BELÉN BETZAIDA BONILLA MORALES

SISTEMA PARA LA REUTILIZACIÓN DE DIAGRAMAS DE CASOS DE USO BASADO EN ONTOLOGÍAS Y TECNOLOGÍAS DE WEB SEMÁNTICA

Universidad Tecnológica de Panamá
Facultad de Ingeniería de Sistemas Computacionales
Maestría en Ciencias de Tecnología de Información y Comunicaciones
con énfasis en Ingeniería de Software

Panamá
2012

II

BELÉN BETZAIDA BONILLA MORALES

SISTEMA PARA LA REUTILIZACIÓN DE DIAGRAMAS DE CASOS DE USO BASADO EN ONTOLOGÍAS Y TECNOLOGÍAS DE WEB SEMÁNTICA

Tesis presentada a la Facultad de Ingeniería de Sistemas Computacionales de la Universidad Tecnológica de Panamá para la obtención del Título de

Magíster en
Ciencias de Tecnología de Información y Comunicaciones con énfasis en Ingeniería de Software

Asesores:
Dr. Sergio Crespo Coelho da Silva Pinto
Dr. Clifton Clunie B.

Panamá
2012

A mi inigualable familia y a mi amado Arcadio, quienes me acompañaron y apoyaron durante este largo camino.

Agradecimientos

Mi agradecimiento a Dios en primera instancia por darme la oportunidad de vivir esta experiencia y por estar siempre a mi lado protegiéndome y dándome fuerzas para seguir adelante.

A mi novio, Arcadio E. Quintero A., quien siempre estuvo a mi lado apoyándome, brindándome toda su ayuda y dándome ánimos para seguir adelante durante todo este período de maestría. Sin su ayuda esto habría sido muy difícil.

A mi madre, a mi padre y a mis hermanos porque en cada paso de mi vida me han apoyado y orientado, han estado conmigo en las buenas y en las malas, y jamás han dejado de creer en mí.

A mi abuela Adela por cada una de sus oraciones, las cuales sin duda alguna me tienen hoy en día donde estoy.

A mi tutor Dr. Sérgio Crespo por su dedicación y gran experiencia, y por brindarme la ayuda necesaria para llevar a cabo esta tesis. Y a mi co-tutor Dr. Clifton Clunie por su constante apoyo y ayuda a lo largo de la maestría.

A la Secretaría Nacional de Ciencia, Tecnología e Innovación (SENACYT) y a la Universidad Tecnológica de Panamá, especialmente a la Facultad de Ingeniería de Sistemas Computacionales, por brindarme la oportunidad de participar de este programa de maestría.

"Lo que sabemos es una gota de agua; lo que ignoramos es el océano".
(Isaac Newton)

Resumen

Este trabajo presenta el desarrollo de un sistema para la búsqueda y recuperación de diagramas de casos de uso utilizando ontologías OWL y tecnologías de Web Semántica. La idea consiste en crear una ontología OWL base que permita almacenar y estructurar la información que se obtenga de los diagramas de casos de uso de manera tal que se puedan realizar catalogaciones de los diagramas y que las consultas que se lleven a cabo sobre la misma, posean características semánticas. La herramienta de apoyo para la búsqueda y recuperación es implementada a través del framework Jena, el cual permite crear aplicaciones semánticas, y las consultas que se realizan sobre la ontología, almacenada de forma persistente en una base de datos MySQL, se realizan a través de SPARQL el cual es el lenguaje de consultas más popular de la Web Semántica. Este sistema propicia una reutilización de diagramas de casos de uso lo cual aporta beneficios en cuanto a disminución de tiempos, costos y esfuerzo durante la fase de análisis y especificación de requisitos de un software.

Palabras Claves: *ontología, Web Semántica, reutilización, OWL*

Abstract

This work presents the development of a system for searching and retrieving use case diagrams using OWL ontologies and Semantic Web technologies. The idea is create an OWL ontology base that lets us store and organize the information obtained from the use case diagrams so that we can make cataloging of the diagrams and that queries that are carried out on it, possess semantic features. The search and retrieval support tool is implemented through the Jena framework, which allows us to create semantic applications, and queries that are made on the ontology, which is persistently stored in a MySQL database, are performed through SPARQL which is the most popular query language for the Semantic Web. This system facilitates the reuse of use case diagrams which brings benefits in terms of reduced time, cost and effort during the requirements analysis and specification of software.

Keywords: *ontology, Semantic Web, reuse, OWL*

Índice general

Lista de Tablas

Pág.

