UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Carrera de Ingeniero Informático

UTILIZACIÓN DE JENA Y RMI PARA MAPEO DINÁMICO ENTRE ONTOLOGÍA Y BASE DE DATOS

ANSGAR MIKEL NELL ARTOLAZABAL

Dirigido por: RAFAEL MARTÍNEZ TOMÁS

Curso: 2011/2012

UTILIZACIÓN DE JENA Y RMI PARA MAPEO DINÁMICO ENTRE ONTOLOGÍA Y BASE DE DATOS

Proyecto de Fin de Carrera de modalidad proyectos realizados con empresas o instituciones

Realizado por: ANSGAR MIKEL NELL ARTOLAZABAL

Dirigido por: RAFAEL MARTÍNEZ TOMÁS

Tribunal calificador:

Presidente: D./Da
. .....
(firma)

Secretario: D./Da
. .....
(firma)

Vocal: D./Da
. .....
(firma)

Fecha de lectura y defensa:
.....
Calificación:
.....

A mi padre.

A todos los que me han ayudado a seguir adelante.

A todo el departamento de Inteligencia Artificial de I3B, especialmente a Aitor Moreno por confiar en mí y darme la oportunidad de colaborar con ellos.

Resumen

El presente proyecto aborda la forma de mantener actualizada en tiempo real una ontología, en el ámbito de un almacén de productos alimenticios, respecto a la base de datos relacional en la que reside la información.

En primer lugar se realiza una introducción para situar el trabajo dentro de un proyecto mayor denominado Hiridenda-2020 que está siendo desarrollado por la empresa I3B, Instituto Ibermática de Innovación, perteneciente a la empresa Ibermática S.A., así como conocer los objetivos, la metodología seguida y la organización de la memoria.

A continuación, se realiza un repaso sobre el estado del arte actual, analizando la aplicación de nuevas tecnologías a almacenes, las diferentes utilizaciones y logros conseguidos mediante el uso de ontologías, especialmente en el mundo de la logística, para finalmente abordar el problema principal estudiando distintas formas de trabajar conjuntamente con bases de datos relacionales y ontologías.

Como información complementaria se han recogido algunas aplicaciones comerciales en uso actualmente que están relacionadas con alguno de los puntos abordados anteriormente, para mostrar la utilidad en el mundo real de las necesidades que se han estudiado.

Tras este repaso, se presenta la estructura de la solución adoptada, explicando a modo introductorio los distintos mecanismos y aplicaciones que se han usado para resolver el problema planteado, incluyendo la ontología creada, mostrando a continuación la arquitectura conseguida ensamblando las distintas piezas explicadas. En este punto se aporta información detallada de la aplicación que se ha implementado, así como de su utilización y acciones a realizar para su funcionamiento. Además, se enumeran las distintas tablas que componen la base de datos y la información que contienen. También se aportan datos sobre los resultados obtenidos y las líneas de trabajo abiertas.

Finalmente se han planteado las conclusiones obtenidas tras la realización del presente proyecto, donde se explican las metas conseguidas, las aportaciones rea-

lizadas a la formación del alumno, y las partes que se ha considerado que no han podido ser desarrolladas en su totalidad.

Palabras clave

Ontologías, mapeo dinámico de ontologías, logística, RMI, procedimientos almacenados, disparadores

English Title

USE OF JENA AND RMI FOR DYNAMIC MAPPING BETWEEN ONTOLOGY AND RELATIONAL DATABASES

Summary

This paper analyzes the way to keep an ontology up-to-date in real time in a food products storage environment, relative to the relational database where the information lives.

First an introduction to the Hiridenda-2020 project, which is being developed by I3B, the Ibermatica Institute for Innovation, belonging to Ibermática S.A., is made in order to put the present work in its context. Moreover, the goals, used methodology and the structure of this paper will be explained

In addition to this, a summary of the current state of the art is made, analyzing the application of new technologies to warehouses, the different uses and goals achieved by using ontologies, specially in the logistics area, ending with the main problem considering different ways of working together with relational databses and ontologies.

As a complementary information some commercial applications, related to some of the analyzed aspects that are being used nowadays, have been studied, in order to show the applicability that the considered matters have in real world.

Following this review, the structure for the considered solution is shown, with a brief explanation of the mechanisms and applications that have been used to solve the given problem, including the created ontology, and showing afterwards the achieved architecture after assembling the different explained pieces. At this point a detailed explanation is given about the implemented application, its use and steps that have to be taken for getting it working. The different database tables and the information they contain are explained in detail as well. There is also information on the results obtained and the new opened working lines which have been opened

as a consequence.

Finally the conclusions obtained after doing this project are shown, where the achieved goals are explained, the contributions to the student's education, and those parts which are considered to not have been developed completely.

Keywords

Ontologies, dynamic ontologies mapping, logistics, RMI, stored procedures, triggers

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El trabajo con la información ha sido desde sus inicios una de las características de la informática. La posibilidad de almacenar y manejar de una forma cómoda y rápida datos ha permitido crear bases de datos colosales cuya información puede ser recuperada fácilmente simultáneamente por distintas personas ubicadas en lugares alejados de la fuente.

La utilización de ontologías (ver 2.2) permite llevar la información a un nivel más cercano a la forma de pensar de los humanos. Aplicando ontologías sobre bases de datos, aumenta el conocimiento sobre los datos almacenados, lo que permite obtener resultados que no pueden ser logrados de otra forma. Si tomamos como ejemplo el caso de animales, podemos pasar de tener datos para cada especie a reflejar toda la estructura de clasificación animal con sus géneros, familias, reinos etc.

El presente trabajo aborda el problema de mantener la coherencia entre la información contenida en una base de datos y la contenida en una ontología. El carácter dinámico de la base de datos obliga a actualizar la información en cuanto se produce el cambio, y para realizarlo se ha desarrollado una arquitectura compuesta por diferentes tecnologías que funciona en tiempo real.

Este proyecto se ha realizado en colaboración con la empresa I3B (Instituto Ibermática de Innovación), perteneciente al grupo Ibermática, donde se está desarrollando el proyecto HIRIDENDA-2020[1] que busca modernizar la forma de funcionar de

[1]La palabra hiridenda está formada por las palabras en euskara *hiria*, cuyo significado es ciudad,

los comercios urbanos, introduciendo nuevas formas de entender la información.

1.1. Objetivos principales

Para comprender el objetivo principal de este trabajo es necesario tener muy clara la arquitectura con la que se va a trabajar. Por un lado, tenemos un almacén comercial, donde se ubican todos los productos que son vendidos a través de un negocio. Como ejemplo, se ha tomado un supermercado de tamaño medio en el que, además de los productos que se pueden encontrar en las estanterías en la zona de venta, existen otros almacenados a la espera de reemplazar las existencias agotadas. Este trabajo se centra solamente en esta segunda zona, ignorando lo que sucede en la zona de ventas. El almacén cuenta con una base de datos relacional, que es alimentada desde distintas zonas; muelles de carga, donde los encargados de recepción introducen la información sobre los lotes que traen los transportistas; movimientos internos dentro del almacén, datos que serán aportados por los encargados de mover los productos; movimientos hacia la zona de ventas, que serán grabados como salidas, etc. Lógicamente la información que contiene la base de datos es dinámica, produciéndose cambios cada poco tiempo. Por otro lado, se ha definido una ontología que representa el conocimiento sobre el trabajo que se realiza en el almacén, sus productos, localizaciones...La estructura de esta ontología, sus clases, son en general estáticas, ya que una vez que se ha definido el conocimiento no se prevén cambios en el mismo. Las instancias que se manejan, en cambio, deben cambiar a la misma velocidad que los cambios que se producen en la base de datos.

El problema que afronta el presente proyecto es el de cómo mantener dinámicamente la coherencia en la información entre una base de datos relacional y la ontología que se ha creado sobre dicha información. Es decir, la base de datos y la ontología sólo tiene una relación virtual en el sentido de que la segunda se ha definido en parte basándose en el conocimiento que se ha detectado en la primera, pero

y *denda*, que significa tienda o comercio. Por lo tanto, una traducción adecuada sería comercio urbano.

esta relación no es real, ya que no existe ningún vínculo que traslade los cambios que se producen en una a la otra. Además, para que la solución sea útil, se pretende que las actualizaciones sucedan simultáneamente en ambos entornos.

1.2. Objetivos posteriores

Este proyecto no es más que un primer paso dentro de una planificación mayor, que no puede ser abarcada debido a la complejidad y necesidad de tiempo que exige. Por lo tanto, para situar lo realizado y comprender su necesidad, es necesario explicar el trabajo que se pretende hacer en un futuro.

Una vez logrados los objetivos propuestos en este trabajo, dentro del proyecto Hiridenda, lo desarrollado aquí se pretende utilizar para aplicar técnicas de Inteligencia Artificial a los almacenes de dichos comercios, de forma que la gestión de la información de los productos se haga de una forma más óptima y útil que en la actualidad, y que la resolución y prevención de las incidencias sea más rápida y automatizada.

Por lo tanto, lo que se pretende lograr posteriormente, es que a partir de un esquema clásico de almacén controlado mediante una base de datos relacional, y valiéndonos de las ventajas que ofrece el uso de ontologías, un sistema inteligente que apoye en su gestión y permita detectar todas aquellas situaciones anómalas, indicando su naturaleza concreta así como su localización.

Algunas de las tareas que se deberían poder llevar a cabo con este sistema son:

1. Detectar e informar de cualquier incidencia que ocurra en el almacén o relacionado con él, a través de los datos de los que se dispone:
    - Temperatura ambiental
    - Posición de la mercancía
    - Características de la mercancía
    - Temperatura de la mercancía

- Humedad ambiental
- Luminosidad
- Fecha/hora de carga/descarga
- Datos logísticos (empresa, camión, albarán, pedido, entrega, tipo de producto...)
- Fecha caducidad del producto
- Almacenero
- Almacén
- Playa de embarque/descarga

Por ejemplo:

- Si la temperatura adecuada para un producto está entre los valores "x" e "y", y el sensor del producto marca una temperatura menor, o el sensor está estropeado, o debemos lanzar una incidencia.
- Si los productos perecederos se ubican siempre en la zona frigorífica, si se ubica un producto fuera y está más de x tiempo, lanzar una alerta al respecto.
- Si un transportista siempre carga n palés, y un día carga una cantidad menor significativa, emitir una alerta...

2. Servir como apoyo a personal con menos experiencia, de manera que puedan realizar labores propias de un experto.

3. Permitir obtener información sobre el estado del almacén de una forma rápida y precisa.

1.3. Metodología

La forma de abordar el presente trabajo ha sido la clásica de un proyecto de investigación. En primer lugar, se ha estudiado en profundidad la literatura existente

sobre los diferentes temas que se han tratado. La principal fuente de información han sido artículos publicados en revistas especializadas, mostrando especial atención a la actualidad de los mismos.

La búsqueda de temas concretos ha sido en un principio difícil por no conocer exactamente los métodos disponibles, y por existir mucha información sobre automatización de almacenes o uso de ontologías, pero menos sobre la mezcla de ambos campos.

También se han estudiado libros (como en el caso de RMI) para refrescar conocimientos adquiridos en la carrera, así como programas o artículos disponibles en Internet para analizar ejemplos de implementaciones.

Una vez determinada la estructura deseada para llevar a cabo el objetivo propuesto, se han realizado pruebas por separado de cada una de las técnicas hasta conocer y confirmar su funcionamiento y utilidad. Esta parte ha sido una de las más laboriosas, por abordar el presente trabajo campos muy distintos como bases de datos con comandos SQL, la utilización de ontologías con Jena y Protégé, el lenguaje de programación Java, o la ya mencionada RMI.

Después se ha procedido a unir todos los elementos en una estructura común, que ha sido finalmente probada con casos ficticios y relativamente simples.

1.4. Organización de la memoria

La presente memoria se ha dividido en tres capítulos principales:

- Capítulo 1: en este capítulo se hace una breve introducción al proyecto, para entender en qué contexto se ha desarrollado. Asimismo, también se explica la metodología utilizada y la presente organización de la memoria.

- Capítulo 2: se repasa el estado actual de los distintos temas abordados, para dar una idea de cuáles son las soluciones que se han venido adaptando tanto en la investigación como en el mundo empresarial para hacer frente a los retos planteados. Las descripciones se han dividido en cuatro campos:

1. Logística
2. Ontologías
3. Actualización dinámica de ontologías
4. Aplicaciones comerciales

- Capítulo 3: este es el capítulo principal de la memoria, donde se explica detalladamente la solución adoptada. Se comienza repasando las tecnologías que se han utilizado y la ontología generada, para mostrar a continuación la arquitectura de la aplicación que se ha desarrollado. Tras esta introducción, se procede a realizar un repaso detallado del programa y se explican los pasos a seguir para su utilización. También se ha considerado importante explicar el contenido de la base de datos y el propósito de cada una de sus tablas.

Capítulo 2

Trabajos relacionados

2.1. Sistemas de apoyo a la logística

Los almacenes y el sector de la logística han ido introduciendo, junto con el resto de los sectores económicos y la sociedad en general, nuevas tecnologías a su funcionamiento diario. Es habitual observar hoy en día cómo un trabajador utiliza una PDA o dispositivo similar para realizar lecturas sobre algún producto, o el uso de un programa informático mediante el que se realiza la gestión de una tienda.

Este hecho, unido al abaratamiento de los costes a la hora de adquirir la tecnología necesaria para equipar a los productos de sensores e identificadores [GMRS07] como etiquetas RFID (para una introducción a esta tecnología consultar [Wan06]) o redes inalámbricas, ha hecho posible generalizar el uso de estos elementos en tareas de monitorización y seguimiento de productos ([WMAZA02, CHJG+02] entre otros). Los distintos caminos de distribución dentro de un almacén se equipan con lectores que obtienen los datos de las etiquetas RFID, permitiendo conocer de qué producto se trata, para poder dirigir el producto a su destino, y mantener actualizado automáticamente el inventario. Un animal sacrificado en el matadero es controlado e identificado sin posibilidad de error gracias a un chip que se le implantó hace años, que se recupera fácilmente sin riesgo para el consumidor, lo que permite garantizar su origen, raza, destino, en definitiva, la calidad del producto.

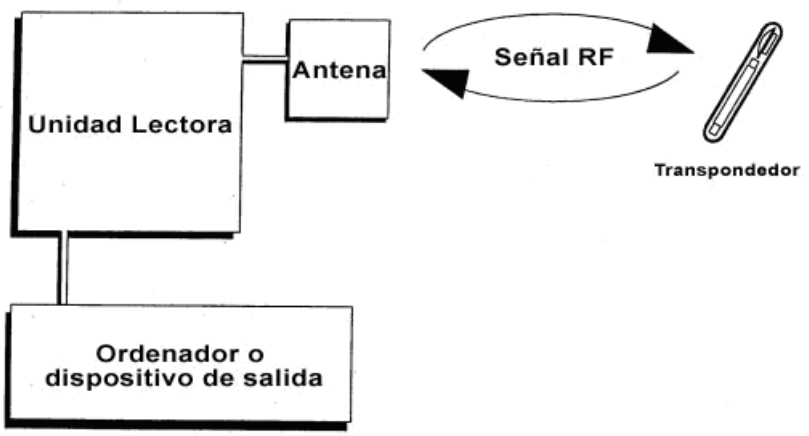

Figura 2.1: Sistema típico de RFID (adaptado de [DKA+09])

En [Gar08] podemos encontrar un sistema de monitorización del estado de productos perecederos durante todo el ciclo de transporte. El autor realiza un análisis en profundidad de los sensores existentes en el mercado, las diferentes técnicas de toma y envío de datos, así como una interfaz que permita al usuario final llevar el control de todos los productos de la empresa.

Por otro lado, la introducción de técnicas de inteligencia artificial ha permitido ir más allá de las posibilidades ofrecidas por la automatización pero supervisadas por humanos. Entre estas técnicas encontramos los sistemas basados en conocimiento (SBC) que permiten apoyar, o incluso sustituir, a los humanos en la toma de decisiones, a partir del análisis de la información proporcionada por el sistema. Tenemos ejemplos como una cadena de corte automático de pescado [dSW97], donde cada pez es analizado para obtener los datos necesarios que permitan extraer el máximo de cada unidad. El sistema se ha estructurado en tres capas: en la primera encontramos sensores y cámaras, en la segunda se procesa la información que se envía a la tercera y última capa, donde se sitúan la base de datos y la base de conocimientos. Otro ejemplo es un sistema de apoyo al control de satélites militares [GS92] que permite afrontar situaciones de crisis o falta de personal cualificado. Sin llegar a sustituir a los humanos que trabajan en el sistema, analiza los datos referentes a conexiones, disponibilidad, etc. facilitando el trabajo a los operadores.

En la literatura también podemos encontrar estudios [MMV05] en los que por

cada producto existe un agente software, haciendo uso de técnicas de sistemas inteligentes distribuidos. Cada agente es capaz de realizar autónomamente acciones como planificación de entregas, asignación de ubicaciones...además de hacer uso de las características propias de los agentes software como son la negociación, la tolerancia a los cambios o caídas parciales, entre otros. Estos productos auto-identificables tienen los siguientes beneficios:

- Mayor exactitud en la realización de inventarios y seguimientos de productos
- Reabastecimientos más efectivos
- Protección frente a falsificaciones
- Identificación de desvíos
- Identificación de robos
- Gestión eficaz del inventario

En [CCLW07] encontramos un sistema basado en agentes que sirve como soporte a la gestión de un almacén y sus procesos logísticos asociados. Implementado en la empresa Eastern Worldwide (EWW), permite controlar las distintas etapas por las que pasa un producto, permitiendo saber su ubicación y estado, así como controlar y prevenir posibles incidencias como un peso excesivo al apilar cajas de leche.

También en Internet podemos encontrar aplicaciones comerciales como VeriWise trucking[1], que es un producto desarrollado por la compañía norteamericana GE que permite realizar un seguimiento de los camiones de una empresa mostrando información sobre rutas, tiempo de transporte, posibles incidencias como el estado de los neumáticos, estado de las puertas, temperatura del refrigerador, etc. Su utilización se basa en colocar sensores en distintos puntos de los camiones, como las puertas, el sistema de refrigeración o el enganche del remolque. Estos sensores se comunican con la central mediante la infraestructura de la telefonía móvil o por satélite. Mediante esta información el responsable puede comprobar en tiempo real hechos como que

[1] http://www.ge.com/equipmentservices/assetintelligence/solutions/segments/trucking/index.html

no se ha roto la cadena de frío de los alimentos, o que la utilización de combustible por parte de un empleado es correcta.

2.2. Ontologías y logística

Una ontología (del griego οντος, genitivo del participio del verbo εἰμί, ser, estar; y λόγος, ciencia, estudio, teoría), expresa, de una manera formal y exhaustiva, todo el conocimiento relacionado con un área o dominio concreto. Es la representación formal de los conceptos más importantes de un dominio y las relaciones entre ellos, que crea una estructura de conceptos relacionados proporcionando un vocabulario común de forma que la información pueda ser compartida. En una ontología, las clases se organizan jerárquicamente, permitiendo tener subclases y superclases, relacionadas a su vez entre ellas [Jim08].

Son varios los trabajos donde se aplica o se crea una ontología al ámbito de la logística, como en [LPK07] donde se ha desarrollado una ontología de logística mediante una semántica de situaciones, de forma que el sistema de notificaciones pueda conocer la situación correcta de cada producto.

Debido a las discrepancias existentes entre las diferentes ontologías incluso dentro de un mismo campo, se ha desarrollado un sistema para encontrar las relaciones semánticas entre distintas ontologías dirigido a procesos de comercio electrónico y logística [NFX09].

Para apoyar la toma de decisiones en el ámbito de la logística, encontramos un sistema que utiliza ontologías para resolver las diferencias existentes entre los sistemas de información y los sistemas de toma de decisiones, donde se ha desarrollado un prototipo denominado UBLDSS que sirve como puente entre los sistemas mencionados anteriormente [JYY08].

En [CT09] tenemos un ejemplo en el que para poder realizar el control global de un sistema de ensamblado de bicicletas mediante agentes, se ha implementado una ontología que engloba un sistema de etiquetado de piezas de bicicleta mediante etiquetas RFID resistentes a ambientes extremos (sección de pintura, soldadura etc.)

y cada uno de los agentes que se encarga de una parte de la cadena. Este sistema permite al cliente consultar en tiempo real el estado de su pedido, sin que sea necesaria la intervención humana para actualizarlo. Por otro lado, para solventar la dificultad de escribir en etiquetas RFID, aunque virtualmente toda la información se encuentre en la propia etiqueta, realmente se utiliza un sistema con una base de datos centralizada. Todo este sistema se encuentra en funcionamiento en una empresa real, en la que se han sustituido las pegatinas con códigos de barras por etiquetas RFID, evitando las lecturas manuales, más propensas a errores, y las zonas donde las piezas, debido a la temperatura o pintura, no podían ir etiquetadas.

La creación de ontologías manualmente es una tarea ardua, cara y consume mucho tiempo [CB]. Además este método es más propenso a la aparición de errores [JSF09]. Existen muchas investigaciones que tienen como objetivo lograr extraer automáticamente ontologías a partir de ontologías más generales, páginas web, o noticias en forma escrita [TM09].

