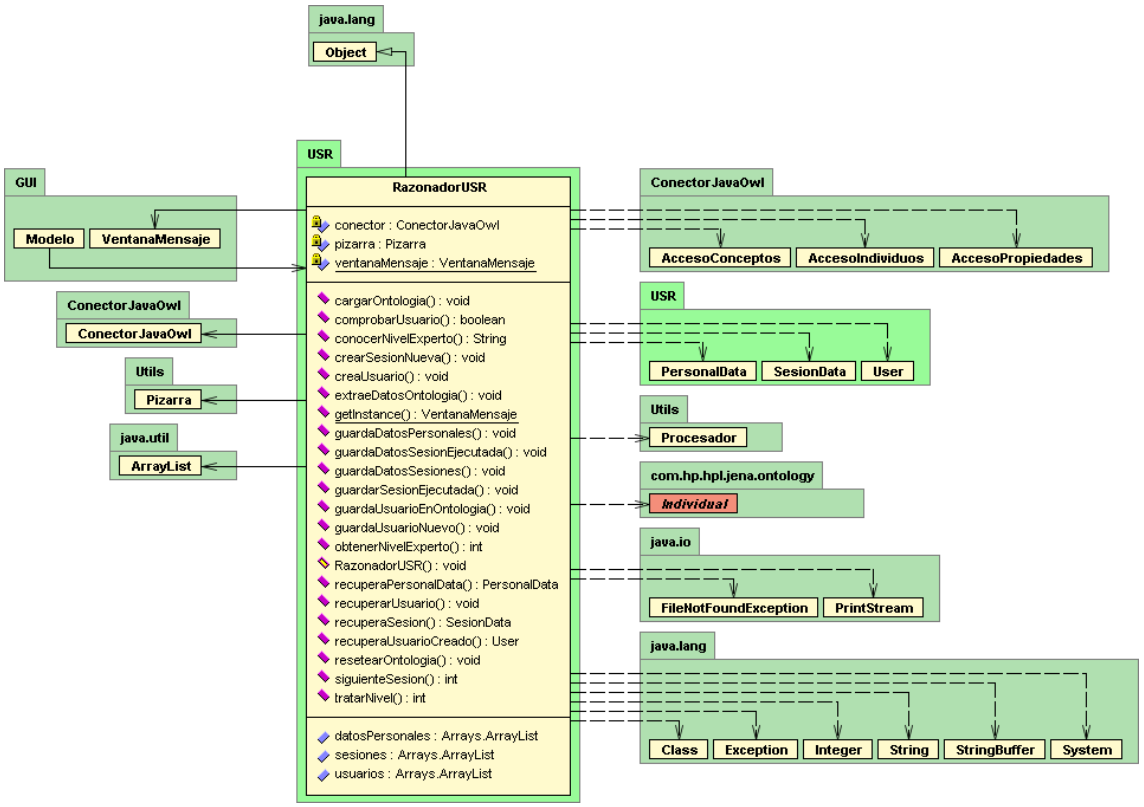

java.lang
Object
DYR
RazonadorDYR
GUI
Modelo
ConectorJavaOwl
ConectorJavaOwl
DYR
Clock
MotorDYR
Utils
Pizarra
java.lang
String
java.util
ArrayList
aumentoWorsenCycles : int
conector : ConectorJavaOwl
mejorUrgencyDegree : int
mood_state : String
mood_state_bef : String
motor : MotorDYR
pizarra : Pizarra
reloj : Clock
worsen_Degree : String
worsen_now : int
activarAutomatico() : void
analizarRespuestas() : void
cargarOntologia() : void
ejecutar() : void
getFaces() : ArrayList
getPredefinedState() : String
RazonadorDYR() : void
RazonadorDYR() : RazonadorDYR
resetearOntologia() : void
restartAutomatico() : void
setMoodState() : void
ajustarUrgencyDegree() : void
compararScore() : boolean
controlNumWorsenCycles() : void
getClockTime() : String
getCyclesNumber() : String
guardarNuevosDatos() : void
inicializarVariables() : void
perteneceNegativas() : boolean
pertenecePositivas() : boolean
setWorsenDegree() : void
resultados : ArrayList
numberWorsenCycles : String
urgencyDegree : String
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
AccesoPropiedades
Utils
Executer
com.hp.hpl.jena.ontology
Individual
java.lang
Class
Exception
Integer