Lista de Figuras

Lista de Siglas

RDF Resource Description Language

OWL Ontology Web Language

UML Unified Modeling Laguage

XML Extensible Markup Language

XMI XML Interchange Language

W3C World Wide Web Consortium

KIF Knowledge Interchange Language

RDFS RDF Schema

OKBC Open Knowledge Base Connectivity

GFP Generic Frame Protocol

FO Frame Ontology

OCML Operational Conceptual Modeling Language

HTML HyperText Markup Language

DAML DARPA Agent Markup Language

OIL Ontology Interchange Language

URI Uniform Resource Identifier

Capítulo 1

Introducción

Este capítulo realiza una introducción al trabajo de investigación. En este sentido se presenta el planteamiento del problema en la sección 1.1. En la sección 1.2 se expone la justificación del trabajo evidenciando el objeto de la investigación. En la sección 1.3 se definen los objetivos general y específicos. Por su parte, la sección 1.4 establece el alcance del trabajo definiendo puntualmente las limitantes del mismo. Finalmente, la sección 1.5 define y explica la metodología empleada para el desarrollo de este trabajo de tesis.

1.1. Planteamiento del Problema

Los diagramas de casos de uso son un tipo de diagrama UML que permiten definir las funcionalidades de un sistema a través de la captura de los requisitos potenciales del mismo. Cada diagrama de caso de uso tiene como finalidad definir la forma como el sistema interactúa con cada usuario o con otros sistemas para alcanzar un objetivo en específico. Son estructuras que ayudan a los analistas a trabajar con los usuarios para determinar la forma en que se usará un sistema.

Con una colección de diagramas de casos de uso se puede hacer el bosquejo de un sistema en términos de lo que los usuarios intentan hacer con él. Esta actividad es crucial para la fase de análisis y especificación de requisitos en el desarrollo de un

software. De esta manera, la importancia de los diagramas de casos de uso radica en que se diseña el sistema desde el propio punto de vista del usuario, siendo la idea principal, involucrar a los usuarios en la etapa inicial de análisis del sistema.

La claridad y precisión en la definición y modelado de los requisitos de un software a través de las diferentes técnicas y diagramas, entre ellos los diagramas de casos de uso, elevan las probabilidades de obtener productos de software de calidad. Sin embargo, el análisis y modelado de requisitos son tareas muy difíciles a las cuales se enfrenta un ingeniero o analista de software y la mayoría de las veces demandan gran cantidad de su tiempo y esfuerzo.

En la Ingeniería de Software, los factores tiempo, costo y calidad son variables de suma importancia ya que definen el éxito o fracaso de un proyecto de software, esto debido a que el propósito es garantizar la calidad del software sin exceder los límites de las variables costo y tiempo. Luego, muchas prácticas que llevan a cabo los equipos de desarrollo de software durante la etapa de análisis y modelado pueden establecer la diferencia entre el ahorro de tiempo, costo y esfuerzo humano, y/o el malgasto de los mismos en detrimento de la calidad del software. Por ejemplo, en muchas ocasiones, a pesar de que dos o más sistemas de software poseen funcionalidades en común, los diagramas de casos de uso correspondientes a las mismas son elaborados por los ingenieros de software cada vez que son requeridos por un nuevo software; esta situación podría considerarse como un malgasto de tiempo y esfuerzo puesto que se estaría realizando una tarea que ya fue realizada con anterioridad, en otro proyecto de software.

Bajo esta condición, surge la reutilización de artefactos de software, la cual permite obtener una disminución y mejora en los tiempos en que se lleva a cabo el análisis y modelado de requisitos, a la vez que se reducen los costos de desarrollo y aumenta la calidad de los productos de software.

Es de gran importancia, entonces, contar con infraestructuras y aplicaciones que permitan llevar a cabo reutilización de artefactos de software. Sin embargo, específicamente para la etapa de análisis de requisitos donde tienen lugar los diagramas de casos de uso, son muy pocos los trabajos de apoyo a la reutilización que se han propuesto y desarrollado.

1.2. Justificación de la Investigación

Debido a que en la actualidad son pocas las aplicaciones o herramientas que apoyan la reutilización de artefactos de software creados durante la fase de análisis de requisitos en el desarrollo de software y más escaso aún, para la reutilización de diagramas de casos de uso, resulta necesario proponer, definir e implementar soluciones que disminuyan esta carencia de una manera eficaz.

Sin embargo, vale la pena que dichas soluciones utilicen tecnologías recientes que permitan, entre otras cosas, una fácil integración con otras tecnologías y aplicaciones, mayor escalabilidad y eficiencia; además, que agreguen un valor extra de innovación a dichas soluciones.

Es por esto que, a través de este trabajo, se propone explorar el uso de ontologías y tecnologías de Web Semántica para la implementación de un sistema que permita reutilizar diagramas de casos de uso a través del almacenamiento, búsqueda y recuperación semántica de los mismos.