También existe la posibilidad de crear la ontología basándose en registros de datos [DKA+09], utilizando técnicas de minería de datos, en este caso datos obtenidos por sensores ubicados a lo largo de la costa de Estados Unidos, que registran datos como temperatura, velocidad del viento, etc..

Como forma de ampliar la información de los productos, existe la posibilidad de utilizar "estanterías inteligentes" [CCW10] dotadas de sensores capaces de manejar datos más complejos y dinámicos.

Figura 2.2: Estanterías inteligentes (tomado de [CCW10])

Resulta necesario mencionar también los sistemas de fabricación holónicos ([JJM03, NFB01]) basados en la idea [vBS74] de que un sistema complejo se puede lograr a partir de estructuras simples si existen formas intermedias estables. Los sistemas de fabricación holónicos no son ni un sinónimo ni representan una alternativa a los sistemas multiagente, si no que los complementan. Mientras que los sistemas de control basados en agentes representan solamente entornos software, un sistema holónico engloba los aspectos tanto físicos como basados en información, del entorno de fabricación.

2.3. Actualización dinámica de ontologías desde bases de datos relacionales

La necesidad de relacionar ontologías con bases de datos relacionales es abordada en muchos casos [KSM08, CS05] como un problema relacionado con Internet, debido a la gran cantidad de bases de datos existentes sobre los que sólo se pueden hacer búsquedas basadas en bases de datos relacionales típicas.

Una de las soluciones propuestas consiste en definir, con un lenguaje como D2RQ un archivo donde se relacionan valores de las distintas tablas de la base de datos con

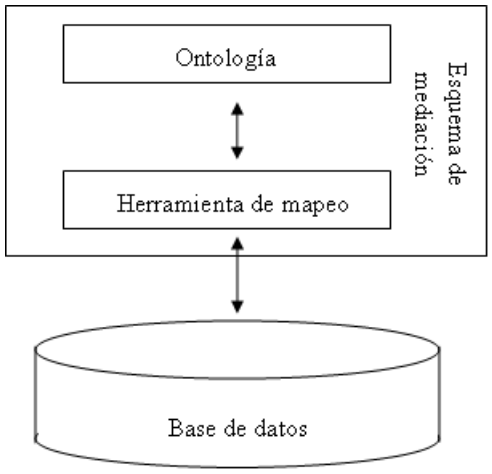

Figura 2.3: Mapeo entre base de datos y ontología

las clases de la ontología. D2R Server[2] es una de las herramientas que nos permite realizar esta tarea. Sin embargo, tiene algunas limitaciones como la imposibilidad de realizar acciones como CREATE, DELETE o UPDATE o la imposibilidad de integrar múltiples bases de datos.

Además, estas herramientas están sobre todo orientadas a posibilitar una navegación mediante web semántica sobre páginas web que se alimentan de bases de datos relacionales, por lo que aunque se han analizado para el presente trabajo no se han considerado apropiadas para su desarrollo.

Olivier Curé y Raphaël Squelbut proponen en su trabajo [CS05] un enfoque diferente que ha servido de base principal para este proyecto. En él proponen la utilización de disparadores (ver 3.1.5) para mantener actualizada una ontología respecto a su base de datos.

El trabajo se basa a su vez en uno anterior de uno de los autores [Cur] en el que se presenta el sistema DBOM , una aplicación, independiente del dominio, que permite la integración de datos y servicios de mantenimiento en un entorno de web semántica. Se analiza cómo las técnicas habituales implican un trabajo de ingeniería inversa al tener que identificar los componentes del sistema y sus relaciones, y se explica que el sistema DBOM utiliza esta misma aproximación, añadiendo la capacidad de actualizar la ontología mediante disparadores.

[2]http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

Capítulo 3

Hiridenda

3.1. Planteamiento de la solución

La solución adoptada se caracteriza por estar formada por diversas piezas desarrolladas independientemente que se unen al final para dar la forma a la estructura diseñada. Por lo tanto, como requisito imprescindible para comprender la solución completa, se va a proceder a explicar primero cada una de las piezas de forma autónoma, para pasar a continuación a mostrar la estructura final.

3.1.1. Protégé

Protégé[1] es una aplicación, desarrollada en la universidad de Stanford, para crear y editar ontologías. Mediante una interfaz gráfica, esta herramienta de código abierto permite realizar tareas comunes como crear clases, entidades, definir las relaciones entre los distintos miembros de la ontología, asignar propiedades de objetos, de datos, etc. También brinda la posibilidad de que el sistema realiza inferencias de forma que se detecten nuevas relaciones o pertenencias a clases. Además, mediante la instalación de diferentes *plug-ins* permite visualizar su estructura de una forma gráfica para facilitar su presentación.

[1] http://protege.stanford.edu/

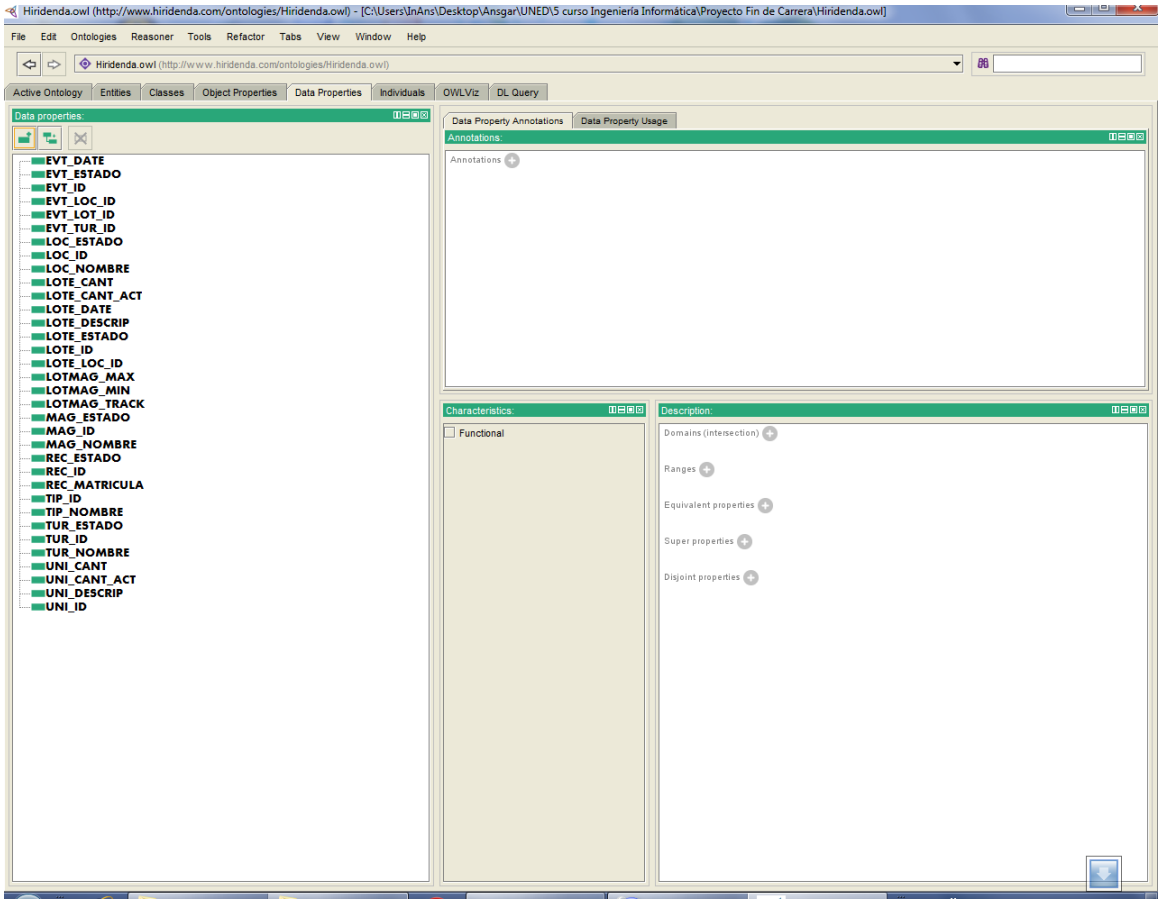

Figura 3.1: Ventana principal de Protégé

Una vez definida la ontología que se desee, permite almacenarla en distintos tipos de archivo, siendo el más habitual el formato OWL.

3.1.2. Ontología

Tras estudiar el funcionamiento del tipo de almacén en el que se quiere implementar el sistema en desarrollo, y con la ayuda de otro departamento de la empresa I3B que ya ha trabajado anteriormente con aplicaciones de logística, se ha definido la siguiente ontología básica para llevar a cabo una primera versión del proyecto.

Esta ontología, al no estar aplicada a ningún almacén concreto, podrá ser ampliada y adaptada a las necesidades que surjan cuando el proyecto se implemente en un entorno conocido.

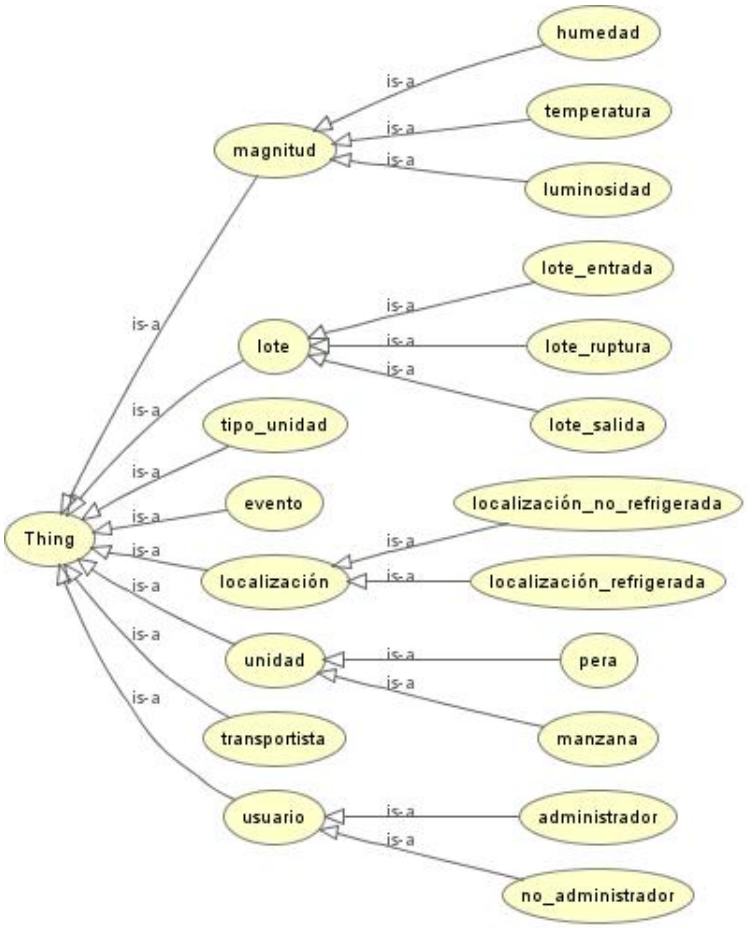

Figura 3.2: Ontología de un almacén

Como es habitual en las ontologías creadas con Protégé, la raíz de todas las clases es Thing, por lo tanto, todas las clases son en última instancia una cosa.

A continuación se hace una breve explicación de cada una de las clases:

- magnitud: en esta clase se agrupan todas las magnitudes físicas que puedan ser medidas por los sensores utilizados. En este caso tenemos tres tipos de magnitud.
  - humedad
  - temperatura
  - luminosidad
- lote: un lote está formado por varias cajas que contienen unidades. Estas unidades pueden ser de tipos distintos.

- lote_entrada: es el lote que se crea en la recepción.
- lote_ruptura: cuando se separan las unidades que forman un lote, se crean lotes de ruptura.
- lote_salida: es el lote que sale del almacén.

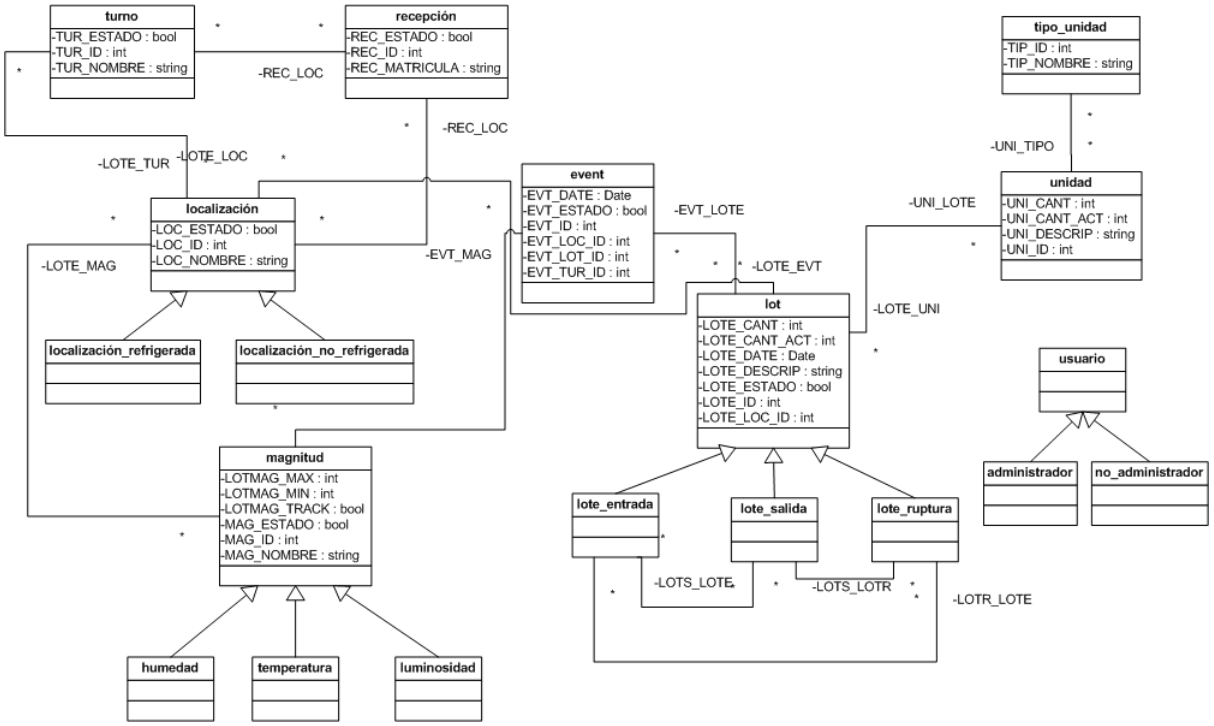

Figura 3.3: Diagrama de la ontología

- tipo_unidad: en esta clase aún por expandir, se definen todos los tipos habituales que se usen en la actividad diaria del almacén, como productos perecederos o no, frutas blandas, conservas, etc.
- evento: recoge todos los sucesos fuera del funcionamiento normal, que generan alarmas.
- localización: las distintas localizaciones que hay en el almacén.
  - localización_no_refrigerada
  - localización_refrigerada
- unidad: esta clase define todos las unidades específicas que existen en un almacén, como zapatilla Converse del número 45.

    - pera: clase de ejemplo que agrupa a todas las unidades que son peras. Se deben definir subclases con la marca y tipo concretos.
    - manzana

- transportista: identifica al transportista que trae o se lleva un lote.
- usuario: identifica a un usuario del sistema del almacén.
    - administrador: subclase donde se agrupan los usuarios administradores.
    - no_administrador: subclase donde se agrupan los usuarios no administradores.

3.1.3. Jena

La herramienta Jena[2] es un marco que permite crear aplicaciones que trabajen con web semántica u ontologías. Actúa como intermediario facilitando las tareas habituales a la hora de usar ontologías, como la lectura, inclusión/eliminación de entidades o instancias, agregación de atributos o la iteración sobre todos los elementos que contiene la ontología.

Jena trabaja con ontologías que pueden encontrarse bien en archivos, en memoria principal o en bases de datos relacionales. En este proyecto se ha decidido obtener la estructura de la ontología, al ser un elemento poco cambiante, desde un archivo. En cambio, debido a la naturaleza dinámica de los datos, como las instancias, estos cambios se almacenan en una base de datos relacional que Jena se encarga de crear y definir.

Aunque en un principio se contempló la posibilidad de, una vez creada la base de datos de la ontología mediante Jena, modificar su contenido directamente a través de disparadores (ver 3.1.5), pronto se descartó esta opción ya que la estructura de la base de datos, sin una lógica aparente, hace imposible esta tarea.

Jena también permite realizar consultas SPARQL para obtener información de la ontología, y además contiene un motor de inferencias basado en reglas.

[2] http://jena.sourceforge.net/

3.1.4. Procedimientos almacenados

Los procedimientos almacenados son una tecnología que permite que un programa se almacene físicamente en una base de datos. Este programa puede así ser utilizado desde la propia base de datos para realizar modificaciones, cálculos, etc. ya que tiene acceso directo a los datos.

Aunque existen procedimientos almacenados de distintas empresas, para este proyecto se ha utilizado la opción que proporciona la compañía Oracle. En este caso, las funciones que se definen en la base de datos deben estar escritas en el lenguaje de programación Java, por lo que las bases de datos compatibles con esta opción contienen lo que se conoce como máquina virtual de Java (JVM).

Una vez que se han definido los archivos Java necesarios en el entorno de desarrollo seleccionado, estos se cargan en la base de datos a través de comandos sql o de un entorno gráfico de gestión de bases de datos como sqldeveloper[3].

Los métodos que se definan pueden ser de cualquier tipo, excepto lógicamente aquéllos que utilicen funciones de interfaz gráfica [Ora].

Las acciones que pueden ser llevadas a cabo por procedimientos almacenados están restringidas por la seguridad de Java, por lo que se deben dar explícitamente permisos para los accesos autorizados como conexiones de red o a archivos.

3.1.5. Disparadores

Un disparador es una función definida en una base de datos que se ejecuta cuando se cumple cierta condición especificada en su definición. Así, podemos tener disparadores que se activan cuando insertamos valores en un determinado campo de una tabla, cuando se eliminan entradas, o incluso cuando los datos que se manejan sean mayores a un valor que nos interese.

Un ejemplo típico es la actualización del salario de los empleados de una empresa cuando se modifica el valor del IPC (Índice de Precios al Consumo), lo que evita tener que realizar el cambio para cada empleado individualmente, con el trabajo y

[3]http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html

las posibilidades de errores que conllevaría.

En Oracle, los disparadores pueden ser utilizados para ejecutar procedimientos almacenados, posibilidad que ha sido ampliamente usada en el presente proyecto.

3.1.6. Invocación de métodos remotos

La invocación de métodos remotos, más conocida como RMI por sus siglas en inglés (Remote Method Invocation), es "una extensión de la invocación a métodos locales que permite que un objeto que vive en un proceso invoque los métodos de un objeto que reside en otro proceso." [CD05]

Aunque la definición habla de procesos distintos, de esta forma se posibilita que una aplicación que se está ejecutando en un equipo llame a un método que reside en otro equipo.

Para implementar este esquema, uno de los equipos actúa como servidor, ofreciendo en un puerto concreto un servicio, que será invocado por el cliente. Ambos equipos pueden intercambiar su papel en determinados momentos cuando sea el servidor el que necesite hacer uso de uno de los servicios que residen en el equipo cliente.

3.1.7. Arquitectura completa

En el siguiente esquema se puede visualizar cómo es el sistema completo una vez se han ensamblado todas sus piezas:

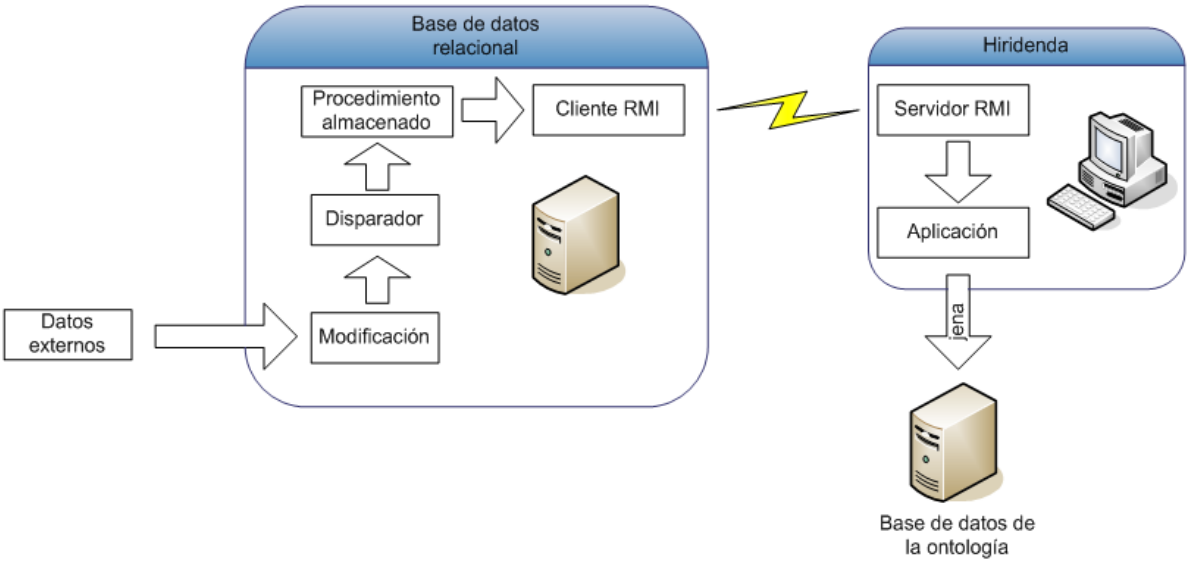

Figura 3.4: Arquitectura completa del sistema

El funcionamiento de la aplicación consiste en los siguientes pasos:

1. Los valores de la base de datos son modificados por una aplicación externa encargada de recoger la información del almacén, productos, transportistas, localizaciones ...