java.lang
Object
USR
User
idUser : String
personalData : PersonalData
getHasPersonalData() : PersonalData
getHasSesionData() : SesionData
getHasSesionDataString() : String
longitudSesionData() : int
setHasPersonalData() : void
setHasSesionData() : void
User() : void
User() : void
User() : User
nivelExperto : String
nombre : String
posicionOcupada : int
sesionData : Arrays.ArrayList
USR
PersonalData
java.lang
String
java.util
ArrayList
GUI
Modelo
USR
RazonadorUSR
SesionData
java.io
PrintStream
java.lang
StringBuffer
System


java.lang
Object
USR
SesionData
nombreCasosVisitados : Arrays.ArrayList
nombreObjeto : String
añadeCaso() : void
getNombreCasos() : ArrayList
getnombreObj() : String
SesionData() : void
SesionData() : void
setnombreCasosVisitados() : void
setnombreObj() : void
idSesion : String
java.lang
String
java.util
ArrayList
GUI
Modelo
USR
RazonadorUSR
User


java.lang
Object
USR
PersonalData
USR
User
USR
RazonadorUSR
java.lang
String
dni : int
nombreObjeto : String
getDNI() : int
getNombreObj() : String
PersonalData() : void
setDNI() : void
setNombreObj() : void
apellidos : String
nombre : String


java.lang
Object
USR
ControlErroresUSR
Utils
Executer
Pizarra
javax.swing
JFrame
ejecutor : Executer
padre : JFrame
pizarra : Pizarra
ControlErroresUSR() : void
manejarError_1() : void
GUI
Modelo
java.awt
Component
java.io
PrintStream
java.lang
Class
Integer
String
StringBuffer
System
javax.swing
Icon
JOptionPane


java.lang
Object
USR
RazonadorUSR
conector : ConectorJavaOwl
pizarra : Pizarra
ventanaMensaje : VentanaMensaje
cargarOntologia() : void
comprobarUsuario() : boolean
conocerNivelExperto() : String
crearSesionNueva() : void
creaUsuario() : void
extraeDatosOntologia() : void
getInstance() : VentanaMensaje
guardaDatosPersonales() : void
guardaDatosSesionEjecutada() : void
guardaDatosSesiones() : void
guardarSesionEjecutada() : void
guardaUsuarioEnOntologia() : void
guardaUsuarioNuevo() : void
obtenerNivelExperto() : int
RazonadorUSR() : void
recuperaPersonalData() : PersonalData
recuperarUsuario() : void
recuperaSesion() : SesionData
recuperaUsuarioCreado() : User
resetearOntologia() : void
siguienteSesion() : int
tratarNivel() : int
datosPersonales : Arrays.ArrayList
sesiones : Arrays.ArrayList
usuarios : Arrays.ArrayList
GUI
Modelo
VentanaMensaje
ConectorJavaOwl
ConectorJavaOwl
Utils
Pizarra
java.util
ArrayList
ConectorJavaOwl
AccesoConceptos
AccesoIndividuos
AccesoPropiedades
USR
PersonalData
SesionData
User
Utils
Procesador
com.hp.hpl.jena.ontology
Individual
java.io
FileNotFoundException
PrintStream
java.lang
Class
Exception
Integer
String
StringBuffer
System

ANEXO 4

RECUPERACIÓN DE UN CASO

La recuperación de un caso en concreto, es necesaria cuando el usuario introduce una nueva consulta. A partir de aquí el sistema ya sabe desde que punto de la ontología debe de "moverse" para aproximarse a lo que el usuario necesita.