Este sistema permitiría a los ingenieros de software almacenar la información de los diagramas de casos de uso en una ontología, lo cual facilitaría realizar búsquedas semánticas sobre un repositorio y garantizar que los resultados obtenidos concuerden con las solicitudes ingresadas por el usuario debido al manejo semántico con que se establecerían las consultas. De esta manera, los equipos de desarrollo de software pueden crear su base de datos de diagramas de casos de uso y contar con una aplicación semántica que les permita buscarlos y recuperarlos, y finalmente reutilizarlos en los nuevos proyectos de software propiciando con esto, un ahorro de tiempo y esfuerzo durante la etapa de análisis de requisitos.

1.3. Objetivos

1.3.1. Objetivo General

- Diseñar e implementar un sistema de software que permita la reutilización de diagramas de casos de uso en UML a través del uso de ontologías y tecnologías

de Web Semántica.

1.3.2. Objetivos Específicos

- Diseñar una ontología que permita almacenar y manipular la información de diagramas de casos de uso.
- Crear una base de datos ontológica que brinde almacenamiento persistente a la ontología y sus individuos.
- Implementar un prototipo de interfaces para facilitar el registro, búsqueda y recuperación de diagramas de casos de uso.
- Realizar pruebas de funcionalidad sobre el sistema de reutilización de diagramas de casos de uso.
- Consolidar los resultados obtenidos en un documento técnico.

1.4. Alcance

El alcance de este trabajo se define a través de las siguientes pautas:

- Este trabajo de investigación se enfoca en el desarrollo e implementación de un sistema de software que permite la reutilización de diagramas de casos de uso.
- Cabe destacar que el sistema sólo estará diseñado para trabajar con diagramas de casos de uso, no así con los demás diagramas UML.
- Este trabajo no involucra la validación lógica previa de los diagramas de casos de uso que se ingresan al sistema sólo la validación, en cuanto a estructura de lenguaje, de los archivos XMI que se generan.
- La sistema trabaja con archivos XMI generados desde cualquier herramienta de modelado UML siempre y cuando el formato de tales archivos sea estándar.

- La herramienta de apoyo al sistema funciona como una aplicación de escritorio y permite el acceso a la base de datos y ficheros en una infraestructura cliente-servidor.

- Los archivos XMI que son descargados a través de la herramienta pueden ser abiertos utilizando cualquier software de modelado UML, siempre y cuando la misma permita la importación de XMI. Cabe destacar que no todos estos software de modelado despliegan el diagrama de forma visual, no obstante, cargan los elementos que componen el diagrama de caso de uso.

- Las pruebas realizadas al sistema están orientadas a validar la funcionalidad de la misma y eficiencia en cuanto a resultados obtenidos; no obstante, no se realizan validaciones en cuanto a tiempo de respuesta con respecto a otras herramientas o soluciones similares.

- Las pruebas al sistema se ejecutan sobre escenarios simulados ya que por cuestiones de tiempo no se pudieron realizar sobre escenarios reales.

1.5. Metodología de Trabajo

Para lograr los objetivos propuestos en esta investigación se hace necesario contar con una metodología de trabajo, la cual puede ser caracterizada como sigue:

- En primer lugar, se definirá el problema de investigación con base a las necesidades actuales en la rama de Ingeniería de Software. De esta manera queda entendido lo que se quiere resolver.

- Luego de definido y analizado el problema, se realizará una investigación y levantamiento bibliográfico sobre los estudios y trabajos previos que abordan y brindan solución a esta problemática de forma tal que se puedan identificar las fortalezas y debilidades de cada uno y de esta forma proponer una nueva solución basada en nuevas tecnologías. También se procederá a llevar a cabo una revisión bibliográfica y documentación sobre las tecnologías y software a utilizar para el desarrollo e implementación de la solución.

- A partir de la revisión bibliográfica y trabajos previos relacionados, se procederá a definir un modelo y arquitectura de software que permita la reutilización de diagramas de casos de uso utilizando ontologías y tecnologías de Web Semántica.
- Una vez definido el modelo y arquitectura de la solución propuesta, se creará una herramienta que implemente el modelo y arquitectura definida utilizando ontologías y tecnologías de Web Semántica.
- Se llevarán a cabo pruebas de funcionalidad de la herramienta y validaciones de eficiencia.

Capítulo 2

Fundamentos Teóricos

2.1. Diagramas de Casos de Uso

2.1.1. Concepto

Los diagramas de casos de uso son un tipo de diagrama UML que definen las secuencias de interacciones entre un sistema y alguien o algo que usa uno de sus servicios. Especifican el comportamiento de un sistema con respecto a los usuarios y definen en términos sencillos sus requerimientos funcionales (Pressman et al., 2006).

Un diagrama de casos de uso es una forma de expresar cómo alguien o algo externo a un sistema lo usa. Cuando se dice “alguien o algo” hacemos referencia a que los sistemas son usados no sólo por personas, sino también por otros sistemas de hardware y software.