2. Estas modificaciones hacen que, por cada tabla y elemento modificado, se ejecute un disparador que llama a su vez a un procedimiento almacenado.

3. El procedimiento almacenado, escrito en lenguaje Java, llama al cliente RMI que implementa la tecnología Java RMI.

4. El cliente RMI envía los datos modificados al servidor RMI que se encuentra en otro equipo.

5. El servidor RMI le pasa la información recibida a la aplicación Hiridenda.

6. La aplicación, utilizando la API de Jena realiza las actualizaciones necesarias en la ontología, que serán reflejadas en su base de datos.

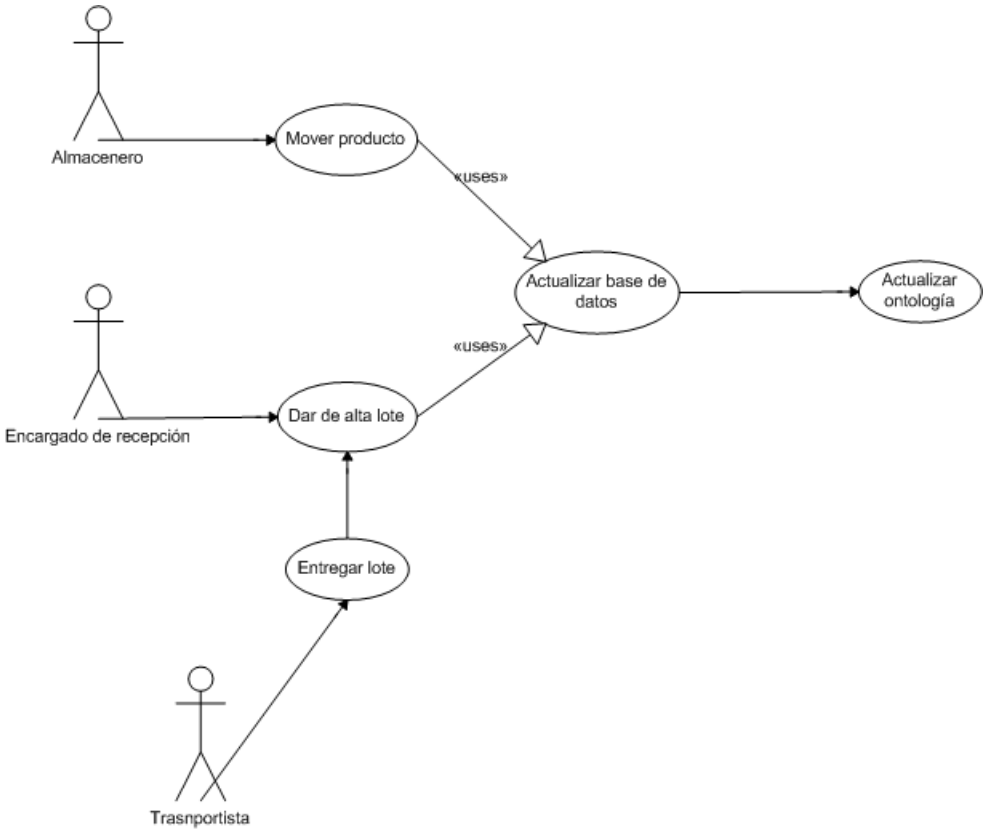

Figura 3.5: Caso de uso de ejemplo

3.1.8. Descripción de la aplicación

Para realizar las pruebas necesarias de cara a comprobar el funcionamiento correcto del esquema descrito, se ha desarrollado una pequeña aplicación de test, que contiene toda la estructura que deberá ser insertada en la aplicación principal que se desarrolle en un futuro.

Parámetros como los datos de conexión, el nombre de la base de datos, etc. han sido introducidos dentro del código fuente, por lo que cualquier cambio supone una nueva compilación. Esta forma de trabajo se sustituirá en la aplicación por la forma que se estime más conveniente, como guardar los datos en un archivo especial o configurar las conexiones mediante una opción del menú.

Todos los mensajes, tanto de error como informativos, son mostrados en la ventana principal de la aplicación.

3.1.8.1. Paquete database

En este paquete se encuentran las clases que sirven para comunicarse con cada una de las tablas de la base de datos (ver 3.1.10). Existe una función para obtener y definir el valor de cada columna. Además, se facilitan las consultas SQL habituales para obtener todas las instancias, o sólo aquellas que cumplen unas condiciones concretas.

Los archivos, uno por cada clase java, son:

TEvento.java

- Constructores
  - TEvento()
  - TEvento(java.math.BigDecimal evtId)
  - TEvento(java.math.BigDecimal evtId, java.lang.String evtLotId, java.math.BigInteger evtEstado, java.util.Date evtDate)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.util.Date getEvtDate()
  - java.math.BigInteger getEvtEstado()
  - java.math.BigDecimal getEvtId()
  - TLocalizacion getEvtLocId()
  - java.lang.String getEvtLotId()
  - TTurno getEvtTurId()
  - int hashCode()
  - void setEvtDate(java.util.Date evtDate)
  - void setEvtEstado(java.math.BigInteger evtEstado)

- void setEvtId(java.math.BigDecimal evtId)
- void setEvtLocId(TLocalizacion evtLocId)
- void setEvtLotId(java.lang.String evtLotId)
- void setEvtTurId(TTurno evtTurId)
- java.lang.String toString()

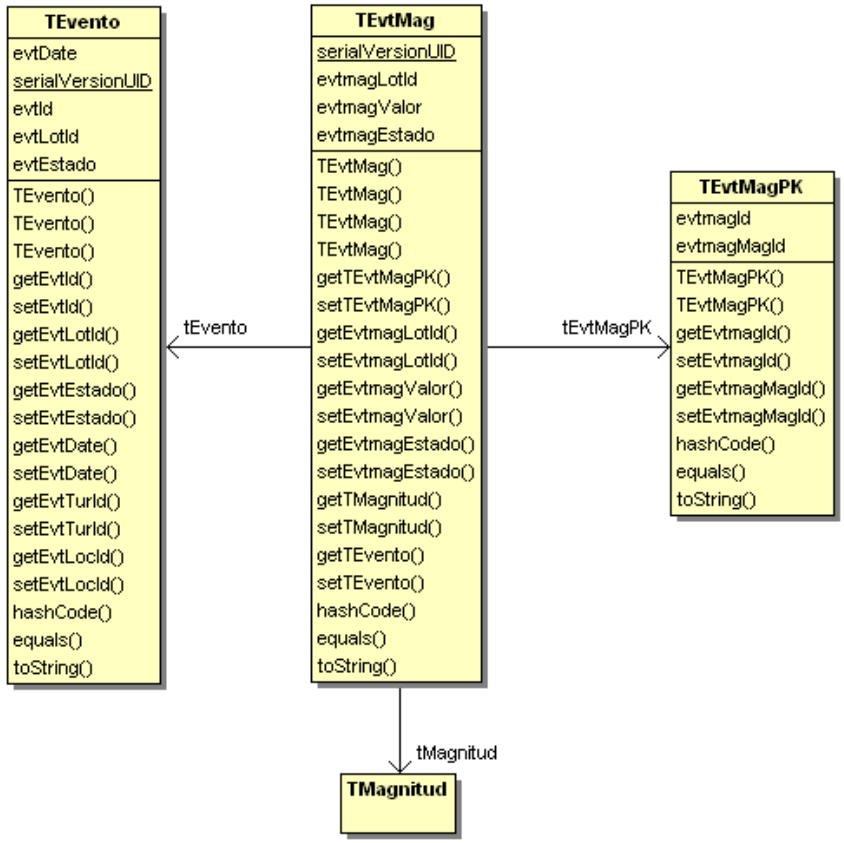

Figura 3.6: Diagrama de la clase TEvento

TEvtMag.java

- Constructores
  - TEvtMag()
  - TEvtMag(java.math.BigInteger evtmagId, java.math.BigInteger evtmagMagId)
  - TEvtMag(TEvtMagPK tEvtMagPK)

- TEvtMag(TEvtMagPK tEvtMagPK, java.lang.String evtmagLotId, java.math.BigInteger evtmagValor, java.math.BigInteger evtmagEstado)

- Métodos

  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getEvtmagEstado()
  - java.lang.String getEvtmagLotId()
  - java.math.BigInteger getEvtmagValor()
  - TEvento getTEvento()
  - TEvtMagPK getTEvtMagPK()
  - TMagnitud getTMagnitud()
  - int hashCode()
  - void setEvtmagEstado(java.math.BigInteger evtmagEstado)
  - void setEvtmagLotId(java.lang.String evtmagLotId)
  - void setEvtmagValor(java.math.BigInteger evtmagValor)
  - void setTEvento(TEvento tEvento)
  - void setTEvtMagPK(TEvtMagPK tEvtMagPK)
  - void setTMagnitud(TMagnitud tMagnitud)
  - java.lang.String toString()

TEvtMagPK.java

- Constructores

  - TEvtMagPK()
  - TEvtMagPK(java.math.BigInteger evtmagId, java.math.BigInteger evtmagMagId)

- Métodos

- boolean equals(java.lang.Object object)
- java.math.BigInteger getEvtmagId()
- java.math.BigInteger getEvtmagMagId()
- int hashCode()
- void setEvtmagId(java.math.BigInteger evtmagId)
- void setEvtmagMagId(java.math.BigInteger evtmagMagId)
- java.lang.String toString()

TLocalizacion.java

- Constructores
  - TLocalizacion()
  - TLocalizacion(java.math.BigDecimal locId)
  - TLocalizacion(java.math.BigDecimal locId, java.lang.String locNombre, java.math.BigInteger locEstado)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.util.Date getLocDateEnd()
  - java.lang.String getLocDescrip()
  - java.math.BigInteger getLocEstado()
  - java.math.BigDecimal getLocId()
  - java.lang.String getLocNombre()
  - java.lang.String getLocX()
  - java.lang.String getLocY()
  - int hashCode()
  - void setLocDateEnd(java.util.Date locDateEnd)

- void setLocDescrip(java.lang.String locDescrip)
- void setLocEstado(java.math.BigInteger locEstado)
- void setLocId(java.math.BigDecimal locId)
- void setLocNombre(java.lang.String locNombre)
- void setLocX(java.lang.String locX)
- void setLocY(java.lang.String locY)
- java.lang.String toString()

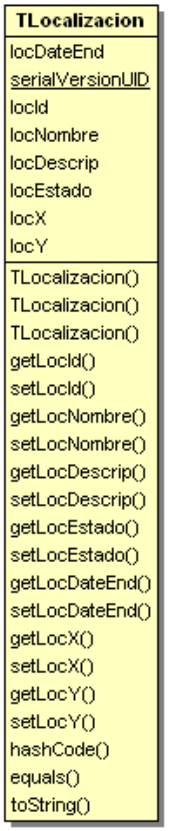

Figura 3.7: Diagrama de la clase TLocalizacion

TLoteEnt.java

- Constructores
  - TLoteEnt()
  - TLoteEnt(java.lang.String loteId)

- TLoteEnt(java.lang.String loteId, java.util.Date loteDate, java.math.BigInteger loteCant, java.math.BigInteger loteCantAct, java.math.BigInteger loteContador, java.math.BigInteger loteEstado)

- Métodos

  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLoteCant()
  - java.math.BigInteger getLoteCantAct()
  - java.math.BigInteger getLoteContador()
  - java.util.Date getLoteDate()
  - java.util.Date getLoteDateEnd()
  - java.lang.String getLoteDescrip()
  - java.math.BigInteger getLoteEstado()
  - java.lang.String getLoteId()
  - TLocalizacion getLoteLocId()
  - TTag getLoteTagId()
  - TTurno getLoteTurId()
  - TTurno getLoteTurIdEnd()
  - int hashCode()
  - void setLoteCant(java.math.BigInteger loteCant)
  - void setLoteCantAct(java.math.BigInteger loteCantAct)
  - void setLoteContador(java.math.BigInteger loteContador)
  - void setLoteDate(java.util.Date loteDate)
  - void setLoteDateEnd(java.util.Date loteDateEnd)
  - void setLoteDescrip(java.lang.String loteDescrip)
  - void setLoteEstado(java.math.BigInteger loteEstado)

- void setLoteId(java.lang.String loteId)
- void setLoteLocId(TLocalizacion loteLocId)
- void setLoteTagId(TTag loteTagId)
- void setLoteTurId(TTurno loteTurId)
- void setLoteTurIdEnd(TTurno loteTurIdEnd)
- java.lang.String toString()

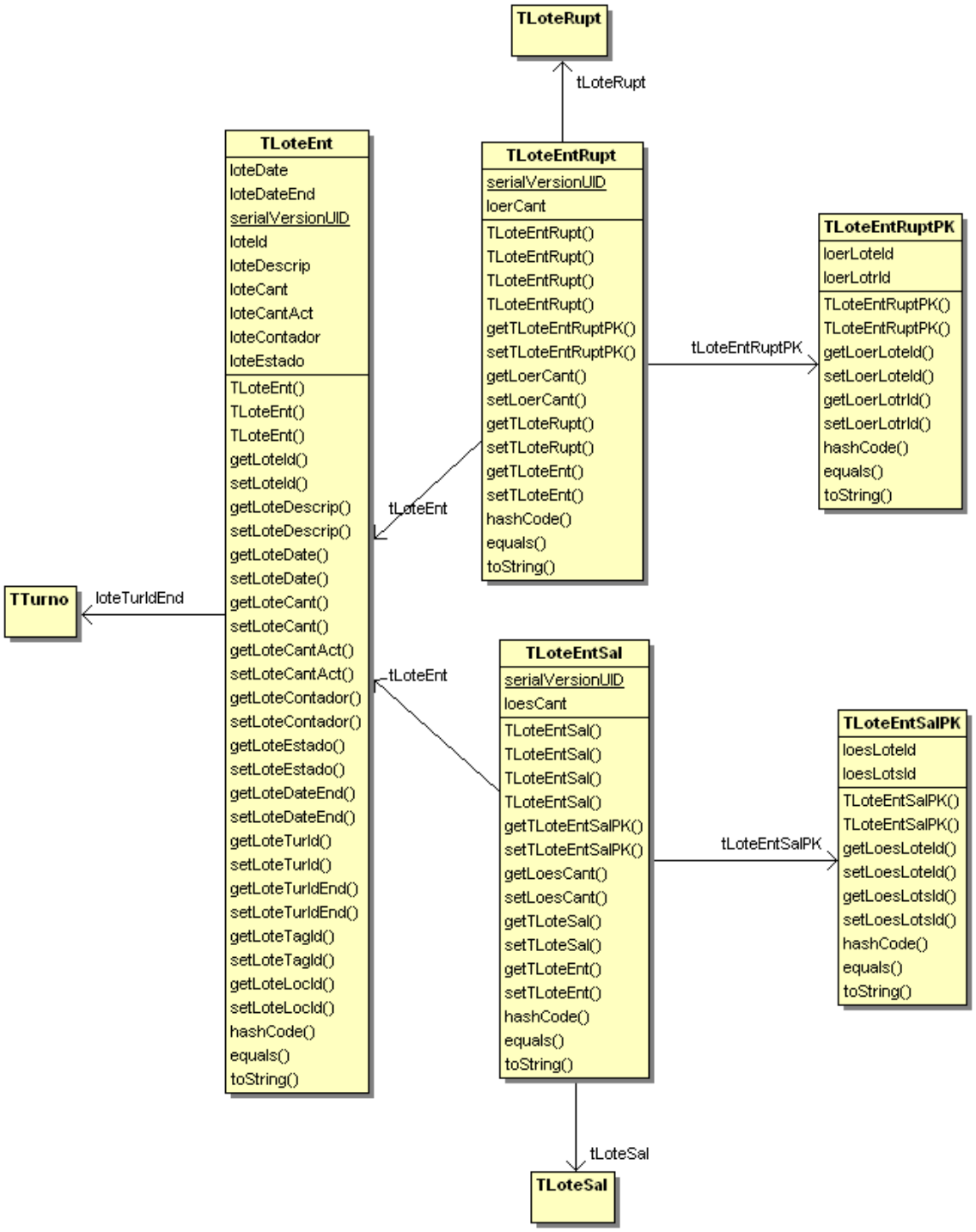

Figura 3.8: Diagrama de la clase TLoteEnt

TLoteEntRupt.java

- Constructores
  - TLoteEntRupt()

- TLoteEntRupt(java.lang.String loerLoteId, java.lang.String loerLotrId)
- TLoteEntRupt(TLoteEntRuptPK tLoteEntRuptPK)
- TLoteEntRupt(TLoteEntRuptPK tLoteEntRuptPK, java.math.BigInteger loerCant)

- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLoerCant()
  - TLoteEnt getTLoteEnt()
  - TLoteEntRuptPK getTLoteEntRuptPK()
  - TLoteRupt getTLoteRupt()
  - int hashCode()
  - void setLoerCant(java.math.BigInteger loerCant)
  - void setTLoteEnt(TLoteEnt tLoteEnt)
  - void setTLoteEntRuptPK(TLoteEntRuptPK tLoteEntRuptPK)
  - void setTLoteRupt(TLoteRupt tLoteRupt)
  - java.lang.String toString()

TLoteEntRuptPK.java

- Constructores
  - TLoteEntRuptPK()
  - TLoteEntRuptPK(java.lang.String loerLoteId, java.lang.String loerLotrId)

- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLoerLoteId()

- java.lang.String getLoerLotrId()
- int hashCode()
- void setLoerLoteId(java.lang.String loerLoteId)
- void setLoerLotrId(java.lang.String loerLotrId)
- java.lang.String toString()

TLoteEntSal.java

- Constructores
  - TLoteEntSal()
  - TLoteEntSal(java.lang.String loesLoteId, java.lang.String loesLotsId)
  - TLoteEntSal(TLoteEntSalPK tLoteEntSalPK)
  - TLoteEntSal(TLoteEntSalPK tLoteEntSalPK, java.math.BigInteger loesCant)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLoesCant()
  - TLoteEnt getTLoteEnt()
  - TLoteEntSalPK getTLoteEntSalPK()
  - TLoteSal getTLoteSal()
  - int hashCode()
  - void setLoesCant(java.math.BigInteger loesCant)
  - void setTLoteEnt(TLoteEnt tLoteEnt)
  - void setTLoteEntSalPK(TLoteEntSalPK tLoteEntSalPK)
  - void setTLoteSal(TLoteSal tLoteSal)
  - java.lang.String toString()

TLoteEntSalPK.java

- Constructores
  - TLoteEntSalPK()
  - TLoteEntSalPK(java.lang.String loesLoteId, java.lang.String loesLotsId)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLoesLoteId()
  - java.lang.String getLoesLotsId()
  - int hashCode()
  - void setLoesLoteId(java.lang.String loesLoteId)
  - void setLoesLotsId(java.lang.String loesLotsId)
  - java.lang.String toString()

TLoteMag.java

- Constructores
  - TLoteMag()
  - TLoteMag(java.lang.String lotmagLoteId, java.math.BigInteger lotmagMagId)
  - TLoteMag(TLoteMagPK tLoteMagPK)
  - TLoteMag(TLoteMagPK tLoteMagPK, java.math.BigInteger lotmagTrack)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLotmagMax()
  - java.math.BigInteger getLotmagMin()

- java.math.BigInteger getLotmagTrack()
- TLoteEnt getTLoteEnt()
- TLoteMagPK getTLoteMagPK()
- TMagnitud getTMagnitud()
- int hashCode()
- void setLotmagMax(java.math.BigInteger lotmagMax)
- void setLotmagMin(java.math.BigInteger lotmagMin)
- void setLotmagTrack(java.math.BigInteger lotmagTrack)
- void setTLoteEnt(TLoteEnt tLoteEnt)
- void setTLoteMagPK(TLoteMagPK tLoteMagPK)
- void setTMagnitud(TMagnitud tMagnitud)
- java.lang.String toString()