Cuando el usuario realiza una nueva consulta y pulsa sobre uno de los botones "Procesar", el sistema realiza los siguientes pasos:

- Un tratamiento del texto introducido por el usuario: lo quitarán las tildes, los espacios no deseados, los signos de puntuación y lo pondrá todo en minúsculas, para realizar una búsqueda estándar.
- Se procederá a la extracción de las palabras importantes del texto introducido, con el propósito de obtener un único término de indexación a partir de las diferentes variaciones morfológicas de una palabra.
- Consulta si existe alguna pregunta de activación como la realizada por el usuario.
    - Si existe, coge los nombres de los casos que tienen dicha pregunta para mostrárselos como opciones al usuario.
    - Sino, busca los casos de la ontología que puedan ser relevantes. Para ello:
        - Para cada una de las palabras importantes existentes en el texto del usuario, comprueba si es una de las palabras importantes que hay dentro de la ontología.
            - Si lo es,
                - Calcula para cada caso, el peso de la palabra en el mismo.
                - Aumenta la probabilidad del caso según el peso obtenido.
                - Si al final la probabilidad del caso es > 0, guarda el caso como una de las opciones posibles.
- Ordena los casos con las opciones posibles.
- Muestra al usuario los casos que tengan mayor probabilidad (hasta un máximo de 10 casos, según lo elegido por el usuario en el menú "Opciones…").

Para calcular el peso que una palabra tiene dentro de un caso de la ontología, se utiliza la siguiente fórmula:

PesoPalabraEnCaso = fdC * $\log_2$ (nCT / nCP)

Donde:

- fdC • frecuencia con la que aparece la palabra en el caso
- nCT • número de casos que tiene la ontología
- nCP • número de casos que tienen la palabra

La intuición es que una palabra muy frecuente en un mismo caso debe ser relevante para ese caso. Sin embargo, si una palabra es común en todos los casos es poco relevante. Las palabras raras tienen poca frecuencia, luego tienen poco peso.

ANEXO 5

PRESENTACIÓN

El objetivo del proyecto es definir un sistema de conversación a nivel de contenidos. Que tendrá un control de la conversación basado en:

1. La particularidad del dominio.
2. Tipo del sistema concreto a instanciar.

Está dividido en tres subtareas:

1. *Dominio*: es la que se encarga de los temas o contenidos de la aplicación. Tal cual están en los casos.
2. *Usuario*: adapta la respuesta global dependiendo de lo que ya sabe y su experiencia.
3. *Dinámica de la Conversación*: complementa la respuesta con un aspecto afectivo basado en el estado del usuario. Usa estrategias de la conversación.

Dotado de conocimiento proporcionado por:

- Ontologías: CCBRonto (conceptos para la mecánica de la conversación). Incluye a CBRonto (conceptos necesarios para CBR)
- Bases de casos: El conocimiento dinámico del dominio

Razonamiento: CBR Conversacional,

- Recibe una pregunta que será la descripción de casos a recuperar.
- Se presentan las respuestas de los casos recuperados.
- El usuario refina su descripción para recuperar nuevos casos hasta encontrar la respuesta buscada.

TEÓRIA:

ONTOLOGÍAS

La RAE (*Real Academia Española*) la define como *"parte de la metafísica que trata del ser en general y de sus propiedades transcendentes"*. Algunos autores piensan que las ontologías son *"colecciones de enunciados redactados en un lenguaje, como el RDF, que define las relaciones entre conceptos y especifica reglas lógicas para razonar con ellos"*.