Es importante resaltar que los diagramas de casos de uso no están pensados para representar el diseño y no pueden describir los elementos internos de un sistema. Los diagramas de casos de uso sirven para facilitar la comunicación con los futuros usuarios del sistema, y con el cliente, y resultan especialmente útiles para determinar las características necesarias que tendrá el sistema. En otras palabras, los diagramas de casos de uso describen qué es lo que debe hacer el sistema, pero no cómo (Övergaard y Palmkvist, 1999).

La Figura 2.1 muestra una representación de un diagrama de casos de uso.

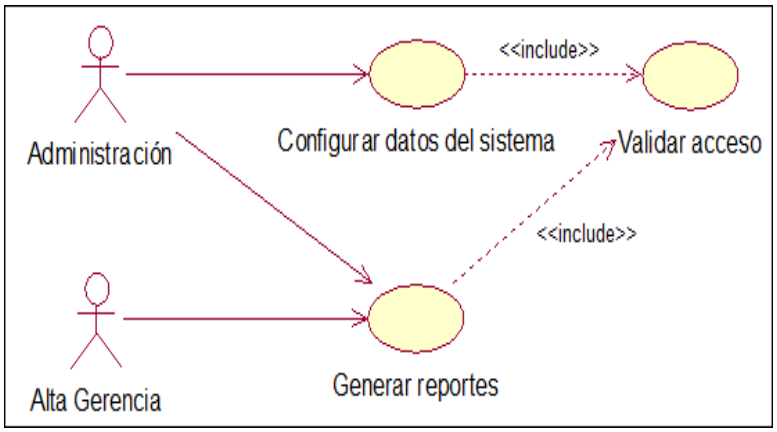


Figura 2.1: Representación gráfica de un diagrama de casos de uso

2.1.2. Elementos

Actores

Un actor es una entidad externa (de fuera del sistema) que interacciona con el sistema participando (y normalmente iniciando) en un caso de uso. Los actores pueden ser personas, otros ordenadores o eventos externos.

Los actores no representan a personas físicas o a sistemas, sino su rol. Esto significa que cuando una persona interactúa con el sistema de diferentes maneras (asumiendo diferentes roles), estará representado por varios actores. Las diferentes funciones que el actor representa son las funciones de negocio reales de los usuarios en un sistema dado. Por ejemplo, una persona que proporciona servicios de atención telefónica a clientes y realiza pedidos para los clientes estaría representada por un actor «equipo de soporte» y por otro actor «representante de ventas».

Los actores se representan con dibujos simplificados de personas, tal como se muestra en la Figura 2.2.

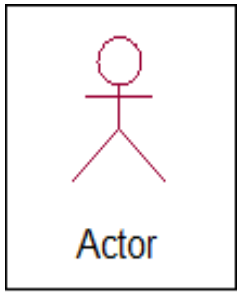


Figura 2.2: Representación gráfica de un actor

Casos de Uso

Un caso de uso describe, desde el punto de vista de los actores, una funcionalidad de negocio de un sistema que produce un resultado concreto y tangible. Capta una función visible para el usuario. Es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso dentro del sistema.

Gráficamente, los casos de uso se representan con un óvalo con el nombre del caso, tal como se muestra en la Figura 2.3.

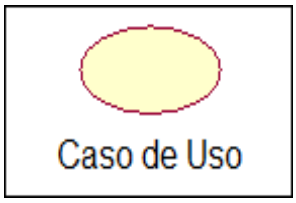


Figura 2.3: Representación gráfica de un caso de uso

Un caso de uso tiene las siguientes características:

- Está expresado desde el punto de vista del actor.
- Se documenta con texto informal.
- Describe tanto lo que hace el actor como lo que hace el sistema cuando interactúa con él, aunque el énfasis está puesto en la interacción.
- Está acotado al uso de una determinada funcionalidad, claramente diferenciada, del sistema.

Relaciones

Entre los casos de uso se pueden presentar tres tipos de relaciones.

a) Relación de extensión: Se dice que el caso de uso A extiende el caso de uso B si dentro de B se ejecuta A cuando se cumple una condición determinada. A tiene que ser un caso de uso que también se pueda ejecutar de forma separada de B, y debe tener el mismo actor primario que éste.

b) Relación de inclusión: Un caso de uso A está incluido dentro de los casos de uso B, C, etc. si es una parte de proceso común a todos éstos. A no es un caso de uso autónomo, en el sentido de que no tendrá actor primario, sino que siempre será puesto en funcionamiento por uno u otro de los casos de uso que lo incluyen. No obstante, su implementación no puede depender de éstos (por ejemplo, no puede utilizar sus variables). Por lo tanto, la inclusión de casos de uso es esencialmente una forma de reutilización.

c) Relación de generalización: Un caso de uso A es una especialización de otro caso de uso B si A realiza todo el proceso de B, más algún proceso específico.