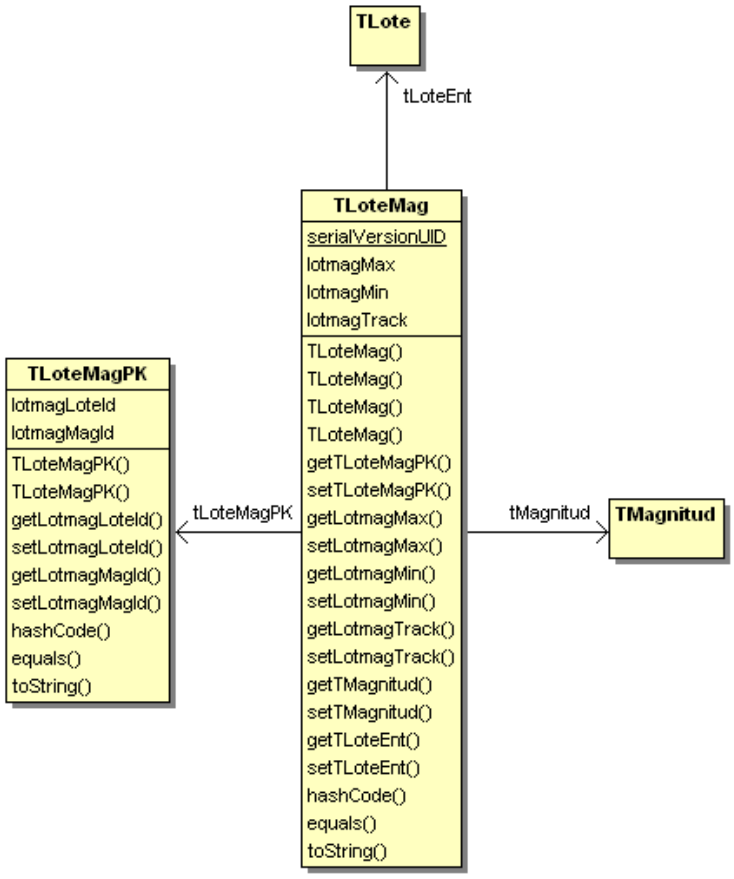

Figura 3.9: Diagrama de la clase TLoteMag

TLoteMagPK.java

- Constructores
  - TLoteMagPK()
  - TLoteMagPK(java.lang.String lotmagLoteId, java.math.BigInteger lotmagMagId)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLotmagLoteId()
  - java.math.BigInteger getLotmagMagId()
  - int hashCode()

- void setLotmagLoteId(java.lang.String lotmagLoteId)
- void setLotmagMagId(java.math.BigInteger lotmagMagId)
- java.lang.String toString()

TLoteRupt.java

- Constructores
  - TLoteRupt()
  - TLoteRupt(java.lang.String lotrId)
  - TLoteRupt(java.lang.String lotrId, java.util.Date lotrDate, java.math.BigInteger lotrCant, java.math.BigInteger lotrCantAct, java.math.BigInteger lotrContador, java.math.BigInteger lotrEstado)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLotrCant()
  - java.math.BigInteger getLotrCantAct()
  - java.math.BigInteger getLotrContador()
  - java.util.Date getLotrDate()
  - java.util.Date getLotrDateEnd()
  - java.lang.String getLotrDescrip()
  - java.math.BigInteger getLotrEstado()
  - java.lang.String getLotrId()
  - TLocalizacion getLotrLocId()
  - TTag getLotrTagId()
  - TTurno getLotrTurId()
  - TTurno getLotrTurIdEnd()

- int hashCode()
- void setLotrCant(java.math.BigInteger lotrCant)
- void setLotrCantAct(java.math.BigInteger lotrCantAct)
- void setLotrContador(java.math.BigInteger lotrContador)
- void setLotrDate(java.util.Date lotrDate)
- void setLotrDateEnd(java.util.Date lotrDateEnd)
- void setLotrDescrip(java.lang.String lotrDescrip)
- void setLotrEstado(java.math.BigInteger lotrEstado)
- void setLotrId(java.lang.String lotrId)
- void setLotrLocId(TLocalizacion lotrLocId)
- void setLotrTagId(TTag lotrTagId)
- void setLotrTurId(TTurno lotrTurId)
- void setLotrTurIdEnd(TTurno lotrTurIdEnd)
- java.lang.String toString()

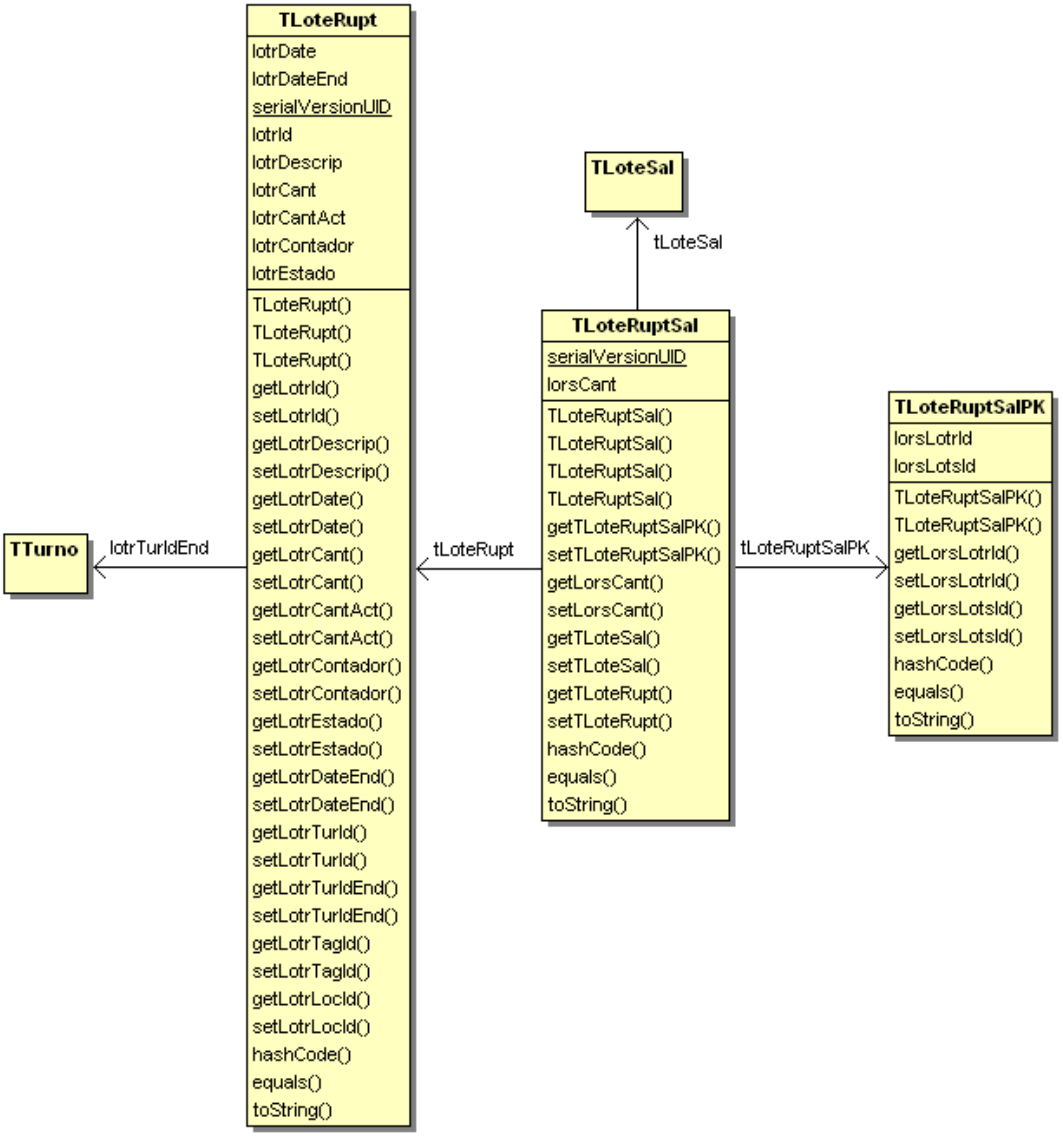

Figura 3.10: .Diagrama de la clase TLoteRupt

TLoteRuptSal.java

- Constructores
  - TLoteRuptSal()
  - TLoteRuptSal(java.lang.String lorsLotrId, java.lang.String lorsLotsId)
  - TLoteRuptSal(TLoteRuptSalPK tLoteRuptSalPK)
  - TLoteRuptSal(TLoteRuptSalPK tLoteRuptSalPK, java.math.BigInteger lorsCant)

- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLorsCant()
  - TLoteRupt getTLoteRupt()
  - TLoteRuptSalPK getTLoteRuptSalPK()
  - TLoteSal getTLoteSal()
  - int hashCode()
  - void setLorsCant(java.math.BigInteger lorsCant)
  - void setTLoteRupt(TLoteRupt tLoteRupt)
  - void setTLoteRuptSalPK(TLoteRuptSalPK tLoteRuptSalPK)
  - void setTLoteSal(TLoteSal tLoteSal)
  - java.lang.String toString()

TLoteRuptSalPK.java

- Constructores
  - TLoteRuptSalPK()
  - TLoteRuptSalPK(java.lang.String lorsLotrId, java.lang.String lorsLotsId)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLorsLotrId()
  - java.lang.String getLorsLotsId()
  - int hashCode()
  - void setLorsLotrId(java.lang.String lorsLotrId)
  - void setLorsLotsId(java.lang.String lorsLotsId)
  - java.lang.String toString()

TLoteSal.java

- Constructores
  - TLoteSal()
  - TLoteSal(java.lang.String lotsId)
  - TLoteSal(java.lang.String lotsId, java.util.Date lotsDate, java.math.BigInteger lotsCant, java.math.BigInteger lotsCantAct, java.math.BigInteger lotsContador, java.math.BigInteger lotsEstado)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLotsCant()
  - java.math.BigInteger getLotsCantAct()
  - java.math.BigInteger getLotsContador()
  - java.util.Date getLotsDate()
  - java.util.Date getLotsDateEnd()
  - java.lang.String getLotsDescrip()
  - java.math.BigInteger getLotsEstado()
  - java.lang.String getLotsId()
  - TLocalizacion getLotsLocId()
  - TTag getLotsTagId()
  - TTurno getLotsTurId()
  - TTurno getLotsTurIdEnd()
  - int hashCode()
  - void setLotsCant(java.math.BigInteger lotsCant)
  - void setLotsCantAct(java.math.BigInteger lotsCantAct)

- void setLotsContador(java.math.BigInteger lotsContador)
- void setLotsDate(java.util.Date lotsDate)
- void setLotsDateEnd(java.util.Date lotsDateEnd)
- void setLotsDescrip(java.lang.String lotsDescrip)
- void setLotsEstado(java.math.BigInteger lotsEstado)
- void setLotsId(java.lang.String lotsId)
- void setLotsLocId(TLocalizacion lotsLocId)
- void setLotsTagId(TTag lotsTagId)
- void setLotsTurId(TTurno lotsTurId)
- void setLotsTurIdEnd(TTurno lotsTurIdEnd)
- java.lang.String toString()

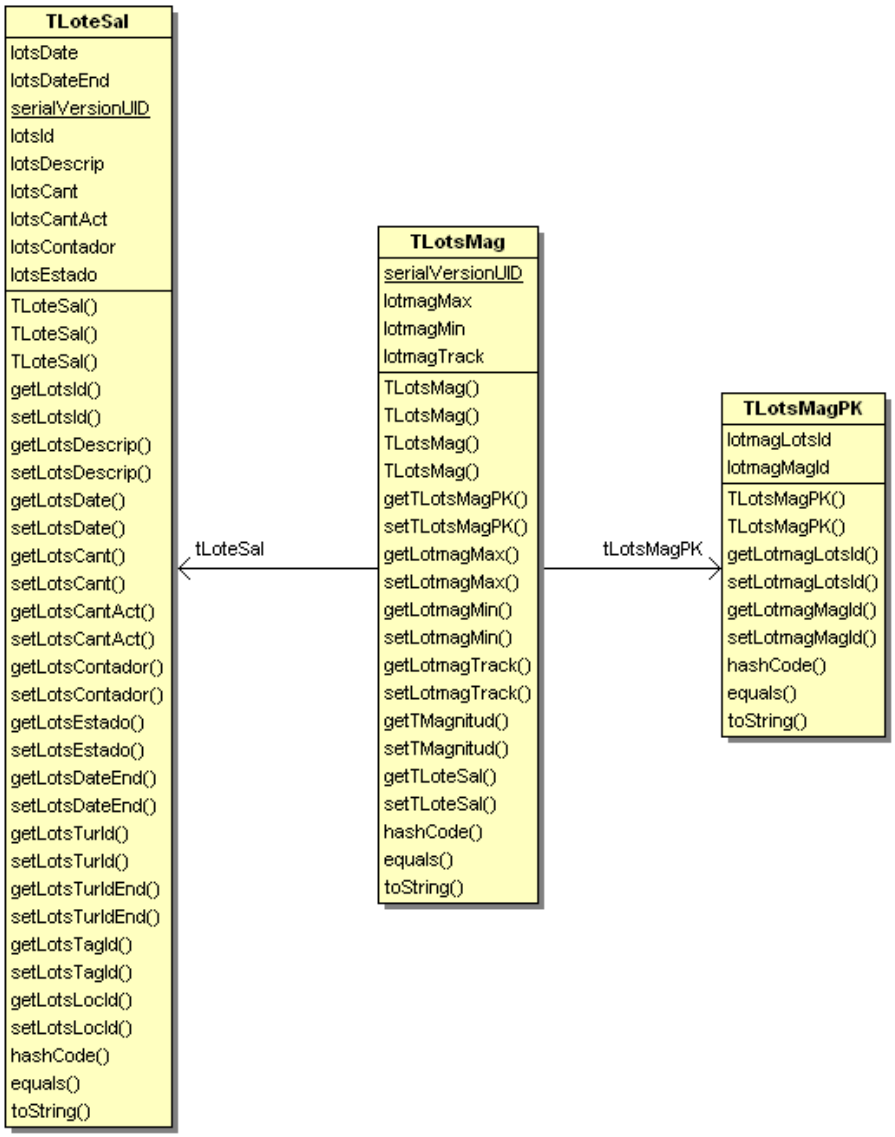

Figura 3.11: .Diagrama de la clase TLoteSal

TLoteUni.java

- Constructores
  - TLoteUni()
  - TLoteUni(java.lang.String lotuniUniId, java.lang.String lotuniLoteId)
  - TLoteUni(TLoteUniPK tLoteUniPK)
  - TLoteUni(TLoteUniPK tLoteUniPK, java.math.BigInteger lotuniCant)
- Métodos

- boolean equals(java.lang.Object object)
- java.math.BigInteger getLotuniCant()
- TLoteEnt getTLoteEnt()
- TLoteUniPK getTLoteUniPK()
- TUnidad getTUnidad()
- int hashCode()
- void setLotuniCant(java.math.BigInteger lotuniCant)
- void setTLoteEnt(TLoteEnt tLoteEnt)
- void setTLoteUniPK(TLoteUniPK tLoteUniPK)
- void setTUnidad(TUnidad tUnidad)
- java.lang.String toString()

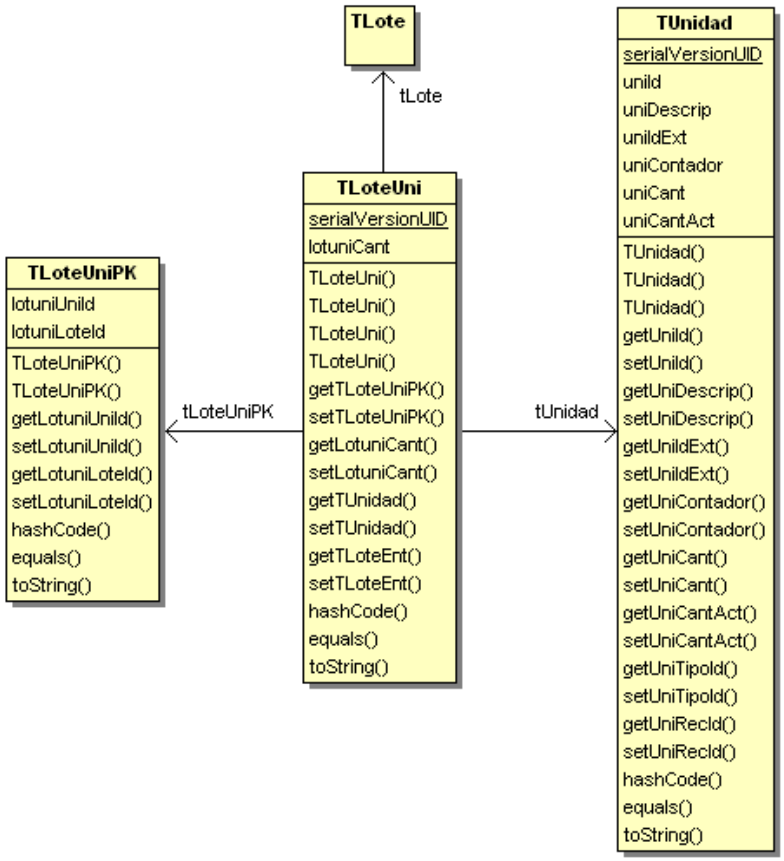

Figura 3.12: Diagrama de la clase TLoteUni

TLoteUniPK.java

- Constructores
  - TLoteUniPK()
  - TLoteUniPK(java.lang.String lotuniUniId, java.lang.String lotuniLoteId)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLotuniLoteId()
  - java.lang.String getLotuniUniId()
  - int hashCode()
  - void setLotuniLoteId(java.lang.String lotuniLoteId)
  - void setLotuniUniId(java.lang.String lotuniUniId)
  - java.lang.String toString()

TLotrMag.java

- Constructores
  - TLotrMag()
  - TLotrMag(java.lang.String lotmagLotrId, java.math.BigInteger lotmagMagId)
  - TLotrMag(TLotrMagPK tLotrMagPK)
  - TLotrMag(TLotrMagPK tLotrMagPK, java.math.BigInteger lotmagTrack)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLotmagMax()
  - java.math.BigInteger getLotmagMin()

- java.math.BigInteger getLotmagTrack()
- TLoteRupt getTLoteRupt()
- TLotrMagPK getTLotrMagPK()
- TMagnitud getTMagnitud()
- int hashCode()
- void setLotmagMax(java.math.BigInteger lotmagMax)
- void setLotmagMin(java.math.BigInteger lotmagMin)
- void setLotmagTrack(java.math.BigInteger lotmagTrack)
- void setTLoteRupt(TLoteRupt tLoteRupt)
- void setTLotrMagPK(TLotrMagPK tLotrMagPK)
- void setTMagnitud(TMagnitud tMagnitud)
- java.lang.String toString()

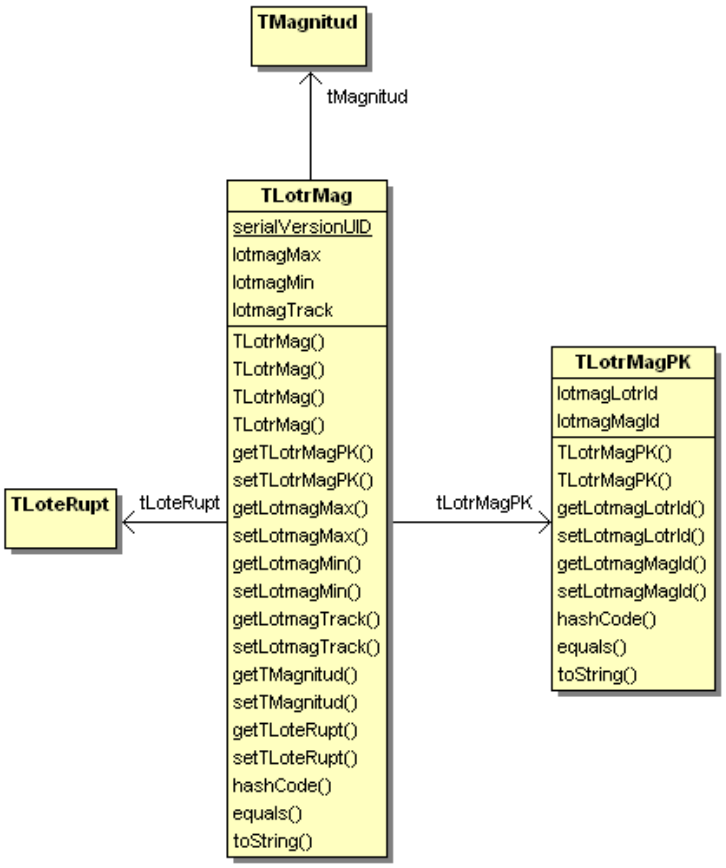

Figura 3.13: Diagrama de la clase TLotrMag

TLotrMagPK.java

- Constructores
  - TLotrMagPK()
  - TLotrMagPK(java.lang.String lotmagLotrId, java.math.BigInteger lotmagMagId)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLotmagLotrId()
  - java.math.BigInteger getLotmagMagId()
  - int hashCode()
  - void setLotmagLotrId(java.lang.String lotmagLotrId)
  - void setLotmagMagId(java.math.BigInteger lotmagMagId)
  - java.lang.String toString()

TLotsMag.java

- Constructores
  - TLotsMag()
  - TLotsMag(java.lang.String lotmagLotsId, java.math.BigInteger lotmagMagId)
  - TLotsMag(TLotsMagPK tLotsMagPK)
  - TLotsMag(TLotsMagPK tLotsMagPK, java.math.BigInteger lotmagTrack)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getLotmagMax()

- java.math.BigInteger getLotmagMin()
- java.math.BigInteger getLotmagTrack()
- TLoteSal getTLoteSal()
- TLotsMagPK getTLotsMagPK()
- TMagnitud getTMagnitud()
- int hashCode()
- void setLotmagMax(java.math.BigInteger lotmagMax)
- void setLotmagMin(java.math.BigInteger lotmagMin)
- void setLotmagTrack(java.math.BigInteger lotmagTrack)
- void setTLoteSal(TLoteSal tLoteSal)
- void setTLotsMagPK(TLotsMagPK tLotsMagPK)
- void setTMagnitud(TMagnitud tMagnitud)
- java.lang.String toString()