Hay diversos autores que han definido el concepto de ontología pero la definición más aceptable de ontología parece ser la expresada por Gruber (1993): una ontología es *"la especificación explícita y formal sobre una conceptualización consensuada*", dentro de un dominio específico. La interpretación de esta definición es que las ontologías

definen sus conceptos, propiedades, relaciones, funciones, restricciones y axiomas de forma "explicita" en algún lenguaje de implementación capaz de contener este conocimiento

La mayoría de las ontologías existentes se caracterizan por los siguientes elementos comunes, que servirán para representar el conocimiento de algún dominio:

- ***Conceptos***: ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
- ***Relaciones***: representan la interacción y enlace entre los conceptos del dominio.
- ***Funciones***: son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
- ***Instancias***: se utilizan para representar objetos determinados de un concepto.
- **Axiomas**: teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología.

CBR (RAZONADOR BASADO EN CASOS)

Un Razonador Basado en Casos resuelve problemas nuevos mediante la adaptación de soluciones previas usadas para resolver problemas similares. La inspiración original de CBR se basó en el razonamiento humano a través del recuerdo.

Cuando CBR se aplica a problemas nuevos se aumentan los casos, el proceso CBR cambia y aprovecha todo lo nuevo que provee el problema resuelto.

El estudio de CBR está impulsado por dos motivaciones primarias.

1. El primero, el de la Ciencia Cognitiva, con el deseo de modelar la conducta humana.
2. El segundo, el de la Inteligencia Artificial, con el deseo de desarrollar tecnologías para hacer a los sistemas de IA más eficaces.

ESQUEMA

El esquema de la aplicación está basado en tres ontologías diferentes, cada una específica de uno de los razonadores (DOROnto, DYROnto y USROnto). Todas ellas comparten una ontología común, CCBROnto.

CCBROnto

Se ha definido CCBROnto, una ontología que incluye el esqueleto donde se pueden definir características adicionales del sistema CCBR.

El modelo del dominio define todo el conocimiento sobre el dominio e incluye la pregunta y los modelos de casos. El esqueleto de elementos más relevante del dominio del modelo son: descripción del dominio, base de casos del dominio, caso del dominio y un concepto genérico. El más importante de ellos es DomCaseBase.

DOROnto

DOROnto es la ontología correspondiente a la tarea del dominio de la aplicación. En ella se incluye toda la información a procesar y a mostrar al usuario.

DYROnto

DYROnto es la ontología correspondiente a la tarea de la dinámica de la conversación. Esta ontología incluye toda la información necesaria para conocer los estados de ánimo posibles del usuario y posee las transiciones posibles dependiendo de los cambios anímicos producidos en el usuario.

USROnto

USROnto es la ontología correspondiente a la tarea de adaptación de la conversación, a la experiencia y conocimiento del usuario. Almacena la información correspondiente a cada usuario que se registra en la aplicación.

TAREAS DETALLADAS

Las tres principales subtareas de las que está compuesta la aplicación: Dominio (DOR), Usuario (USR), Dinámica de la Conversación (DYR); interactúan entre si mediante una pizarra, un ejecutor y una agenda.

En cada ciclo de la ejecución, cuando el usuario introduce una consulta, está es procesada por las tareas de la aplicación; que

interactúan con los datos almacenados en las ontologías, y obtienen una salida como resultado de la consulta.

Si el usuario no es capaz de ofrecer una buena descripción del problema, el razonador no será capaz de encontrar un caso apropiado. Por ello, el sistema va guiando al usuario para entender mejor el problema. Si podemos formalizar cada tarea de forma separada y buscar una solución para cada uno, entonces podemos buscar soluciones parciales incrementadamente hasta que cubramos el tema principal.

Teniendo en cuenta el dominio o tema, se puede distinguir entre sistemas de diagnóstico, soporte de decisión, recomendaciones y sistemas que justifican sus recomendaciones,…la aplicación está diseñada para que funcione indiferentemente del tema a tratar. Para ello, sólo habría que modificar la ontología del dominio con el nuevo tema.

RAZONADOR DYR

A la vez que el usuario interactúa con los otros razonadores, el DYR va comprobando, el grado de satisfacción que le producen al usuario los resultados que va obteniendo, el estado de ánimo del usuario, el carácter del usuario, el método de aprendizaje del usuario, el modo de uso de la aplicación por parte del usuario.