2.1.3. Importancia

La técnica de diagramas de casos de uso tiene gran importancia en sistemas interactivos, ya que expresa la intención que tiene el actor (su usuario) al hacer uso del sistema.

Como técnica de extracción de requerimiento permite que el analista se centre en las necesidades del usuario, qué espera éste lograr al utilizar el sistema, evitando que las personas especializadas en informática dirijan la funcionalidad del nuevo sistema basándose solamente en criterios tecnológicos.

A su vez, durante la extracción de requerimientos, el analista se concentra en las tareas centrales del usuario describiendo por lo tanto los casos de uso que mayor valor aportan al negocio. Esto facilita luego la priorización del requerimiento.

2.2. Ontologías

2.2.1. Concepto

El término Ontología proviene de la filosofía, la cual lo define como una teoría que trata de la naturaleza y organización de la realidad, es decir de lo que “existe”. Conocimiento del ser (del griego onto: ser y logos: conocimiento).

La definición de ontología más concisa, aplicada al área de la informática, es la del consorcio W3C: “Ontología es un término que ha sido tomado prestado de la filosofía para referirnos a la ciencia de describir los tipos de entidades en el mundo y cómo se relacionan entre ellos.”

Una ontología define formalmente un conjunto común de términos que son usados para describir y representar un dominio o área de conocimiento (Heflin, 2010).

Hay varios términos que necesitan se aclarados en esta última definición. En primer lugar, un dominio es simplemente un área específica o área de conocimiento como lo es el área de la medicina, bienes raíces y educación. Segundo, una ontología contiene términos y relaciones entre esos términos. Los términos son a menudo llamados clases o conceptos, y estas palabras son intercambiables. Las relaciones entre estas clases pueden ser expresadas utilizando una estructura jerárquica: las superclases representan conceptos de alto nivel y las subclases, conceptos más específicos. Tercero, a lado de las relaciones anteriores entre las clases, hay otro nivel de relación expresado a través de un grupo especial de términos: propiedades. Estas propiedades describen varias características y atributos de los conceptos, y pueden ser usados también para asociar clases diferentes. Por lo tanto, las relaciones entre las clases no son solo del tipo superclases y subclases, sino también las relaciones expresadas en términos de propiedades.

Al contar con los términos y las relaciones entre estos términos claramente definidos, una ontología codifica el conocimiento del dominio, de tal manera que dicho conocimiento puede ser entendido por una computadora. Esta es la idea principal de una ontología.

Otra definición que se le ha dado a el término ontología, en el área de computación,

es la siguiente: “Una ontología es una especificación formal y explícita de una conceptualización compartida” (Gruber, 1993). El término conceptualización se refiere a un modelo abstracto de algún fenómeno en el mundo al haber identificado los conceptos relevantes de ese fenómeno. Explícito significa que el tipo de los conceptos utilizados, y las restricciones sobre su uso, se definen explícitamente. Formal se refiere al hecho de que la ontología debería ser legible por máquinas. Compartida refleja la idea de que una ontología captura conocimiento consensual, es decir, que no es privado de algún individuo, sino aceptado por un grupo.

Las ontologías son una herramienta para compartir información y conocimiento, es decir, para conseguir la interoperabilidad. Al definir un vocabulario formal de los conceptos del dominio y un conjunto de relaciones entre ellos, permite que las aplicaciones “comprendan” la información.

2.2.2. Componentes

Las ontologías comparten muchas similitudes estructurales, independientemente del lenguaje en el cual se expresen. De esta manera, según Corcho et al. (2006) las ontologías tienen los siguientes componentes que servirán para representar el conocimiento de algún dominio:

a) Clases: Representan los conceptos o ideas básicas que se intentan formalizar. Las clases en una ontología son usualmente organizadas en taxonomías en las cuales los mecanismos de herencia pueden ser aplicados.

b) Relaciones: Representan las interacciones y enlaces entre los conceptos del dominio. Suelen formar la taxonomía del dominio. Las ontologías usualmente contienen relaciones binarias. El primer argumento es conocido como el dominio y el segundo argumento como el rango.

c) Funciones: Son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.

d) Instancias o Individuos: Se utilizan para representar objetos o elementos determinados de una clase.

e) Axiomas o Reglas: Son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Permiten, junto al mecanismo de la herencia de conceptos, inferir conocimiento que no esté indicado explícitamente en la taxonomía de conceptos.

2.2.3. Clasificación

Las ontologías se pueden clasificar teniendo en cuenta diferentes criterios. Las divisiones propias de cada clasificación pueden coincidir o parecerse a las establecidas por otra clasificación.

Algunas de las clasificaciones más relevantes se explican a continuación.