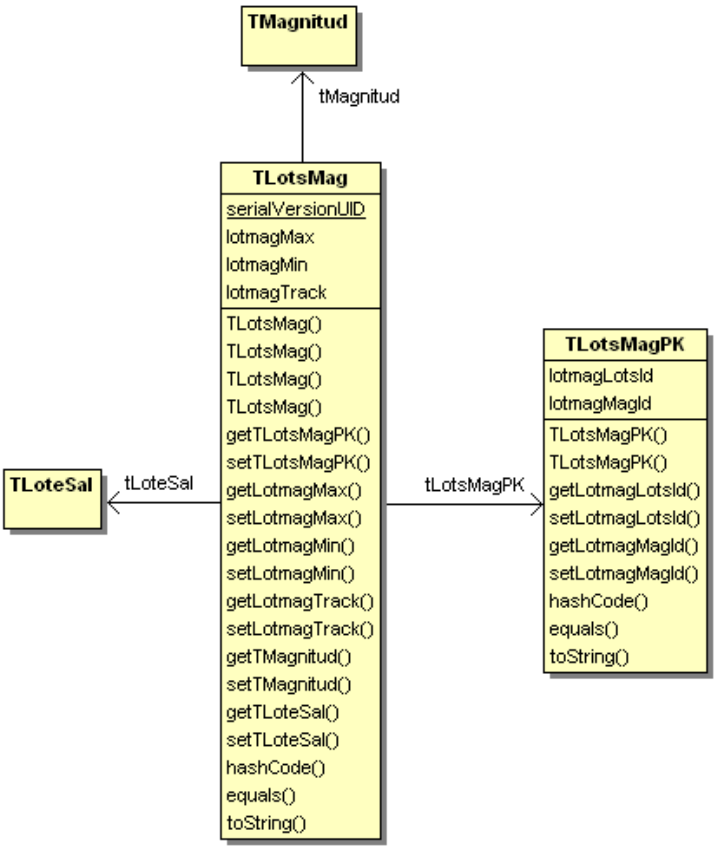

Figura 3.14: Diagrama de la clase TLotsMag

TLotsMagPK.java

- Constructores
  - TLotsMagPK()
  - TLotsMagPK(java.lang.String lotmagLotsId, java.math.BigInteger lotmagMagId)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getLotmagLotsId()
  - java.math.BigInteger getLotmagMagId()
  - int hashCode()

- void setLotmagLotsId(java.lang.String lotmagLotsId)
- void setLotmagMagId(java.math.BigInteger lotmagMagId)
- java.lang.String toString()

TMagnitud.java

- Constructores
  - TMagnitud()
  - TMagnitud(java.math.BigDecimal magId)
  - TMagnitud(java.math.BigDecimal magId, java.lang.String magNombre, java.math.BigInteger magEstado)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.util.Date getMagDateEnd()
  - java.lang.String getMagDescrip()
  - java.math.BigInteger getMagEstado()
  - java.math.BigDecimal getMagId()
  - java.lang.String getMagNombre()
  - int hashCode()
  - void setMagDateEnd(java.util.Date magDateEnd)
  - void setMagDescrip(java.lang.String magDescrip)
  - void setMagEstado(java.math.BigInteger magEstado)
  - void setMagId(java.math.BigDecimal magId)
  - void setMagNombre(java.lang.String magNombre)
  - java.lang.String toString()

TRecepcion.java

- Constructores
  - TRecepcion()
  - TRecepcion(java.lang.String recId)
  - TRecepcion(java.lang.String recId, java.util.Date recDate, java.math.BigInteger recContador, java.math.BigInteger recEstado)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getRecAlbaran()
  - java.math.BigInteger getRecContador()
  - java.util.Date getRecDate()
  - java.util.Date getRecDateEnd()
  - java.lang.String getRecDniTransp()
  - java.math.BigInteger getRecEstado()
  - java.lang.String getRecId()
  - TLocalizacion getRecLocId()
  - java.lang.String getRecMatricula()
  - TTurno getRecTurId()
  - TTurno getRecTurIdEnd()
  - int hashCode()
  - void setRecAlbaran(java.lang.String recAlbaran)
  - void setRecContador(java.math.BigInteger recContador)
  - void setRecDate(java.util.Date recDate)
  - void setRecDateEnd(java.util.Date recDateEnd)

- void setRecDniTransp(java.lang.String recDniTransp)
- void setRecEstado(java.math.BigInteger recEstado)
- void setRecId(java.lang.String recId)
- void setRecLocId(TLocalizacion recLocId)
- void setRecMatricula(java.lang.String recMatricula)
- void setRecTurId(TTurno recTurId)
- void setRecTurIdEnd(TTurno recTurIdEnd)
- java.lang.String toString()

TTag.java

- Constructores
  - TTag()
  - TTag(java.lang.String tagId)
  - TTag(java.lang.String tagId, java.math.BigInteger tagEstado)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.util.Date getTagDateEnd()
  - java.lang.String getTagDescrip()
  - java.math.BigInteger getTagEstado()
  - java.lang.String getTagId()
  - int hashCode()
  - void setTagDateEnd(java.util.Date tagDateEnd)
  - void setTagDescrip(java.lang.String tagDescrip)
  - void setTagEstado(java.math.BigInteger tagEstado)
  - void setTagId(java.lang.String tagId)
  - java.lang.String toString()

TTipo.java

- Constructores
  - TTipo()
  - TTipo(java.math.BigDecimal tipId)
  - TTipo(java.math.BigDecimal tipId, java.lang.String tipNombre)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.lang.String getTipDescrip()
  - java.math.BigDecimal getTipId()
  - java.lang.String getTipNombre()
  - int hashCode()
  - void setTipDescrip(java.lang.String tipDescrip)
  - void setTipId(java.math.BigDecimal tipId)
  - void setTipNombre(java.lang.String tipNombre)
  - java.lang.String toString()

TTurno.java

- Constructores
  - TTurno()
  - TTurno(java.math.BigDecimal turId)
  - TTurno(java.math.BigDecimal turId, java.lang.String turNombre, java.math.BigInteger turEstado)
- Métodos
  - boolean equals(java.lang.Object object)

- java.util.Date getTurDateEnd()
- java.lang.String getTurDescrip()
- java.math.BigInteger getTurEstado()
- java.math.BigDecimal getTurId()
- java.lang.String getTurNombre()
- int hashCode()
- void setTurDateEnd(java.util.Date turDateEnd)
- void setTurDescrip(java.lang.String turDescrip)
- void setTurEstado(java.math.BigInteger turEstado)
- void setTurId(java.math.BigDecimal turId)
- void setTurNombre(java.lang.String turNombre)
- java.lang.String toString()

TUnidad.java

- Constructores
  - TUnidad()
  - TUnidad(java.lang.String uniId)
  - TUnidad(java.lang.String uniId, java.lang.String uniIdExt, java.math.BigInteger uniContador, java.math.BigInteger uniCant, java.math.BigInteger uniCantAct)
- Métodos
  - boolean equals(java.lang.Object object)
  - java.math.BigInteger getUniCant()
  - java.math.BigInteger getUniCantAct()
  - java.math.BigInteger getUniContador()

- java.lang.String getUniDescrip()
- java.lang.String getUniId()
- java.lang.String getUniIdExt()
- TRecepcion getUniRecId()
- TTipo getUniTipoId()
- int hashCode()
- void setUniCant(java.math.BigInteger uniCant)
- void setUniCantAct(java.math.BigInteger uniCantAct)
- void setUniContador(java.math.BigInteger uniContador)
- void setUniDescrip(java.lang.String uniDescrip)
- void setUniId(java.lang.String uniId)
- void setUniIdExt(java.lang.String uniIdExt)
- void setUniRecId(TRecepcion uniRecId)
- void setUniTipoId(TTipo uniTipoId)
- java.lang.String toString()

Estos archivos se han generado de forma automática gracias al asistente para aplicaciones con conexiones a bases de datos que ofrece el entorno de desarrollo NetBeans[4].

3.1.8.2. Paquete hiridenda

Este es el paquete principal de la aplicación, ya que contiene tanto las clases que gestionan la ontología y las conexiones con la base de datos, como el interfaz gráfico del programa. A continuación se enumeran y explican los archivos que componen este paquete.

[4]http://netbeans.org/

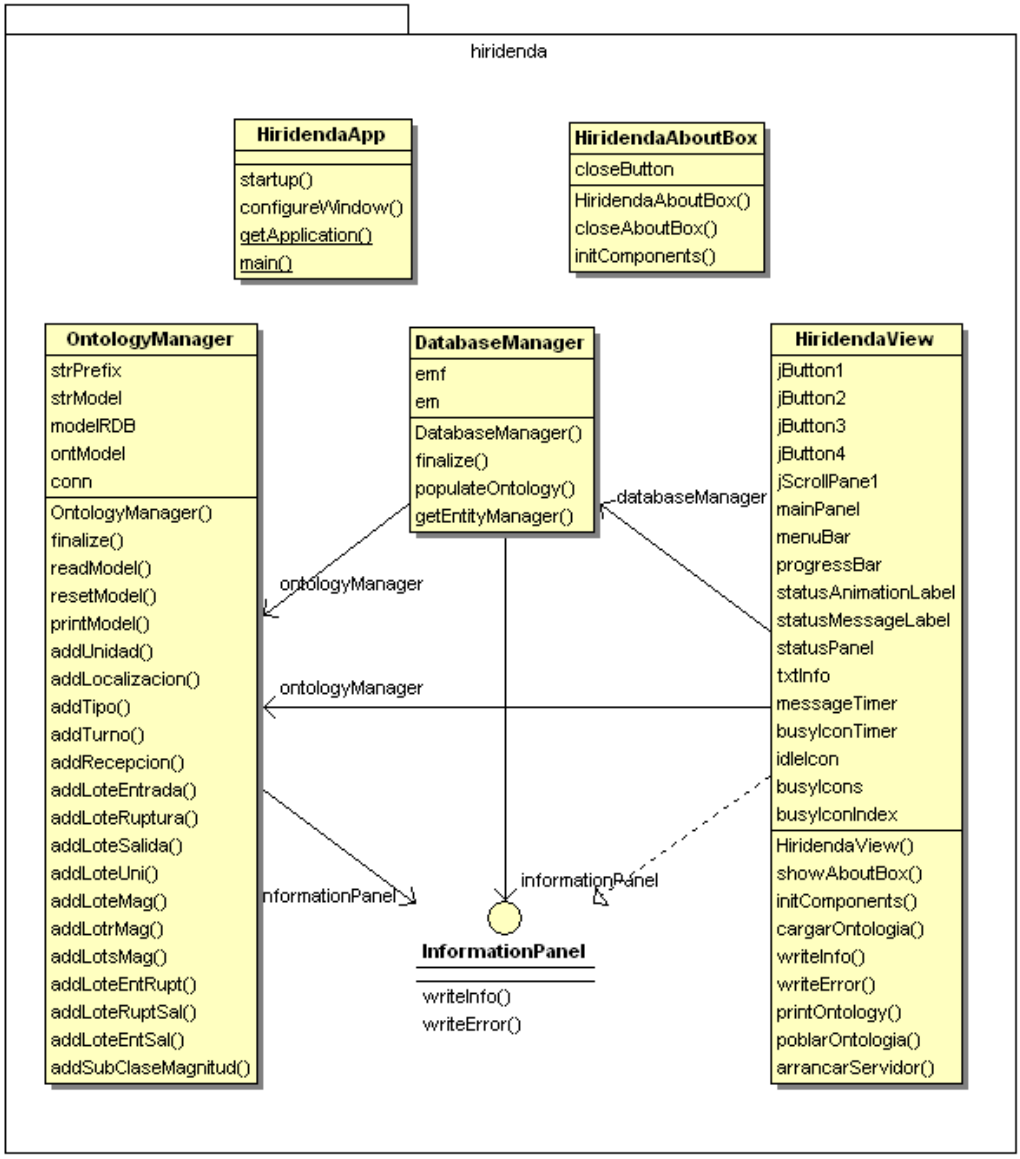

Figura 3.15: Diagrama del paquete hiridenda

DatabaseManager.java Esta clase se encarga de leer la información que contiene la base de datos, y trasladarla a la ontología. Para ello, cuenta con una sola función principal denominada populateOntology que obtiene todas las instancias de las tablas que son relevantes para la ontología. Tras esto pasa la información obtenida a la clase ontologyManager (ver a continuación).

- Constructores
  - DatabaseManager(InformationPanel informationPanel, OntologyManager ontologyManager)

- Métodos
  - protected void finalize()
  - void populateOntology()

OntologyManager.java Como indica su nombre, esta clase es la encargada de gestionar todas las acciones relacionadas con la ontología.

- Constructores
  - OntologyManager(InformationPanel informationPanel): El constructor establece una conexión con la base de datos, de forma que la ontología se almacene de forma persistente, mediante una variable de tipo ModelRDB[5]. Además, por encima de esta conexión, se crea una máscara de tipo OntModel[6] que permite un manejo más sencillo de la ontología.
- Métodos
  - void addLocalizacion(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger boolEstado)
  - void addLoteEntrada(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - void addLoteEntRupt(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteEntSal(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)

[5]Según la documentación oficial, ModelRDB es una implementación persistente, mediante una base de datos relacional, de la API de RDF.

[6]OntModel es una vista mejorada de un modelo Jena que se sabe que contiene datos de una ontología. La documentación oficial se puede encontrar en http://jena.sourceforge.net/javadoc/

- void addLoteRuptSal(java.lang.String domainID, java.lang.String rangeID)
- void addLoteRuptura(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
- void addLoteSalida(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
- void addLoteUni(java.lang.String domainID, java.lang.String rangeID)
- void addLotrMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
- void addLotsMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
- void addRecepcion(java.lang.String intID, java.lang.String strMatricula, java.lang.String strDNI, java.math.BigInteger boolEstado, java.lang.String strTurID, java.lang.String strLocID)
- void addSubClaseMagnitud(java.lang.String intID)
- void addTipo(java.lang.String intID, java.lang.String strNombre)
- void addTurno(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger boolEstado)
- void addUnidad(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.lang.String strTipoID, java.lang.String strRecepcionID)
- protected void finalize(): El destructor comprueba si la conexión con la

base de datos sigue activa, y en caso afirmativo la cierra para que no interfiera con ejecuciones posteriores.

- void printModel(): Esta función nos permite visualizar, mediante un árbol en la interfaz gráfica, el contenido de la ontología. Se recorren todas las clases mostrando su nombre, y para cada instancia de cada clase se muestra su nombre y los valores de sus propiedades.
- void readModel(java.lang.String strPath): Para cargar desde cero una ontología, se suministra al programa la estructura desde un archivo de extensión owl. Antes de leerlo, se ejecuta la función resetModel, que elimina todo el contenido de la ontología anterior.
- void resetModel(): Como se ha indicado anteriormente, esta función borra tanto las clases como las instancias y sus propiedades del modelo RDB.

Como se puede observar en el listado de funciones, se ha implementado una función para añadir una instancia por cada una de las clases definidas en la ontología. El funcionamiento es parecido para todas las funciones, aunque el número de acciones ejecutadas varía dependiendo de las propiedades de la clase en cuestión. El esquema general es el siguiente:

1. Se obtiene una variable de tipo OntClass con la clase que interesa.
2. Se crea una nueva instancia formada por el nombre de la clase más el identificador numérico asignado por la base de datos.
3. Se añade la instancia a la clase.
4. Si la clase tiene propiedades de datos, se toman los valores y se asignan a la instancia.
5. Si la clase tiene propiedades de objetos, se asignan los individuos correspondientes a la instancia creada.

3.1.8.3. Paquete hiridenda.RMI

En este paquete están implementadas las clases necesarias para llevar a cabo la comunicación mediante el mecanismo RMI. Algunos archivos se utilizan en la propia aplicación, y otros deben ser cargados en la base de datos y compilados.

Algunos de los archivos han sido adaptados del tutorial sobre RMI con Eclipse que se puede encontrar en `http://code.nomad-labs.com/2010/03/26/an-improved-rmi-tutorial`

Es importante destacar que todas las funciones que se vayan a ejecutar desde la base de datos, deben ser de tipo estático.

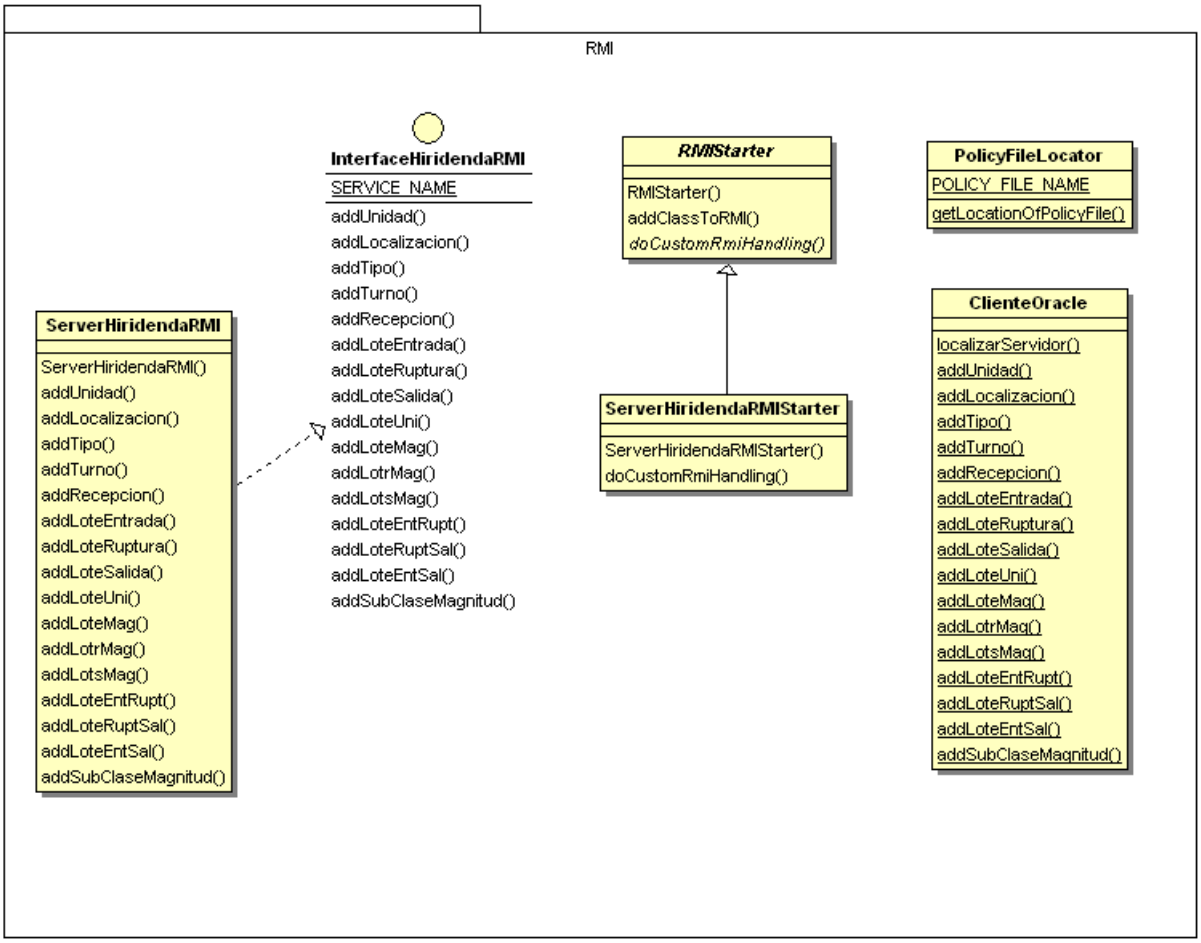

Figura 3.16: Diagrama del paquete hiridenda.RMI

A continuación se detallan los archivos utilizados:

ClienteOracle.java En esta clase se encuentran las funciones que Oracle llamará a través de los disparadores cada vez que haya modificaciones en la base de datos.

- Constructores
  - ClienteOracle()
- Métodos
  - static void addLocalizacion(java.lang.String intID, java.lang.String strNombre, java.lang.Integer boolEstado)
  - static void addLoteEntrada(java.lang.String intID, java.lang.String strNombre, java.lang.Integer intCantidad, java.lang.Integer intCantidadActual, java.lang.Integer boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - static void addLoteEntRupt(java.lang.String domainID, java.lang.String rangeID)
  - static void addLoteEntSal(java.lang.String domainID, java.lang.String rangeID)
  - static void addLoteMag(java.lang.String domainID, java.lang.String rangeID, java.lang.Integer intMax, java.lang.Integer intMin, java.lang.Integer boolTrack)
  - static void addLoteRuptSal(java.lang.String domainID, java.lang.String rangeID)
  - static void addLoteRuptura(java.lang.String intID, java.lang.String strNombre, java.lang.Integer intCantidad, java.lang.Integer intCantidadActual, java.lang.Integer boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - static void addLoteSalida(java.lang.String intID, java.lang.String strNombre, java.lang.Integer intCantidad, java.lang.Integer intCantidadActual, java.lang.Integer boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)

- static void addLoteUni(java.lang.String domainID, java.lang.String rangeID)
- static void addLotrMag(java.lang.String domainID, java.lang.String rangeID, java.lang.Integer intMax, java.lang.Integer intMin, java.lang.Integer boolTrack)
- static void addLotsMag(java.lang.String domainID, java.lang.String rangeID, java.lang.Integer intMax, java.lang.Integer intMin, java.lang.Integer boolTrack)
- static void addRecepcion(java.lang.String intID, java.lang.String strMatricula, java.lang.String strDNI, java.lang.Integer boolEstado, java.lang.String strTurID, java.lang.String strLocID)
- static void addSubClaseMagnitud(java.lang.String intID)
- static void addTipo(java.lang.String intID, java.lang.String strNombre)
- static void addTurno(java.lang.String intID, java.lang.String strNombre, java.lang.Integer boolEstado)

localizarServidor Esta función auxiliar se encarga de establecer una conexión con el servidor cada vez que se vaya a ejecutar una modificación.