Todo esto lo realiza para obtener una serie de parámetros, los cuales van a servir para poder ajustar cada vez más el tema de conversación con el usuario, el modo de la conversación, el tono de la conversación, etc.

Todo orientado lo mayormente posible a las características personales del usuario, para poder ajustar y personalizar tanto la aplicación como las conversaciones con el usuario.

El Razonador DYR, posee tres subtareas:

- Subtarea Diagnosis: Método de *Decisión de Cambiar a otro Bucle*
  Conocimiento: Ontologías y Bases
  Razonamiento
- Subtarea Predicción: Método de *Selección del nuevo Bucle*
  Conocimiento: Ontologías y Bases
  Razonamiento
- Subtarea Planificación: Método que *Implementa el nuevo Bucle*
  Conocimiento: Ontologías y Bases
  Razonamiento

La implementación del razonador DYR, se ha intentado hacer de manera sencilla. Posee un paquete propio con las clases necesarias para su trabajo personal, dentro de estas clases la más relevante es la

clase RazonadorDYR, la cual posee método específico para recoger toda la información necesaria para la ejecución de un ciclo del razonador. Una vez recabada toda esta información, existe otra clase, MotorDYR, a la cual el razonador la pasa toda la información recogida anteriormente y esta ejecuta cada una de las tareas descritas, devolviendo al razonador los resultados de la ejecución de un ciclo.

RAZONADOR USR

Para un correcto control en el uso de la aplicación, el usuario debería registrarse en la misma, para así conocer cuál es su grado de experiencia y qué es lo que más le conviene en cada caso.

La información se obtiene a través de unas breves cuestiones que se le realizan al usuario al entrar en la fase de registro.

Si está registrado o si no lo está, se le realiza un breve test para saber su nivel de conocimiento o si, en sesiones anteriores, aprendió algo.

Una vez el usuario introduzca una cuestión, el sistema, a la vez que busca lo que más se aproxima a la petición, mostrará un mensaje si algunas de las posibilidades a mostrar ya han sido vistas en sesiones anteriores o en la actual.

Cuando elija una opción, ésta se guardará para, posteriormente y una vez se haya acabado la sesión, pueda volcarse a la ontología y quedar así registrado para visitas posteriores.

Dependiendo del nivel de conocimiento el sistema debería mostrar unos casos u otros, pero esta parte no ha sido implementada, puesto que la ontología utilizada es demasiado sencilla.

ANEXO 6

EXPERIMENTOS Y RESULTADOS

En este apartado, se van ha incluir una serie de opiniones de usuarios de la aplicación. Todos estos usuarios, se decidió que fuesen anónimos, ya que lo que más interesa es su opinión y no sus datos personales.

Las pruebas consistieron:

Primero se realizó una breve explicación, a los usuarios, de que es lo que iban a hacer.

Segundo se explicó de manera breve, en que consistía la aplicación creada y como funcionaba.

Tercero, se dejó la aplicación a dichos usuarios para que la probasen de manera absolutamente libre.

Y por último se les pasó una tabla para que la rellenasen y poder obtener sus opiniones y comentarios.

Los resultados obtenidos fueron los siguientes:

USUARIO	USUARIO_01				
EDAD	20				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?				X	
¿Qué te parece la interfaz gráfica de la aplicación?				X	
¿Cómo calificaría el funcionamiento de la aplicación?			X		
¿Qué te ha parecido la ventana DOR?				X	
¿Qué te ha parecido la ventana USR?				X	
¿Qué te ha parecido la ventana DYR?				X	
Califica la utilidad de la aplicación			X		
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
Mi opinión es buena, en general, aunque no se entiende muy bien para que sirve la aplicación.					

USUARIO	USUARIO_02				
EDAD	26				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?				X	
¿Qué te parece la interfaz gráfica de la aplicación?			X		
¿Cómo calificaría el funcionamiento de la aplicación?				X	
¿Qué te ha parecido la ventana DOR?			X		
¿Qué te ha parecido la ventana USR?			X		
¿Qué te ha parecido la ventana DYR?			X		
Califica la utilidad de la aplicación			X		
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
No se ve, de manera clara la finalidad de la aplicación.					