Según la granularidad de su conceptualización

Esta clasificación fue propuesta por van Heijst et al. (1997) y divide las ontologías como sigue:

a) Terminológicas: Especifican los términos que son usados para representar conocimiento en el universo de discurso. Suelen usarse para unificar vocabulario en un dominio determinado (contenido léxico y no semántico).

b) De información: Especifican la estructura de almacenamiento de bases de datos. Ofrecen un marco para el almacenamiento estandarizado de información (estructura de los registros de una base de datos).

c) De modelado de conocimiento: Especifican conceptualizaciones del conocimiento. Poseen una rica estructura interna y suelen estar ajustadas al uso particular del conocimiento que describen (términos y semántica).

Según el tema de conceptualización o su contenido

Guarino (1998) propuso esta clasificación que divide las ontologías de la siguiente manera:

a) Ontologías de alto nivel: Describen conceptos muy generales que son independientes de un problema o dominio particular (tiempo, espacio, materia, evento o acción).

b) De dominio y tarea: Estas ontologías describen, respectivamente, el vocabulario relacionado a un dominio genérico (como medicina, o automóviles) o una tarea genérica o actividad a través de la especialización de los términos introducidos en la ontología de alto nivel.

c) De aplicación: Contienen todas las definiciones que son necesarias para modelar el conocimiento requerido para una aplicación en particular. Describen conceptos dependiendo de un dominio y/o tarea en particular.

Según el alcance de su aplicabilidad

Por su parte Mizoguchi et al. (1995) propuso las siguientes divisiones:

a) De dominio: Proporcionan el vocabulario necesario para describir un dominio dado o subdominio, como la medicina, el tráfico, el sector mobiliario. Incluyen términos relacionados con:

   - Los objetos del dominio y sus componentes.
   - Un conjunto de verbos o frases que dan nombre a actividades y procesos que tienen lugar en ese dominio.
   - Conceptos primitivos que aparecen en teorías, relaciones y fórmulas que regulan o rigen el dominio.

b) Generales: Representan conceptos generales que no son específicos de un dominio. Por ejemplo, ontologías sobre el tiempo, el espacio, ontologías de conducta, de causalidad, etc.

c) De tareas: Proporcionan el vocabulario para describir términos involucrados en los procesos de resolución de problemas los cuales pueden estar relacionados con tareas similares en el mismo dominio o en dominios distintos. Incluyen nombres, verbos, frases y adjetivos relacionados con la tarea.

2.2.4. Lenguajes

Existen multitud de lenguajes que permiten la representación de ontologías. No todos ellos permiten el mismo nivel de expresividad a la ontología construida ni tampoco ofrecen las mismas funcionalidades.

A la hora de elegir un lenguaje para la definición de una ontología deben tenerse en cuenta los siguientes aspectos:

- El lenguaje debe poseer una sintaxis bien definida para poder "leer" con facilidad la ontología definida.
- Debe tener una semántica bien definida para comprender perfectamente el funcionamiento de la ontología.
- Debe tener suficiente expresividad para poder capturar varias ontologías.
- Debe ser fácilmente mapeable desde/hacia otros lenguajes ontológicos.
- Debe ser eficiente a la hora de realizar razonamiento.

Se pueden clasificar los diferentes lenguajes en dos grandes grupos: lenguajes tradicionales y lenguajes basados en estándares y en Web.

Lenguajes Tradicionales

Los lenguajes de ontologías tradicionales se distinguen según la forma en la que se basan para representar el conocimiento: basado en frames, en lógica descriptiva, predicados de primer orden y segundo orden, o los orientados a objetos.

Entre los más representativos se encuentran:

a) Ontolingua (Farquhar et al., 1996): Fue desarrollado en 1992 por KSL (Universidad de Standford), es un lenguaje de ontologías basado en KIF y en FO empleado en el Ontolingua Server. KIF permite definir objetos, relaciones y funciones y

utiliza predicados de lógica de primer orden. Como KIF es un lenguaje no orientado a construir ontologías, sino para intercambio de conocimiento, Ontolingua emplea FO para permitir la descripción de ontologías utilizando los paradigmas de frames, y con el que empezaron a surgir términos como clase, instancia o subclase. FO no permitía axiomas, pero al surgir a partir de KIF, sí permite que se incluyan axiomas de KIF dentro de sus definiciones.

b) Protocolo OKBC (Chaudhri et al., 1998): Como su nombre indica no se trata de un lenguaje sino un protocolo basado en el GFP , es decir, basado en frames. Es utilizado como complemento de lenguajes de representación de conocimiento, y permite, como otros sistemas basados en frames, clases, constantes, atributos, frames e instancias que sirven de base de conocimiento. También implementa una interfaz para acceder al conocimiento al igual que funciones utilizadas para acceder a través de la red a un conocimiento compartido. Fue empleado junto a Ontolingua.