Además de esta función auxiliar, existe una función add*Nombre_de_la_clase* por cada clase de la ontología, que coincide con el listado mostrado en OntologyManager.java. En algunas de estas funciones se han tenido que hacer conversiones entre tipos numéricos debido a las diferencias entre Oracle y Jena.

InterfaceHiridendaRMI.java Esta clase, que debe ser utilizada tanto en el cliente RMI como en el servidor, se define el nombre del servicio que deberá ser buscado por el cliente. Además, se definen todas las funciones que ofrece el servidor, que una vez más coincide con el listado de tipo add*Nombre_de_la_clase* de OntologyManager.java.

Todas las funciones deben arrojar la excepción RemoteException, que contempla aquéllas excepciones relacionadas con la comunicación como la falta de respuesta.

- Métodos

  - void addLocalizacion(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger boolEstado)
  - void addLoteEntrada(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - void addLoteEntRupt(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteEntSal(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
  - void addLoteRuptSal(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteRuptura(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - void addLoteSalida(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - void addLoteUni(java.lang.String domainID, java.lang.String rangeID)
  - void addLotrMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)

- void addLotsMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
- void addRecepcion(java.lang.String intID, java.lang.String strMatricula, java.lang.String strDNI, java.math.BigInteger boolEstado, java.lang.String strTurID, java.lang.String strLocID)
- void addSubClaseMagnitud(java.lang.String intID)
- void addTipo(java.lang.String intID, java.lang.String strNombre)
- void addTurno(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger boolEstado)

PolicyFileLocator.java Debido a las restricciones de seguridad impuestas por Java, es necesario definir el acceso que se le permite a una aplicación. Mediante esta clase se crea un archivo server.policy en el directorio adecuado a partir de uno ya existente donde se define que el acceso de la aplicación es ilimitado.

- Constructores
  - PolicyFileLocator()
- Métodos
  - static java.lang.String getLocationOfPolicyFile()

RMIStarter.java Aunque esta clase no se usa directamente, ya que es una clase abstracta, contiene las funciones básicas necesarias para arrancar un servicio RMI.

- Constructores
  - RMIStarter()
- Métodos
  - protected void addClassToRMI(java.lang.Class classToAdd)
  - abstract void doCustomRmiHandling()

ServerHiridendaRMI.java Este archivo es el equivalente a ClienteOracle.java para el servidor. Recibe los datos del cliente, y pasa la información al gestor de la ontología para que realice las modificaciones oportunas.

- Constructores
  - ServerHiridendaRMI(OntologyManager ontologyManager)
- Métodos
  - void addLocalizacion(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger boolEstado)
  - void addLoteEntrada(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - void addLoteEntRupt(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteEntSal(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
  - void addLoteRuptSal(java.lang.String domainID, java.lang.String rangeID)
  - void addLoteRuptura(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)
  - void addLoteSalida(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger intCantidad, java.math.BigInteger intCantidadActual, java.math.BigInteger boolEstado, java.lang.String strDate, java.lang.String strTurID, java.lang.String strLocID)

- void addLoteUni(java.lang.String domainID, java.lang.String rangeID)
- void addLotrMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
- void addLotsMag(java.lang.String domainID, java.lang.String rangeID, java.math.BigInteger intMax, java.math.BigInteger intMin, java.math.BigInteger boolTrack)
- void addRecepcion(java.lang.String intID, java.lang.String strMatricula, java.lang.String strDNI, java.math.BigInteger boolEstado, java.lang.String strTurID, java.lang.String strLocID)
- void addSubClaseMagnitud(java.lang.String intID)
- void addTipo(java.lang.String intID, java.lang.String strNombre)
- void addTurno(java.lang.String intID, java.lang.String strNombre, java.math.BigInteger boolEstado)

ServerHiridendaRMIStarter.java Implementa la clase abstracta RMIStarter, añadiendo la clase InterfaceHiridendaRMI al servidor RMI y arrancando el servicio con el nombre dado en InterfaceHiridendaRMI.

- Constructores
  - ServerHiridendaRMIStarter(OntologyManager ontologyManager, InformationPanel informationPanel)
- Métodos
  - void doCustomRmiHandling()

3.1.9. Ejecución de la aplicación

Para poder llevar a cabo la ejecución de prueba que se ha diseñado, se deben realizar los siguientes pasos, una vez que se haya creado la base de datos correspondiente

al almacén, que en nuestro caso se llamará hiridenda, con la información completada, y la base de datos que utilizará Jena para guardar de una forma persistente la ontología. Esta última base de datos estará al principio vacía.

1. Cargar en la base de datos hiridenda las clases Java necesarias para poder definir los procedimientos almacenados. En la siguiente imagen se puede observar la localización de la opción si se ejecuta desde sqldeveloper.

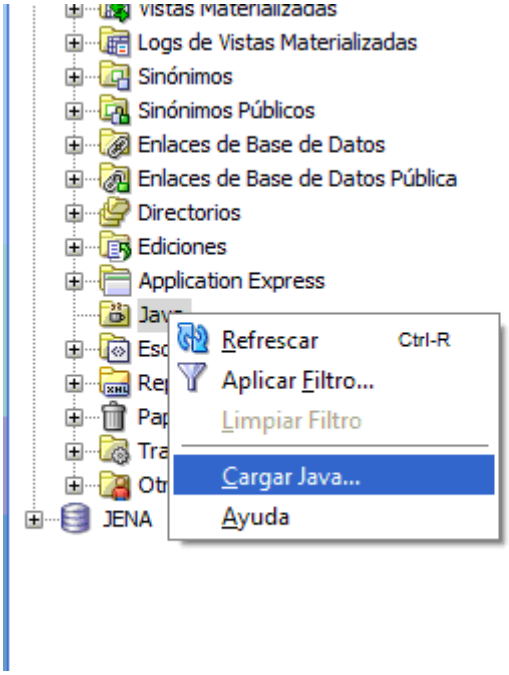

Figura 3.17: Carga de clases Java desde sqldeveloper

Estas clases ya han sido explicadas en el punto 3.1.8.3, se encuentran dentro de la carpeta \src\hiridenda\RMI\ y deben cargarse obligatoriamente debido a las dependencias en el siguiente orden:

   a) InterfaceHiridendaRMI.java

   b) ClienteOracle.java

2. Definir los procedimientos almacenados relacionándolos con las funciones Java cargadas anteriormente. A continuación se muestra un fragmento de código que define uno de los procedimientos almacenados que se han utilizado en el proyecto:

Algoritmo 3.1 Definición de un procedimiento almacenado en Oracle

```
CREATE OR REPLACE PROCEDURE addLocalizacion(intID VARCHAR2, strNombre
VARCHAR2, boolEstado NUMBER)
AS LANGUAGE JAVA
NAME 'hiridenda/RMI/ClienteOracle.addLocalizacion(java.lang.String,
java.lang.String, java.lang.Integer)';
```

3. Crear los disparadores que accionarán los procedimientos almacenados definidos anteriormente. Siguiendo el ejemplo anterior, el disparador para las modificaciones que se realicen sobre las localizaciones se definiría de la siguiente forma:

Algoritmo 3.2 Definición de un disparador en Oracle

```
create or replace trigger addLocalizacion
after insert or update on T_LOCALIZACION
for each row
begin
addLocalizacion(:new.LOC_ID,:new.LOC_NOMBRE,:new.LOC_ESTADO);
end;
```

4. Conceder permisos a la base de datos para que pueda comunicarse con el servidor RMI. En el caso de ejemplo, estando tanto el cliente como el servidor RMI en el mismo equipo con dirección IP 192.168.1.30 se han dado los siguientes permisos:

Algoritmo 3.3 Permisos de acceso para los procedimientos almacenados

```
CALL dbms_java.grant_permission( 'SYSTEM', 'SYS:java.net.SocketPermission',
'127.0.0.1:*', 'connect,resolve' );
CALL dbms_java.grant_permission( 'SYSTEM', 'SYS:java.net.SocketPermission',
'192.168.1.30:*', 'connect,resolve' );
```

5. Arrancar el registro RMI en la máquina donde se ubica el servidor. Si se está trabajando en un entorno Windows, esto se puede realizar ejecutando desde la consola de comandos la orden `start rmiregistry`.

6. Arrancar la aplicación Java desde la consola de comandos o directamente haciendo doble clic en la compilación realizada para ser distribuida.

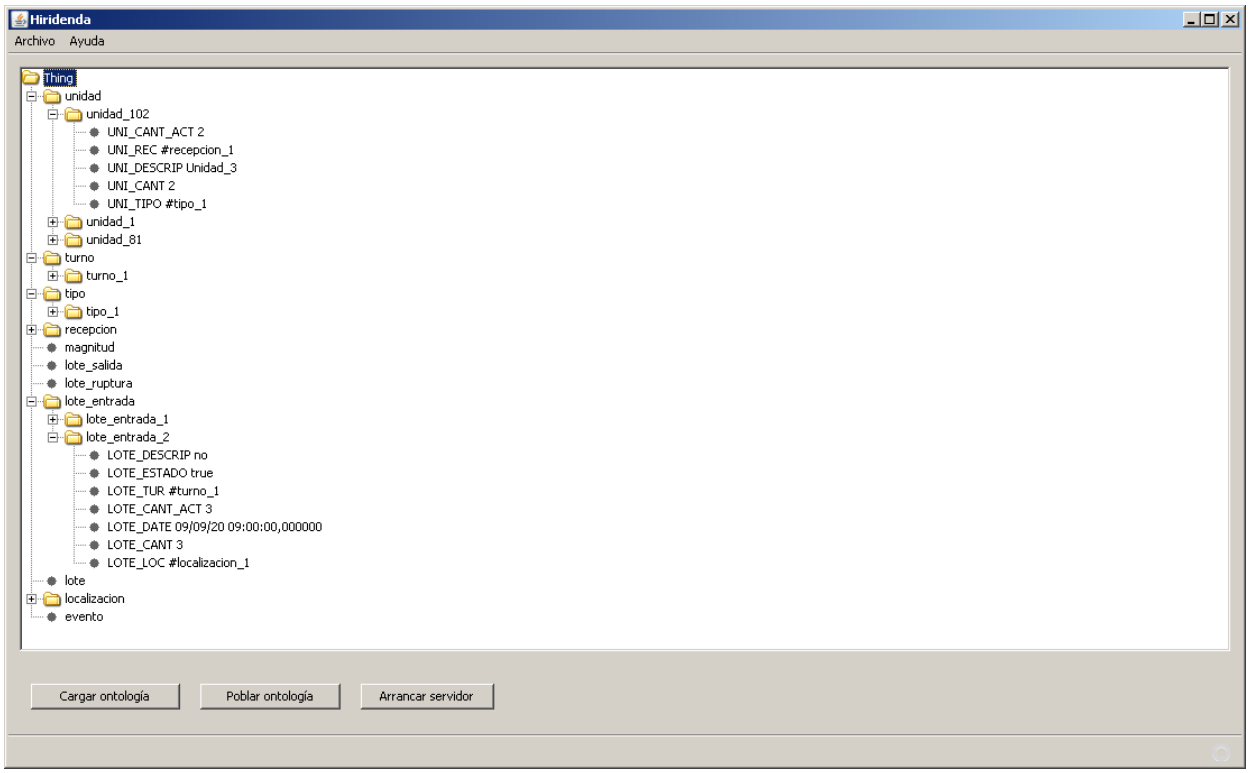

Figura 3.18: Captura de pantalla de la aplicación de ejemplo

7. Arrancar el servidor RMI desde el botón habilitado para ello en el programa.

8. Cargar la ontología que se ha definido mediante Protégé en un archivo owl. Esto provocará que la ontología actual, tanto en memoria como en la base de datos, se vacíe y se inicialice con la nueva estructura.

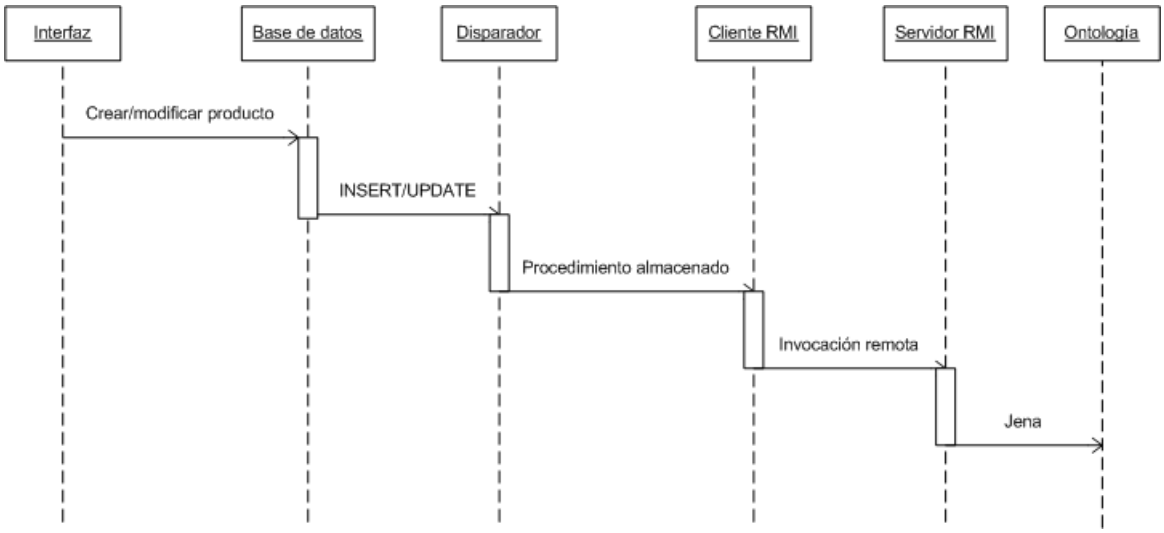

Figura 3.19: Diagrama de secuencia

1. Poblar la ontología con la información de la base de datos hiridenda.

2. A partir de este momento, la aplicación irá recibiendo todas las actualizaciones que se realicen en la base de datos hiridenda, pudiéndose imprimir el contenido de la ontología en cualquier momento para observar su evolución.

3.1.10. Base de datos

A continuación se describen las tablas que conforman la base de datos, especificando por cada campo su contenido y tipo de datos. También se incluye el detalle de la relación entre las tablas.

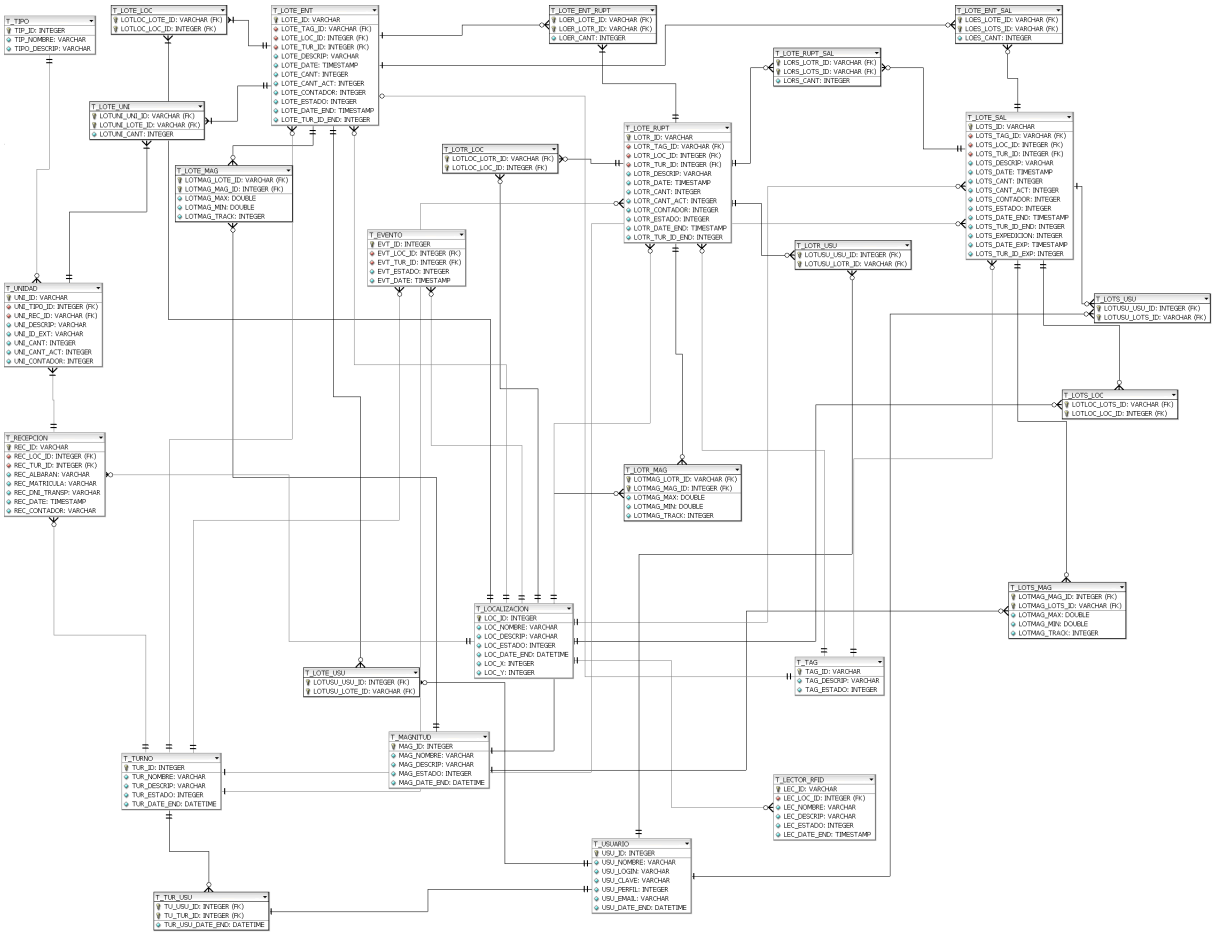

Figura 3.20: Diagrama Entidad-Relación de la base de datos

3.1.10.1. Entidad T_TURNO

Esta entidad almacena un registro por cada uno de los turnos de trabajo definidos.

Nombre	Tipo	Descripción	Obligatorio
TUR_ID	Number	Código identificador y único del turno. Su generación será automática al realizar una inserción en la base de datos.	X
TUR_NOMBRE	Varchar	Nombre asignado al turno. Deberá ser único.	X
TUR_DESCRIP	Varchar	Descripción del turno.	
TUR_ESTADO	Number	Indicador del estado del turno: activo o no. Sólo admitirá dos posibles valores: 0 Turno activo, valor por defecto 1 Turno inactivo.	X
TUR_DATE_END	Date	Fecha en la que el turno pasa a estado inactivo. El valor de este campo para los turnos activos será NULL.	
Clave Primaria	TUR_ID		

Cuadro 3.1: Entidad T_TURNO

3.1.10.2. Entidad T_USUARIO

Esta entidad almacena un registro por cada uno de los usuarios con acceso al sistema.

Nombre	Tipo	Descripción	Obligatorio
USU_ID	Number	Código identificador y único del usuario. Su generación será automática al realizar una inserción en la base de datos.	X
USU_NOMBRE	Varchar	Nombre del usuario.	X
USU_LOGIN	Varchar	Login del usuario para el acceso al sistema.	X
USU_CLAVE	Varchar	Contraseña del usuario para acceder al sistema. Deberá ser única.	X
TUR_PERFIL	Number	Indicador del perfil del usuario: administrador o no. Sólo admitirá dos posibles valores: ▪ 0 Administrador. ▪ 1 No administrador, valor por defecto.	X
USU_DATE_END	Date	Fecha en la que el usuario es marcado como dado de baja. El valor de este campo para los usuarios activos será NULL.	
Clave Primaria	USU_ID		

Cuadro 3.2: Entidad T_USUARIO

3.1.10.3. Entidad T_TUR_USU

Esta entidad almacena un registro por cada asociación turno - usuario.

Nombre	Tipo	Descripción	Obligatorio
TU_TUR_ID	Number	Código identificador y único del turno. Su generación será automática al realizar una inserción en la base de datos.	X
TU_USU_ID	Number	Código identificador y único del usuario. Su generación será automática al realizar una inserción en la base de datos.	X
TUR_USU_DATE_END	Date	Fecha en la que el usuario es cambiado de turno. El turno activo tendrá por defecto el valor NULL.	