USUARIO	USUARIO_03				
EDAD	25				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?				X	
¿Qué te parece la interfaz gráfica de la aplicación?		X			
¿Cómo calificaría el funcionamiento de la aplicación?				X	
¿Qué te ha parecido la ventana DOR?				X	
¿Qué te ha parecido la ventana USR?			X		
¿Qué te ha parecido la ventana DYR?		X			
Califica la utilidad de la aplicación				X	
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
La aplicación me a parecido buena, sobre todo su finalidad, una vez me la han explicado, pero no me gusta la idea de que una aplicación intente establecer mi estado de ánimo.					

USUARIO	USUARIO_04				
EDAD	40				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?		X			
¿Qué te parece la interfaz gráfica de la aplicación?			X		
¿Cómo calificaría el funcionamiento de la aplicación?			X		
¿Qué te ha parecido la ventana DOR?				X	
¿Qué te ha parecido la ventana USR?				X	
¿Qué te ha parecido la ventana DYR?				X	
Califica la utilidad de la aplicación			X		
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
Mi conocimiento sobre la informática es bastante reducido, así que no le veo una gran utilidad a la aplicación, pero me ha gustado poder probarla.					

USUARIO	USUARIO_05				
EDAD	23				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?				X	
¿Qué te parece la interfaz gráfica de la aplicación?					X
¿Cómo calificaría el funcionamiento de la aplicación?				X	
¿Qué te ha parecido la ventana DOR?				X	
¿Qué te ha parecido la ventana USR?				X	
¿Qué te ha parecido la ventana DYR?				X	
Califica la utilidad de la aplicación				X	
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
Me gusta la finalidad que tiene esta aplicación, aunque por lo que he podido observar necesita de algunas mejoras en su ejecución, y ampliar su conocimiento para poder verla en su funcionamiento completo.					

USUARIO	USUARIO_06				
EDAD	30				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?					X
¿Qué te parece la interfaz gráfica de la aplicación?				X	
¿Cómo calificaría el funcionamiento de la aplicación?				X	
¿Qué te ha parecido la ventana DOR?					X
¿Qué te ha parecido la ventana USR?				X	
¿Qué te ha parecido la ventana DYR?				X	
Califica la utilidad de la aplicación				X	
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
Le faltaría mejorar un poco el conocimiento de la aplicación, pero en general me parece bastante buena y muy aprovechable, para distintos campos.					

USUARIO	USUARIO_07				
EDAD	43				
Califica cada cuestión con los valores	Muy malo	Malo	NS/NC	Bueno	Muy bueno
¿Cómo de intuitiva es la aplicación?			X		
¿Qué te parece la interfaz gráfica de la aplicación?				X	
¿Cómo calificaría el funcionamiento de la aplicación?			X		
¿Qué te ha parecido la ventana DOR?			X		
¿Qué te ha parecido la ventana USR?			X		
¿Qué te ha parecido la ventana DYR?			X		
Califica la utilidad de la aplicación			X		
Escribe una breve opinión de lo que te ha parecido la aplicación en global.					
La verdad es que no me gusta la informática, con lo cual mi opinión sobre la aplicación no creo que sea muy válida.					

La verdad es que todos los usuarios que han pasado el test de la aplicación son amigos, con lo cual su opinión siempre suele ser amistosa, por lo que existen ciertos parámetros que no se pueden evaluar bien, o si los evaluamos, podemos obtener resultados un poco engañosos. Lo que si que hemos intentado es que estos usuario no sean todos del mismo rango de edad, para poder obtener una opinión más variada.

En general se puede observar como la opinión sobre la aplicación es buena, aunque se puede observar como la funcionalidad de la aplicación no se comprende fácilmente. A los usuarios, que probaron la aplicación, les costó comprender cual es la finalidad de la aplicación y también les costó entender que lo que estaban probando es un prototipo.