c) OCML (Motta, 1998): Fue desarrollado originalmente como parte del proyecto VITAL. Es un lenguaje basado en marcos que aporta mecanismos para expresar ítems tales como relaciones, funciones, reglas, clases e instancias. Se le añadió un módulo de mecanismos de lógica y una interfaz para poder interactuar con el conocimiento. Una de las ventajas que tiene es que es compatible con estándares como Ontolingua.

d) F-logic (Balaban, 1995): Fue desarrollado en 1995 en la Universidad de Karsruhe. Integra marcos y calcula predicados de primer orden. Permite la representación de conceptos, taxonomías, relaciones binarias, funciones, instancias, axiomas y reglas deductivas. Tiene una estructura con aspectos parecidos a la los lenguajes orientados a objetos, como la herencia, tipos polimórficos, etc. Con Flogic se han creado de bases de datos hasta ontologías y se puede combinar con otros programas de lógica para interactuar con la información almacenada en la ontología.

Lenguajes basados en estándares y en Web

Estos lenguajes fueron desarrollados específicamente para el desarrollo de ontologías. Se basan en los lenguajes de marcado HTML y XML. Se diferencian de los

anteriores porque fueron desarrollados para ser utilizados en la web. Dado que la web es mucho más extensa que una base de conocimiento, es necesario el uso de estándares que permitan implementar ontologías en este ambiente.

Los lenguajes más destacados se definen a continuación:

a) RDF (Manola y Miller, 2004) y RDFS (Guha y Brickley, 2004): RDF fue desarrollado por la W3C con el objetivo de "especificar semántica, para los datos basados en XML, de una manera interoperable y estandarizada".

   RDF Schema es un vocabulario para describir las propiedades y las clases de los recursos RDF, con una semántica para establecer jerarquías de generalización entre dichas propiedades y clases.

   Más adelante se profundizará más sobre RDF y RDF Schema.

b) DAML+OIL (McGuinness et al., 2002): El programa DAML es una iniciativa de DARPA (Defense Advance Research Projects Agency) en 1999 con el objetivo de proveer los fundamentos de la Web semántica.

   Los usuarios de RDF y RDFS conforme utilizaban estos lenguajes para expresar los metadatos de sus recursos, observaban que ambos lenguajes describían un conjunto escaso de facilidades para expresar estos metadatos. La comunidad de usuarios vieron en RDF una herramienta para expresar sus recursos, pero también lamentaban que no existiesen facilidades para permitir la inferencia sobre las descripciones RDF. Por estos motivos, unieron sus esfuerzos los desarrolladores de DAML con los de OIL para favorecerse de los sistemas de clasificación basados en frames de este último. El resultado final fue el lenguaje DAML+OIL, un lenguaje que a pesar de tener muchas coincidencias con OIL, también tiene sus diferencias, la principal es que se trata de un lenguaje que se aleja de los ideales de frames de OIL, para acercarse más a una base de lenguajes de lógica descriptiva.

   Por lo tanto, DAML+OIL es un lenguaje que tiene una semántica sencilla y bien definida, y que nos ofrece más expresiones sofisticadas para las descripciones de clasificaciones y propiedades de los recursos que las que ofrecía RDF y RDFS.

c) OWL (McGuinness y van Harmelen, 2004): Surge en la W3C por la búsqueda de un lenguaje de especificación de ontologías que sirva como estándar para todos

los investigadores de la Web semántica. Deriva del lenguaje DAML + OIL y se construye sobre la sintaxis de RDF/XML.

El lenguaje OWL será explicado con mayor detalle, más adelante en este capítulo.

2.2.5. Metodologías y Métodos de Construcción

Para construir una ontología se requiere de una serie de actividades orientadas a su desarrollo. Las metodologías proporcionan un conjunto de directrices que indican cómo hay que llevar a cabo las actividades identificadas en el proceso de desarrollo, qué técnicas son las más apropiadas en cada actividad y qué produce cada una de ellas.

Hay dos métodos principales que nos permiten diferenciar dos tipos de ontologías según su construcción:

- Kactus: Es un método de construcción de ontologías que se basa en tomar una base de conocimiento y, a partir de ésta, determinar y conceptualizar cuales son los términos y relaciones más importantes que representaran a la ontología (Bernaras et al., 1996).
- Sensus: Es un método que representa a las ontologías construidas a partir de una rama de una ontología más general y que es especializada para obtener una ontología nueva (Swartout et al., 1996). Es decir, consiste en crear ontologías específicas de dominio a partir de una ontología más general.