Clave Primaria	TU_TUR_ID & TU_USU_ID	
Clave Foránea	Campo	Tabla - Campo
	TU_TUR_ID	T_TURNO – TUR_ID
	TU_USU_ID	T_USUARIO – USU_ID

Cuadro 3.3: Entidad T_TUR_USU

3.1.10.4. Entidad T_LOCALIZACION

Esta entidad registrará todas las posibles localizaciones asignables a un lote.

Nombre	Tipo	Descripción	Obligatorio
LOC_ID	Number	Código identificador y único de la localización. Su generación será automática al realizar una inserción en la base de datos.	X
LOC_NOMBRE	Varchar	Nombre de la localización.	X
LOC_DESCRIP	Varchar	Descripción de la localización.	
LOC_ESTADO	Number	Estado de la localización: activa o no activa. Sólo admitirá dos posibles valores: ▪ 0 Activa, valor por defecto. ▪ 1 No activa.	X
LOC_DATE_END	Timestamp	Fecha en la que la localización es marcada como inactiva. El valor de este campo para las localizaciones activas será NULL.	
LOC_X	Number	Posición X de la localización	X
LOC_Y	Number	Posición Y de la localización	X

Clave Primaria	LOC_ID

Cuadro 3.4: Entidad T_LOCALIZACION

3.1.10.5. Entidad T_TAG

Esta entidad recoge un registro por cada una de las etiquetas de localización asignables a los lotes.

Nombre	Tipo	Descripción	Obligatorio
TAG_ID	Varchar	Identificador del tag	X
TAG_DESCRIP	Varchar	Descripción del tag	
TAG_ESTADO	Number	Estado del tag: libre u ocupado. Sólo admitirá dos posibles valores: ▪ 0 Libre. ▪ 1 Ocupado, es decir, asignado a un lote.	X

Clave Primaria	TAG_ID

Cuadro 3.5: Entidad T_TAG

3.1.10.6. Entidad T_LECTOR_RFID

Esta entidad almacena un registro por cada lector definido.

Nombre	Tipo	Descripción	Obligatorio
LEC_ID	Varchar	Código identificador y único del lector.	X
LEC_NOMBRE	Varchar	Nombre del lector.	X
LEC_DESCRIP	Varchar	Descripción del lector.	
LEC_LOC_ID	Integer	Localización asociada al lector	X
LEC_ESTADO	Integer	Estado del lector activo o no activo. Sólo admitirá dos posibles valores: ▪ 0 Activo, valor por defecto. ▪ 1 No activo.	
LEC_DATE_END	Date	Fecha en la que el lector es marcado como inactivo. El turno activo tendrá por defecto el valor NULL.	

Clave Primaria	LEC_ID	
Clave Foránea	Campo	Tabla - Campo
	LEC_LOC_ID	T_LOCALIZACION – LOC_ID

Cuadro 3.6: Entidad T_LECTOR_RFID

3.1.10.7. Entidad T_MAGNITUD

Esta entidad registrará las diferentes magnitudes a controlar por los sensores.

Nombre	Tipo	Descripción	Obligatorio
MAG_ID	Number	Código identificador y único de la magnitud. Su generación será automática al realizar una inserción en la base de datos.	X
MAG_NOMBRE	Varchar	Nombre de la magnitud	X
MAG_DESCRIP	Varchar	Descripción de la magnitud	
MAG_ESTADO	Number	Estado de la magnitud: activa o no activa. Sólo admitirá dos posibles valores: ▪ 0 Activa, valor por defecto. ▪ 1 No activa.	X
MAG_DATE_END	Timestamp	Fecha en la que la magnitud es marcada como inactiva. El valor de este campo para las magnitudes activas será NULL.	

Clave Primaria	MAG_ID

Cuadro 3.7: Entidad T_MAGNITUD

3.1.10.8. Entidad T_RECEPCION

Esta entidad almacenará un registro por cada unas de las entradas al almacén.

Nombre	Tipo	Descripción	Obligatorio
REC_ID	Varchar	Identificador de la recepción	X
REC_ALBARAN	Varchar	Identificador externo del albarán de entrada	X
REC_MATRICULA	Varchar	Matrícula del remolque	X
REC_DNI_TRANSP	Varchar	DNI del transportista	X
REC_DATE	Timestamp	Fecha de la recepción, por defecto la del sistema	X
REC_TUR_ID	Integer	Turno que genera el registro	X
REC_LOC_ID	Number	Identificador de la localización donde se realiza la recepción.	X
REC_CONTADOR	Varchar	Contador secuencial del código	X
REC_ESTADO	Integer	Flag que indica si la recepción ha sido borrada de forma lógica. Admite sólo dos valores ▪ 0: Activa ▪ 1: Inactiva, marcada como borrada	X
REC_DATE_END	Timestamp	Fecha en la que se ha marcado el registro como borrado.	
REC_TUR_ID_END	Integer	Turno que ha realizado el marcaje de borrado.	

Clave Primaria	REC_ID	
Clave Foránea	Campo	Tabla - Campo
	REC_TUR_ID	T_TURNO – TUR_ID
	REC_LOC_ID	T_LOCALIZACION – LOC_ID
	REC_TUR_ID_END	T_TURNO – TUR_ID

Cuadro 3.8: Entidad T_RECEPCION

3.1.10.9. Entidad T_TIPO

Esta entidad almacenará los diferentes tipos de unidades. Se alimenta mediante la llamada a un servicio web.

Nombre	Tipo	Descripción	Obligatorio
TIP_ID	Varchar	Código identificador y único	X
TIP_NOMBRE	Varchar	Nombre	X
TIP_DESCRIPT	Varchar	Descripción	
Clave Primaria	TIP_ID		

Cuadro 3.9: Entidad T_TIPO

3.1.10.10. Entidad T_UNIDAD

Esta entidad almacenará las diferentes unidades recibidas en el sistema.

Nombre	Tipo	Descripción	Obligatorio
UNI_ID	Varchar	Código identificador y único de la unidad.	X
UNI_DESCRIP	Varchar	Descripción de la unidad	
UNI_ID_EXT	Varchar	Identificador externo de la unidad	X
UNI_TIPO_ID	Varchar	Identificador del tipo de la unidad: pantalones, pasta, ...	X
UNI_REC_ID	Varchar	Identificador de la recepción	X
UNI_CANT	Integer	Número de bultos o paquetes que componen la unidad	X
UNI_CANT_ACT	Integer	Número de bultos o paquetes actuales de la unidad	X
UNI_CONTADOR	Integer	Contador secuencial del código	X

Clave Primaria	UNI_ID	
Clave Foránea	Campo	Tabla - Campo
	UNI_REC_ID	T_RECEPCION – REC_ID
	UNI_TIPO_ID	T_TIPO – TIPO_ID

Cuadro 3.10: Entidad T_UNIDAD

3.1.10.11. Entidad T_LOTE_ENT

Esta entidad almacenará cada uno de los lotes resultantes de un proceso de entrada.

Nombre	Tipo	Descripción	Obligatorio
LOTE_ID	Varchar	Código identificador y único del lote.	X
LOTE_TUR_ID	Number	Identificador del turno que genera el lote	X
LOTE_DESCRIP	Varchar	Descripción del lote	
LOTE_DATE	Timestamp	Fecha de creación del lote	X
LOTE_CANT	Integer	Número de unidades que componen el lote	X
LOTE_TAG_ID	Varchar	Identificador del tag de localización asignado	X
LOTE_CANT_ACT	Integer	Unidades actuales asignadas al lote	X
LOTE_CONTADOR	Integer	Contador secuencial del código	X
LOTE_LOC_ID	Integer	Identificación de la localización donde se ha generado el lote	X
LOTE_ESTADO	Integer	Flag que indica si el lote ha sido borrado de forma lógica. Admite sólo dos valores ▪ 0: Activa ▪ 1: Inactiva, marcado como borrado	X
LOTE_DATE_END	Timestamp	Fecha en la que se ha marcado el registro como borrado	
LOTE_TUR_ID_END	Integer	Turno que ha realizado el marcaje de borrado	

Clave Primaria	LOTE_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTE_TAG_ID	T_TAG – TAG_ID
	LOTE_LOC_ID	T_LOCALIZACION – LOC_ID
	LOTE_TUR_ID	T_TURNO_ID

Cuadro 3.11: Entidad T_LOTE_ENT

3.1.10.12. Entidad T_LOTE_UNI

Esta entidad registra las unidades que componen cada lote de entrada

Nombre	Tipo	Descripción	Obligatorio
LOTUNI_UNI_ID	Varchar	Código identificador de la unidad.	X
LOTUNI_LOTE_ID	Varchar	Código identificador del lote.	X
LOTUNI_CANT	Integer	Número de unidades	X

Clave Primaria	LOTUNI_UNI_ID & LOTUNI_LOTE_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTUNI_UNI_ID	T_UNIDAD – UNI_ID
	LOTUNI_LOTE_ID	T_LOTE_ENT – LOTE_ID

Cuadro 3.12: Entidad T_LOTE_UNI

3.1.10.13. Entidad T_LOTE_MAG

Esta entidad recoge la asignación de magnitudes a lotes.

Nombre	Tipo	Descripción	Obligatorio
LOTMAG_LOTE_ID	Varchar	Código identificador del lote.	X
LOTMAG_MAG_ID	Integer	Código identificador de la magnitud.	X
LOTMAG_MAX	Number	Límite superior permitido	X
LOTMAG_MIN	Number	Límite inferior permitido	
LOTMAG_TRACK		Integer 0 si la magnitud es medible entre límite inferior y superior, o 1 en caso contrario siendo el valor bueno de la magnitud el recogido en el campo LOTMAG_MAX	X

Clave Primaria	LOTMAG_LOTE_ID & LOTMAG_MAG_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTMAG_MAG_ID	T_MAGNITUD – MAG_ID
	LOTMAG_LOTE_ID	T_LOTE_ENT – LOTE_ID

Cuadro 3.13: Entidad T_LOTE_MAG

3.1.10.14. Entidad T_LOTE_LOC

Esta entidad recoge las localizaciones permitidas para un lote. La presencia en otra localización dará lugar a una alarma.

Nombre	Tipo	Descripción	Obligatorio
LOTLOC_LOC_ID	Number	Código identificador de la localización.	X
LOTLOC_LOTE_ID	Varchar	Código identificador del lote.	X

Clave Primaria	LOTLOC_LOC_ID & LOTLOC_LOTE_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTLOC_LOC_ID	T_LOCALIZACION – LOC_ID
	LOTLOC_LOTE_ID	T_LOTE_ENT – LOTE_ID

Cuadro 3.14: Entidad T_LOTE_LOC

3.1.10.15. Entidad T_LOTE_USU

Esta entidad recoge los usuarios suscriptores a un lote, es decir, los usuarios que recibirán alarmas ante magnitudes o localizaciones del lote erróneas.

Nombre	Tipo	Descripción	Obligatorio
LOTUSU_USU_ID	Number	Código identificador del usuario.	X
LOTUSU_LOTE_ID	Varchar	Código identificador del lote.	X

Clave Primaria	LOTUSU_USU_ID & LOTUSU_LOTE_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTUSU_USU_ID	T_USUARIO – USU_ID
	LOTUSU_LOTE_ID	T_LOTE_ENT – LOTE_ID

Cuadro 3.15: Entidad T_LOTE_USU

3.1.10.16. Entidad T_LOTE_RUPT

Esta entidad almacenará cada uno de los lotes resultantes de un proceso de ruptura de lotes.

Nombre	Tipo	Descripción	Obligatorio
LOTR_ID	Varchar	Código identificador y único del lote.	X
LOTR_TUR_ID	Number	Identificador del turno que genera el lote	X
LOTR_DESCRIP	Varchar	Descripción del lote	
LOTR_DATE	Timestamp	Fecha de creación del lote.	X
LOTR_CANT	Integer	Número de unidades que componen el lote.	X
LOTR_TAG_ID	Varchar	Identificador del tag de localización asignado.	X
LOTR_CANT_ACT	Integer	Unidades actuales asignadas al lote.	X
LOTR_CONTADOR	Integer	Contador secuencial del código.	X
LOTR_LOC_ID	Integer	Identificación de la localización donde se ha generado el lote.	X
LOTR_ESTADO	Integer	Flag que indica si el lote ha sido borrado de forma lógica. Admite sólo dos valores ▪ 0: Activa ▪ 1: Inactiva, marcado como borrado	X
LOTR_DATE_END	Timestamp	Fecha en la que se ha marcado el registro como borrado.	
LOTR_TUR_ID_END	Integer	Turno que ha realizado el marcaje de borrado.	

Clave Primaria	LOTR_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTR_TAG_ID	T_TAG – TAG_ID
	LOTR_LOC_ID	T_LOCALIZACION – LOC_ID
	LOTR_TUR_ID	T_TURNO_ID

Cuadro 3.16: Entidad T_LOTE_RUPT

3.1.10.17. Entidad T_LOTE_ENT_RUPT

Esta entidad registra los lotes de entrada que componen cada lote de ruptura.

Nombre	Tipo	Descripción	Obligatorio
LOER_LOTE_ID	Varchar	Código identificador del lote de entrada	X
LOER_LOTR_ID	Varchar	Código identificador del lote de ruptura.	X
LOER_CANT	Integer	Número de unidades del lote	X

Clave Primaria	LOER_LOTE_ID & LOER_LOTR_ID	
Clave Foránea	Campo	Tabla - Campo
	LOER_LOTR_ID	T_LOTE_RUPT – LOTR_ID
	LOER_LOTE_ID	T_LOTE_ENT – LOTE_ID

Cuadro 3.17: Entidad T_LOTE_ENT_RUPT

3.1.10.18. Entidad T_LOTR_MAG

Esta entidad recoge la asignación de magnitudes a lotes de ruptura.

Nombre	Tipo	Descripción	Obligatorio
LOTMAG_LOTR_ID	Varchar	Código identificador del lote de ruptura.	X
LOTMAG_MAG_ID	Integer	Código identificador de la magnitud.	X
LOTMAG_MAX	Number	Límite superior permitido	X
LOTMAG_MIN	Number	Límite inferior permitido	
LOTMAG_TRACK	Integer	0 si la magnitud es medible entre límite inferior y superior, o 1 en caso contrario siendo el valor bueno de la magnitud el recogido en el campo LOTMAG_MAX	X

Clave Primaria	LOTMAG_LOTR_ID & LOTMAG_MAG_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTMAG_MAG_ID	T_MAGNITUD – MAG_ID
	LOTMAG_LOTR_ID	T_LOTE_RUPT – LOTR_ID

Cuadro 3.18: Entidad T_LOTR_MAG

3.1.10.19. Entidad T_LOTR_LOC

Esta entidad recoge las localizaciones permitidas para un lote. La presencia en otra localización dará lugar a una alarma.

Nombre	Tipo	Descripción	Obligatorio
LOTLOC_LOC_ID	Number	Código identificador de la localización.	X
LOTLOC_LOTR_ID	Varchar	Código identificador del lote de ruptura.	X

Clave Primaria	LOTLOC_LOC_ID & LOTLOC_LOTR_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTLOC_LOC_ID	T_LOCALIZACION – LOC_ID
	LOTLOC_LOTR_ID	T_LOTE_RUPT – LOTR_ID

Cuadro 3.19: Entidad T_LOTR_LOC

3.1.10.20. Entidad T_LOTR_USU

Esta entidad recoge los usuarios suscriptores a un lote de ruptura, es decir, los usuarios que recibirán alarmas ante magnitudes o localizaciones del lote erróneas.

Nombre	Tipo	Descripción	Obligatorio
LOTUSU_USU_ID	Number	Código identificador del usuario.	X
LOTUSU_LOTR_ID	Varchar	Código identificador del lote de ruptura.	X

Clave Primaria	LOTUSU_USU_ID & LOTUSU_LOTR_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTUSU_USU_ID	T_USUARIO – USU_ID
	LOTUSU_LOTR_ID	T_LOTE_RUPT – LOTR_ID

Cuadro 3.20: Entidad T_LOTR_USU

3.1.10.21. Entidad T_LOTE_SAL

Esta entidad almacenará cada uno de los lotes resultantes de un proceso de ruptura de lotes de ruptura.

Nombre	Tipo	Descripción	Obligatorio
LOTS_ID	Varchar	Código identificador y único del lote.	X
LOTS_TUR_ID	Number	Identificador del turno que genera el lote	X
LOTS_DESCRIP	Varchar	Descripción del lote	
LOTS_DATE	Timestamp	Fecha de creación del lote	X
LOTS_CANT	Integer	Número de unidades que componen el lote	X
LOTS_TAG_ID	Varchar	Identificador del tag de localización asignado	X
LOTS_CANT_ACT	Integer	Unidades actuales asignadas al lote	X
LOTS_CONTADOR	Integer	Contador secuencial del código	X
LOTS_LOC_ID	Integer	Identificación de la localización donde se ha generado el lote	X
LOTS_ESTADO	Integer	Flag que indica si el lote ha sido borrado de forma lógica. Admite sólo dos valores ▪ 0: Activa ▪ 1: Inactiva, marcado como borrado	X
LOTS_DATE_END	Timestamp	Fecha en la que se ha marcado el registro como borrado	
LOTS_TUR_ID_END	Integer	Turno que ha realizado el marcaje de borrado	
LOTS_EXPEDICION	Integer	Flag que indica si el lote ha sido expedido. Dos posibles valores: ▪ 0: no expedido ▪ 1: expedido.	
LOTS_DATE_EXP	Timestamp	Fecha en la que se ha marcado el registro como expedido	
LOTS_TUR_ID_EXP	Integer	Turno que ha realizado el marcaje de expedición	

Clave Primaria	LOTS_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTS_TAG_ID	T_TAG – TAG_ID
	LOTS_LOC_ID	T_LOCALIZACION – LOC_ID
	LOTS_TUR_ID	T_TURNO_ID

Cuadro 3.21: Entidad T_LOTE_SAL

3.1.10.22. Entidad T_LOTE_ENT_SAL

Esta entidad registra los lotes de entrada que componen cada lote de salida.

Nombre	Tipo	Descripción	Obligatorio
LOES_LOTE_ID	Varchar	Código identificador del lote de entrada	X
LOES_LOTS_ID	Varchar	Código identificador del lote de salida.	X
LOES_CANT	Integer	Número de unidades del lote	X

Clave Primaria	LOES_LOTE_ID & LOES_LOTS_ID	
Clave Foránea	Campo	Tabla - Campo
	LOES_LOTS_ID	T_LOTE_SAL – LOTS_ID
	LOES_LOTE_ID	T_LOTE_ENT – LOTE_ID

Cuadro 3.22: Entidad T_LOTE_ENT_SAL

3.1.10.23. Entidad T_LOTE_RUPT_SAL

Esta entidad registra los lotes de ruptura que componen cada lote de salida.

Nombre	Tipo	Descripción	Obligatorio
LORS_LOTR_ID	Varchar	Código identificador del lote de ruptura	X
LORS_LOTS_ID	Varchar	Código identificador del lote de salida.	X
LORS_CANT	Integer	Número de unidades del lote	X

Clave Primaria	LORS_LOTR_ID & LORS_LOTS_ID	
Clave Foránea	Campo	Tabla - Campo
	LORS_LOTS_ID	T_LOTE_SAL – LOTS_ID
	LORS_LOTR_ID	T_LOTE_RUPT – LOTR_ID

Cuadro 3.23: Entidad T_LOTE_RUPT_SAL

3.1.10.24. Entidad T_LOTS_MAG

Esta entidad recoge la asignación de magnitudes a lotes de salida.

Nombre	Tipo	Descripción	Obligatorio
LOTMAG_LOTS_ID	Varchar	Código identificador del lote de salida.	X
LOTMAG_MAG_ID	Integer	Código identificador de la magnitud.	X
LOTMAG_MAX	Number	Límite superior permitido	X
LOTMAG_MIN	Number	Límite inferior permitido	
LOTMAG_TRACK	Integer	0 si la magnitud es medible entre límite inferior y superior, o 1 en caso contrario siendo el valor bueno de la magnitud el recogido en el campo LOTMAG_MAX	X

Clave Primaria	LOTMAG_LOTS_ID & LOTMAG_MAG_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTMAG_MAG_ID	T_MAGNITUD – MAG_ID
	LOTMAG_LOTS_ID	T_LOTE_SAL – LOTS_ID

Cuadro 3.24: Entidad T_LOTS_MAG

3.1.10.25. Entidad T_LOTS_LOC

Esta entidad recoge las localizaciones permitidas para un lote. La presencia en otra localización dará lugar a una alarma.

Nombre	Tipo	Descripción	Obligatorio
LOTLOC_LOC_ID	Number	Código identificador de la localización.	X
LOTLOC_LOTS_ID	Varchar	Código identificador del lote de salida.	X

Clave Primaria	LOTLOC_LOC_ID & LOTLOC_LOTS_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTLOC_LOC_ID	T_LOCALIZACION – LOC_ID
	LOTLOC_LOTS_ID	T_LOTE_SAL – LOTS_ID

Cuadro 3.25: Entidad T_LOTS_LOC

3.1.10.26. Entidad T_LOTS_USU

Esta entidad recoge los usuarios suscriptores a un lote de salida, es decir, los usuarios que recibirán alarmas ante magnitudes o localizaciones del lote erróneas.