También se puede observar como la calificación de la interfaz gráfica es buena, esto nos hace pensar que la elección que se hizo para hacer la interfaz fue la acertada, pero se hicieron algunos comentarios, por parte de los usuarios que la probaron, que hace pensar que se podrían mejorar algunos aspectos, para que la interfaz sea más amigable.

En el aspecto negativo, que se ha podido recoger de las pruebas, es que varios usuarios han visto que es necesario incrementar el conocimiento de la aplicación para poder observar mejor todo su funcionamiento, también alguno de ellos no se sintió muy cómodo, ó incluso muy satisfecho, con la idea de que una aplicación le esté evaluando todo el rato su estado de ánimo.

ANEXO 7

RESUMEN DEL PROYECTO EN CASTELLANO

Esta aplicación es un Sistema Conversacional Inteligente, basado en razonamiento por casos.

Su objetivo es la orientación para conseguir un fin mediante el uso de un seguimiento activo del usuario y del conocimiento que éste va adquiriendo.

Está comstituido por tres tareas: DOROnto, USROnto y DYROnto, cada una de las cuales tiene sus propias funciones.

DOROnto: es la que se encarga de los temas o contenidos de la aplicación.

USROnto: adapta la respuesta global dependiendo de lo que ya sabe el usuario y su experiencia.

DYROnto: complementa la respuesta con un aspecto afectivo basado en el estado del usuario. Usa estrategias de la conversación.

RESUMEN DEL PROYECTO EN INGLES

This aplication is an Inteligent Conversational System, based en case reasoner.

Its objetive is to orient for get a cause using active pursuit of the user and its knowledge.

Its based on three parts: DOROnto, USROnto and DYROnto. Each one has its own functions.

DOROnto: It has the subjects or contents of the aplication.

USROnto: Adapts the answer in relation of its experience amd what it knows

DYROnto: It work with the emotional aspect based in the user state. It uses conversational strategies.

ANEXO 8

LISTADO DE PALABRAS CLAVE

- Conversación
- Razonamiento
- Razonador
- Conocimiento
- Casos
- Experiencia
- Estado

Agradecimientos

<< Ocurra lo que ocurra, aún en el día más borrascoso, las horas y el tiempo pasan. William Shakespeare (1564-1616). Escritor británico.>>

<< Más veces descubrimos nuestra sabiduría con nuestros disparates que con nuestra ilustración. Oscar Wilde (1854-1900). Dramaturgo y novelista irlandés.>>

<< Siempre que te pregunten si puedes hacer un trabajo, contesta que sí y ponte enseguida a aprender como se hace. Franklin Delano Roosevelt (1882-1945) Político estadounidense. >>

Los autores de este proyecto queremos agradecer a todos aquellos que con consejos, colaboraciones, críticas, sugerencias, pedidos y estímulo, nos han ayudado a desarrollar este Sistema Conversacional:

En especial nuestros amigos, compañeros y familiares por su constante apoyo y las críticas nacidas de su nutrida experiencia; por su seguimiento de nuestro trabajo, su generoso consejo, y su permanente apoyo.

A nuestro tutor, Héctor Gómez Gauchía por brindarnos su valiosa ayuda, su paciencia, su experiencia y su amable colaboración.

También a los investigadores, los medios y sitios web por las noticias, artículos, fotos y otros materiales, que tuvieron la gentileza de facilitarnos.

A todos, gracias.

Autorización

Los autores de este proyecto, Laruska Roldán Hontanilla, Rubén Rivilla Aparicio y Mercedes Huertas Migueláñez, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar, con fines académicos y no comerciales, la memoria, el código y el prototipo desarrollado sobre este Sistema Conversacional Inteligente.

Firma de los autores:

Laruska Roldán Hontanilla Rubén Rivilla Aparicio

Mercedes Huertas Migueláñez