Nombre	Tipo	Descripción	Obligatorio
LOTUSU_USU_ID	Number	Código identificador del usuario.	X
LOTUSU_LOTS_ID	Varchar	Código identificador del lote de salida.	X

Clave Primaria	LOTUSU_USU_ID & LOTUSU_LOTS_ID	
Clave Foránea	Campo	Tabla - Campo
	LOTUSU_USU_ID	T_USUARIO – USU_ID
	LOTUSU_LOTS_ID	T_LOTE_SAL – LOTS_ID

Cuadro 3.26: Entidad T_LOTS_USU

3.1.10.27. Entidad T_EVENTO

Esta entidad recogerá las medidas de las magnitudes en un momento dado.

Nombre	Tipo	Descripción	Obligatorio
EVT_ID	Number	Código identificador del evento. Número consecutivo	X
EVT_TUR_ID	Number	Identificador del turno que realiza el evento	X
EVT_LOC_ID	Number	Identificador de la localización donde está la unidad en el momento del evento	X
EVT_LOT_ID	Varchar	Identificador del lote sobre el que se produce el evento	X
EVT_ESTADO	Number	Indicador de si el evento ha originado una alarma por localización no permitida. Admite dos valores: ▪ 0 No ha generado alarma, valor por defecto ▪ 1 Ha generado alarma	X
EVT_DATE	Timestamp	Fecha en la que se produce el evento	X

Clave Primaria	EVT_ID	
Clave Foránea	Campo	Tabla - Campo
	EVT_LOC_ID	T_LOCALIZACION – LOC_ID
	EVT_LOT_ID	T_LOTE – LOT_ID
	EVT_TUR_ID	T_TURNO – TUR_ID

Cuadro 3.27: Entidad T_EVENTO

3.1.10.28. Entidad T_EVT_MAG

Esta entidad recoge por cada evento sobre un lote el valor de las magnitudes asociadas al mismo

Nombre	Tipo	Descripción	Obligatorio
EVTMAG_ID	Number	Código identificador del evento.	X
EVTMAG_LOT_ID	Varchar	Código identificador del lote	X
EVTMAG_MAG_ID	Number	Identificador de la magnitud	X
EVTMAG_VALOR	Number	Valor de la magnitud	X
EVTMAG_ESTADO	Number	Indicador de si el evento ha originado una alarma por valor no permitido o fuera de rango. Admite dos valores: ▪ 0 No ha generado alarma, valor por defecto ▪ 1 Ha generado alarma	X

Clave Primaria	EVTMAG_ID	
Clave Foránea	Campo	Tabla - Campo
	EVTMAG_ID	T_EVENTO – EVT_ID
	EVTMAG_LOT_ID & EVTMAG_MAG_ID	T_LOT_MAG – LOT_ID & MAG_ID

Cuadro 3.28: Entidad T_EVT_MAG

3.2. Ámbitos de aplicación posibles y líneas de trabajo abiertas

Como se podrá comprobar en el punto 3.3, la implementación llevada a cabo resulta adecuada para entornos de almacenes pequeños y medianos, donde el número de actualizaciones realizadas en la base de datos no sea muy elevado. Este es el único punto que se debe tener en cuenta a la hora de aplicar este sistema.

Por otro lado, la forma en la que se ha desarrollado el proyecto permite implementarlo en entornos ya existentes, puesto que la utilización de bases de datos

relacionales típicas es la forma que se ha definido para trabajar. Sobre esta base de datos, no hay más que definir una ontología y adaptar las operaciones que realiza la aplicación para tener un nuevo sistema funcional.

En cuanto al tipo de ámbito donde se pueda aplicar, no hay restricciones ya que en principio parece posible definir ontologías sobre cualquier tipo de negocio que funcione con almacenes.

Respecto a las líneas de trabajo abiertas, en primer lugar se ve claramente la necesidad de integrar el trabajo desarrollado en uno más amplio que aborde todos los aspectos necesarios para su funcionamiento final, como la implementación de una interfaz que permita a los usuarios su utilización, o el desarrollo de un motor de consultas semánticas para obtener información ampliada.

Además, resultaría interesante la opción de desarrollar un sistema que permita, dentro de lo posible, automatizar la adaptación del presente esquema a nuevos ámbitos, a partir de una ontología definida y especificando únicamente las relaciones existentes entre ontología y base de datos. Esto es así debido a la gran similitud de las diferentes operaciones necesarias, que no son más que adaptaciones particulares de un esquema común. Este esquema consta de los siguientes pasos:

1. Crear o recuperar la instancia deseada
2. Modificar sus propiedades de datos
3. Por cada una de las propiedades de objetos:
    a) Recuperar la instancia
    b) Definir la instancia recuperada como propiedad de objeto de la instancia de 1.

Finalmente, también resultaría de interés investigar nuevas formas, u optimizar las existentes, para que la comunicación que se realiza por RMI sea más eficiente, permitiéndose así la adaptación a sistemas más dinámicos en cuanto a utilización de bases de datos. Una opción sería modificar el funcionamiento actual de forma que la

conexión entre base de datos y aplicación sea permanente, y no tenga que realizarse por cada llamada a un procedimiento almacenado.

3.3. Análisis de resultados

Como se puede observar en la siguiente tabla, se debe tener en cuenta que la ejecución de procedimientos almacenados y las conexiones RMI que se requieren suponen un incremento considerable del tiempo necesario para realizar la acción.

Para obtener una visión más concreta de este hecho, se han realizado un número importante de inserciones automáticas consecutivas, comenzando con una cifra pequeña de 500, y acabando con 5000 inserciones. Además, para obtener una estimación intermedia, se han realizado 1000 inserciones.

	Número de inserciones				**Número de inserciones**		
Sin procedimientos	***5000***	***1000***	***500***	*Con procedimientos*	***5000***	***1000***	***500***
	3,232 s	1,188 s	0,781 s		336,033 s	32,435 s	15,734 s
	4,155 s	1,391 s	0,593 s		359,542 s	39,330 s	16,907 s
	3,957 s	1,131 s	0,469 s		375,796 s	35,077 s	18,609 s
	4,231 s	1,673 s	0,376 s		434,795 s	35,749 s	15,186 s
	4,206 s	1,345 s	0,516 s		457,231 s	36,049 s	15,486 s
	4,988 s	1,281 s	0,469 s		482,502 s	36,436 s	17,202 s
	4,452 s	1,435 s	0,562 s		525,454 s	38,325 s	18,985 s
	4,689 s	1,438 s	0,562 s		245,530 s	39,470 s	22,218 s
	5,39 s	1,655 s	0,422 s		223,316 s	40,423 s	13,613 s
	5,424 s	1,013 s	0,641 s		212,890 s	42,419 s	16,469 s

Cuadro 3.29: Comparativa de tiempos en inserciones realizadas con y sin procedimientos almacenados

Hay que aclarar que el tiempo requerido para las distintas operaciones varía enormemente dependiendo de la carga del ordenador en el que se ejecuta, habiéndose obtenido lecturas de hasta 600 segundos en el caso más extremo de 5000 inserciones. Un simple reinicio y la ejecución de menos programas ha conseguido bajar estos valores hasta los 200 segundos.

La tabla muestra claramente que la utilización de la técnica explicada en este proyecto no es adecuada para casos de bases de datos extensas con una gran actividad, o que debe ser realizada con equipos muy potentes que puedan ofrecer tiempos

de respuesta mucho menores.

En este proyecto, el almacén que se ha tenido en mente en todo momento es uno de tipo pequeño, donde la actividad de la base de datos es de aproximadamente una inserción por minuto, por lo que la infraestructura actual es más que suficiente para cumplir con las necesidades.

3.4. Planificación de tareas y actividades

En un principio, y según se presentó en el anteproyecto las fechas aproximadas de realización fueron:

- Septiembre-Octubre 2010: Estudio del estado del arte y comprensión del trabajo a realizar.
- Octubre-Noviembre 2010: Creación de una ontología basada en conocimiento de expertos humanos así como de historiales de datos.
- Noviembre 2010-Marzo 2011: Implementación del sistema.
- Marzo-Mayo 2011: Pruebas del sistema.
- Mayo-Julio 2011: Finalización del sistema.
- Julio-Noviembre 2011: Finalización de la redacción de la memoria del proyecto.
- Diciembre 2011: Presentación del proyecto.

En general, se puede decir que las fechas se han ido cumpliendo, excepto en la creación de la ontología, que no ha sido posible hacerla junto con un experto, tiempo que se ha invertido en comenzar a realizar las pruebas de las distintas técnicas del sistema. También la redacción de la memoria se ha visto atrasada, comenzando en septiembre hasta el mes de octubre.

3.5. Presupuesto

El presente proyecto se ha desarrollado en colaboración con el Instituto Ibermática de Innovación (I3B) perteneciente al grupo Ibermática.

3.5.1. Costes

Los costes más importantes son los que se refieren a la mano de obra ya que los costes materiales no son imprescindibles al poderse aprovechar la infraestructura existente.

3.5.1.1. Costes materiales

Como costes materiales se pueden tomar la compra de un servidor dedicado a mantener de forma persistente la estructura de la ontología, lo que dependerá de su complejidad y de las posibilidades que ofrezcan los servidores existentes.

Material	**Coste (euros)**
1 servidor	≈1.000
Total	**1.000**

Cuadro 3.30: Costes materiales

3.5.1.2. Mano de obra

El proyecto ha sido realizado por un futuro ingeniero informático, cuyo desglose de tareas y costes es el siguiente:

Concepto	**Duración (horas)**
Estudio de métodos de mapeo dinámico de ontologías	70
Creación de la ontología	30
Creación de la infraestructura RMI	30
Creación del interfaz con Jena	40
Creación de procedimientos almacenados	20
Creación de disparadores	30
Ensamblado de la arquitectura	35
Creación del interfaz gráfico	25
Total	**270**

Cuadro 3.31: Desglose de los costes por mano de obra

Si estimamos el sueldo medio de un ingeniero informático en 30€/hora, el coste total por mano de obra es el siguiente:

Tiempo	Coste	Total (euros)
270	30€/hora	8.100

Cuadro 3.32: Coste total por mano de obra

3.5.2. Gastos Generales

Se consideran gastos generales las amortizaciones, las cargas fiscales y jurídicas y gastos de desplazamiento, que se valoran en el 15 % del presupuesto de ejecución material.

3.5.3. Presupuesto total

El presupuesto total puede variar, tal y como se ha indicado anteriormente, dependiendo de si es necesario adquirir un nuevo servidor o no.

Concepto	Coste (euros)
Costes materiales (opcional)	≈ 1.000
Costes por mano de obra	8.100
Presupuesto de ejecución material	9.100
Gastos generales	1.365
Presupuesto de ejecución	10.465
I.V.A. (18 %)	1.883,70
Total	12.348,70

Cuadro 3.33: Presupuesto total

Capítulo 4

Conclusiones

En la presente memoria se ha intentado reflejar todo el trabajo llevado a cabo en los últimos meses para desarrollar la solución que se ha considerado como la más adecuada dentro de las posibilidades existentes. Tras realizar el análisis del trabajo realizado, la primera conclusión es positiva ya que se considera que se ha conseguido el objetivo básico que era encontrar una solución funcional y factible que permitiera mantener la coherencia entre la información contenida en una base de datos relacional y una ontología.

En cuanto al aprendizaje realizado, se puede decir que comenzó desde el principio. La búsqueda de referencias en la literatura científica ha obligado a aprender a utilizar las bases de datos de las que dispone la biblioteca de la UNED, y ha mostrado el enorme potencial que tienen las mismas, que ha servido para hacerse una idea de la cantidad de producción científica que se puede encontrar hoy en día.

La implementación de la solución ha permitido repasar y poner en práctica muchos aspectos aprendidos durante la carrera. Se han tenido que refrescar los conocimientos sobre el mecanismo de funcionamiento del RMI, sentencias SQL, el lenguaje Java...en un entorno real, lo que por un lado ha llevado a demostrar la utilidad de lo estudiado, y por otro lado ha permitido ampliar los conocimientos anteriores.

Los resultados han demostrado la viabilidad de la solución propuesta, permitiendo conocer con más detalle los límites hasta los que se puede forzar la arquitectura, y ofreciendo información para posteriores trabajos.

Es necesario repasar de nuevo la limitación del presente proyecto, teniendo que ser entendido como sólo un primer paso en la construcción de un esquema completo, que deberá incorporarse dentro de un sistema mayor teniendo que tener en cuenta las limitaciones, restricciones y necesidades impuestas.

La necesidad de realizar el trabajo en coordinación con un equipo de personas, todas ellas profesionales del sector con amplia experiencia tanto en ontologías como en logística de almacenes, localizadas en distintos puntos geográficos, ha permitido enriquecerse de sus conocimientos, y aprender cómo debe ser realizado el trabajo en grupo. En cada momento se han tenido que tomar decisiones en distintos ámbitos, lo que ha permitido ayudar a valorar qué formas de comunicación resultan más apropiadas según las necesidades existentes. Por un lado ha habido reuniones con varios asistentes para tratar temas generales, o encuentros menos numerosos para hablar de partes muy concretas de la aplicación, intercambios de correos electrónicos, llamadas de teléfono. No han sido pocas las ocasiones en las que un problema tratado en principio por medio de correos ha tenido que ir arreglándose primero con llamadas, y después con la presencia física de los implicados. En resumen, la aportación de experiencia para desenvolverse con un equipo de trabajo ha sido enorme.

Por lo tanto, se considera que la ejecución del presente proyecto ha aportado muchos aspectos positivos y necesarios en la formación de un ingeniero, y una vez acabada esta parte, analizando todos los pasos dados la sensación que queda es ampliamente positiva.

Bibliografía

[BCD05] Carlos Bento, Amílcar Cardoso, and Gaël Dias, editors. *Progress in Artificial Intelligence, 12th Portuguese Conference on Artificial Intelligence, EPIA 2005, Covilhã, Portugal, December 5-8, 2005, Proceedings*, volume 3808 of *Lecture Notes in Computer Science*. Springer, 2005.

[CB] Valerie Cross and Vishal Bathija. Automatic ontology creation using adaptation. *Artif. Intell. Eng. Des. Anal. Manuf.*, 24:127–141.

[CCLW07] H.K.H. Chow, K.L. Choy, W.B. Lee, and T.Q. Wang. A rfid based knowledge management systems - an intelligent approach for managing logistics processes. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 1, pages 287 –292, june 2007.

[CCW10] Chia-Chen Chen, Mu-Yen Chen, and Nien-Chu Wu. Development of an rfid-based management system for fashion industry. In *Service Systems and Service Management (ICSSSM), 2010 7th International Conference on*, pages 1 –6, june 2010.

[CD05] George Coulouris and Jean Dollimore. *Sistemas Distribuidos - 3b: Edicion (Spanish Edition)*. Pearson Educacion, 2005.

[CHJG+02] G. Caja, J. Hernández-Jover, J. Ghirardi, D. Garín, and J.H. Mocket. Aplicación de la identificación electrónica a la trazabilidad del ganado y de la carne. *II Seminario Internacional FUNDISA*, octubre 2002.

[CS05] Olivier Curé and Raphaël Squelbut. A database trigger strategy to maintain knowledge bases developed via data migration. In Bento et al. [BCD05], pages 206–217.

[CT09] Ruey-Shun Chen and Mengru (Arthur) Tu. Development of an agent-based system for manufacturing control and coordination with ontology and rfid technology. *Expert Syst. Appl.*, 36:7581–7593, May 2009.

[Cur] Olivier Curé. Semi-automatic data migration in a self-medication knowledge-based system.

[DKA+09] S.S. Durbha, R.L. King, S.K. Amanchi, S. Bheemireddy, and N.H. Younan. Information services and middleware for the coastal sensor web. In *Data Mining Workshops, 2009. ICDMW '09. IEEE International Conference on*, pages 656 –661, dec. 2009.

[dSW97] C.W. de Silva and N. Wickramarachchi. An intelligent supervisory control system for a fish processing workcell. In *Knowledge-Based Intelligent Electronic Systems, 1997. KES '97. Proceedings., 1997 First International Conference on*, volume 2, pages 470 –477 vol.2, may 1997.

[DVS+06] Hai Deng, M. Varanasi, K. Swigger, O. Garcia, R. Ogan, and E. Kougianos. Design of sensor-embedded radio frequency identification (serfid) systems. In *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*, pages 792 –796, june 2006.

[Gar08] Luis Ruiz García. Development of monitoring applications for refrigerated perishable goods transportation. 2008.

[GMR08] I. Gil, A. Mollá, and M. E. Ruiz. Automatización del almacén y surtido en la distribución de productos de uso duradero. *Universia Business Review*, (19):118–133, tercer trimestre 2008.

[GMRS07] F. Gandino, B. Montrucchio, M. Rebaudengo, and E.R. Sanchez. Analysis of an rfid-based information system for tracking and tracing in an agri-food chain. In *RFID Eurasia, 2007 1st Annual*, pages 1 –6, sept. 2007.

[GS92] M.B. Green and P.R. Sims. An adaptable expert aid to fault diagnosis in satellite communication networks. In *Information-Decision-Action Systems in Complex Organisations, 1992., International Conference on*, pages 153 –157, apr 1992.

[Hin94] D.J. Hind. Radio frequency identification and tracking systems in hazardous areas. In *Electrical Safety in Hazardous Environments, 1994., Fifth International Conference on*, pages 215 –227, apr 1994.

[ISL+] Peter Ibach, Vladimir Stantchev, Florian Lederer, Andreas Weiß, Thomas Herbst, and Torsten Kunze. Wlan-based asset tracking for warehouse management 1 abstract.

[Jim08] A. Jiménez. Inteligencia artificial ii. *Dpto. de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla*, 2008.

[JJM03] J. Jarvis, D. Jarvis, and D. Mcfarlane. Achieving holonic control–an incremental approach. *Computers in Industry*, 51(2):211–223, June 2003.

[JSF09] Dietmar Jannach, Kostyantyn Shchekotykhin, and Gerhard Friedrich. Automated ontology instantiation from tabular web sources-the allright system. *Web Semant.*, 7:136–153, September 2009.

[JYY08] Ha Jinbing, Wei Youna, and Jin Ying. Logistics decision-making support system based on ontology. In *Computational Intelligence and Design, 2008. ISCID '08. International Symposium on*, volume 1, pages 309 –312, oct. 2008.

[KSM08] Nikolaos Konstantinou, Dimitrios-Emmanuel Spanos, and Nikolas Mitrou. Ontology and database mapping: a survey of current implementations and future directions. *J. Web Eng.*, 7:1–24, March 2008.

[LCR+10] J. Lagares, J. M. Cassany, A. Rodriguez, D. Borrell, and M. Xargayó. Identificación por radiofrecuencia: inteligencia rentable para el sector cárnico. *Eurocarne: La revista internacional del sector cárnico*, (185):126–132, 2010.

[LPK07] Peng Lian, Dae-Won Park, and Hyuk-Chul Kwon. Design of logistics ontology for semantic representing of situation in logistics. In *Digital Media and its Application in Museum Heritages, Second Workshop on*, pages 432 –437, dec. 2007.

[Mcf] Mcfarlane. The intelligent product in manufacturing control and management.

[MMV05] D. McFarlane, V. Marik, and P. Valckenaers. Guest editors' introduction: Intelligent control in the manufacturing supply chain. *Intelligent Systems, IEEE*, 20(1):24 – 26, jan.-feb. 2005.

[NFB01] G. Nucci Franco and A. Batocchio. Towards an axiomatic framework to support the design of holonic systems. In *Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on*, pages 654 –659, 2001.

[NFX09] Guihua Nie, Meng Fu, and Huan Xia. A semantic mapping system based on e-commerce logistics ontology. In *Software Engineering, 2009. WCSE '09. WRI World Congress on*, volume 2, pages 133 –136, may 2009.

[Ora] Oracle. Java stored procedures developer's guide. `http://download.oracle.com/docs/cd/B10501_01/java.920/a96659.pdf`.

[SJR+] Joshua R. Smith, Bing Jiang, Sumit Roy, Matthai Philipose, Kishore Sundara-rajan, and Er Mamishev. Id modulation: Embedding sensor data in an rfid timeseries. In *Matthai Philipose, Kishore Sundara-Rajan, Alexander Mamishev. Proceedings of Information Hiding 2005, LNCS 3727*, pages 234–246.

[TM09] Mitja Trampus and Dunja Mladenic. Constructing event templates from written news. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, volume 3, pages 507 –510, sept. 2009.

[vBS74] Ludwig von Bertalanffy and John W. Sutherland. General systems theory: Foundations, developments, applications. *Systems, Man and Cybernetics, IEEE Transactions on*, 4(6):592, nov. 1974.

[Wan06] R. Want. An introduction to rfid technology. *Pervasive Computing, IEEE*, 5(1):25 – 33, jan.-march 2006.

[WMAZA02] C.Y. Wong, D. McFarlane, A. Ahmad Zaharudin, and V. Agarwal. The intelligent product driven supply chain. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 4, page 6 pp. vol.4, oct. 2002.

Nomenclatura

API Application Programming Interface

D2RQ Database To RDF Query

DBOM DataBase Ontology Mapping

IP Internet Protocol

JVM Java Virtual Machine

OWL Ontology Web Language

PDA Personal Digital Assistant

RDB Relational DataBase

RDF Resource Description Framework

RFID Radio Frequency IDentification

RMI Remote Method Invocation

SPARQL SPARQL Protocol and RDF Query Language

SQL Sctructured Query Language

Índice alfabético