Las principales metodologías para construir ontologías son:

a) Metodología TOVE: Se utilizó para construir la ontología TOVE, acerca de los procesos de modelado de una empresa (Grüninger y Fox, 1995). Está basada en la identificación de escenarios y la formulación de preguntas de competencia (Competency Questions), extracción de conceptos y relaciones relevantes y la formalización en Lógica de Primer Orden.

b) Metodología ENTERPRISE: Utilizada para construir la ontología ENTERPRISE, también sobre procesos de modelado de empresa (Uschold y King, 1995). Se basa en identificar el propósito para posteriormente construir, evaluar y documentar las ontologías.

c) METHONTOLOGY: Lleva a cabo su objetivo mediante las tareas de especificación, adquisición de conocimiento, conceptualización, integración, implementación, evaluación y documentación de las ontologías (Fernandez-Lopez et al., 1997). Esta es la metodología más consensuada para la creación de ontologías.

   Las etapas a través de las cuales se construye una ontología según esta metodología son:

   - Especificación: Donde se identifica el objetivo y el alcance de la ontología.
   - Conceptualización: Donde se describe, en un modelo conceptual, la ontología que debería ser construida para llegar a la especificación definida en el paso anterior.
   - Formalización: Donde se transforma la descripción hallada en el paso anterior a un modelo formal.
   - Implementación: Donde se implementa la ontología formalizada en un lenguaje de representación de conocimiento formal.
   - Mantenimiento: Donde se modifica y se corrige la ontología implementada.

2.2.6. Beneficios

El uso de ontologías tiene muchas ventajas y beneficios. Dentro de los beneficios más sobresalientes, según Yu (2011), se pueden mencionar los siguientes:

- Proveen una definición común y compartida sobre los conceptos claves dentro de un dominio.
- Ofrecen los términos que pueden ser usados cuando se crean documentos RDF en un dominio.
- Proveen una forma para la reutilización del conocimiento de un dominio.

- Permiten hacer explícitos los supuestos de un dominio.
- En conjunto con los lenguajes de descripción de ontologías como RDFS y OWL, proveen una vía para codificar conocimiento y semánticas de manera tal que pueda ser comprendido por las máquinas.
- Hacen posible el procesamiento automático de máquina a larga escala.

2.3. Web Semántica

2.3.1. Concepto

El término Web Semántica fue ideado por el director de la W3C, Tim Berners-Lee, y formalmente introducido al mundo en el artículo “The Semantic Web” (Berners-Lee et al., 2001). En este artículo se presenta la siguiente definición: “La Web Semántica es una extensión de la Web actual en la cual la información es proporcionada con un significado bien definido, permitiendo a las personas y a las computadoras trabajar en cooperación”.

De acuerdo con la W3C (2001), la Web Semántica provee un marco común que permite que los datos sean compartidos y reutilizados a través de los límites de aplicación, las empresas y la comunidad.

Otra definición para la Web Semántica es la presentada por Yu (2011), la cual expresa lo siguiente: “La Web Semántica es una colección de tecnologías y estándares que permiten a las computadoras entender el significado (semántica) de la información en la Web”.

De esta manera, según las definiciones anteriores, se puede concluir que la Web Semántica es una Web extendida, que posee mayor significado, permitiendo que los usuarios de Internet puedan encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. El hecho de que la información en la Web posea mayor significado o semántica, permite que se puedan obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización

de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de manera sencilla. La Web Semántica se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, comúnmente, el acceso a la información se convierte en una tarea difícil. La Web Semántica busca poder clasificar, dotar de estructura y anotar los recursos con semántica explícita procesable por máquinas, permitiendo que éstas tengan un nivel suficiente de comprensión de la web como para hacerse cargo de una parte (la más costosa, rutinaria, o físicamente inabarcable) del trabajo que actualmente realizan manualmente los usuarios que navegan e interactúan en Internet.

2.3.2. Tecnologías y Herramientas

Se presentan a continuación las tecnologías y herramientas que brindan soporte a la Web Semántica y hacen posible su desarrollo.

XML y XML Schema

XML (Bray et al., 2004), es un metalenguaje extensible de etiquetas desarrollado por la W3C. Permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XML se basa en Unicode por lo tanto, los documentos XML pueden usar caracteres de casi todas las lenguas que se hablan. Por otro lado, XML usa URIs para especificar de forma única los recursos de la Web Semántica, del mismo modo que la Web actual usa URLs.

XML aporta la sintaxis superficial para los documentos estructurados, pero sin dotarles de ninguna restricción sobre el significado.

XML Schema es un vocabulario para describir las propiedades y las clases de los recursos RDF, con una semántica para establecer jerarquías de generalización entre dichas propiedades y clases.

RDF y RDF Schema

RDF es un modelo estándar para el intercambio de datos en la Web. Es un modelo de datos para los recursos y las relaciones que se puedan establecer entre ellos. Aporta una semántica básica para este modelo de datos que puede representarse mediante XML.

Se trata de una infraestructura que permite la interoperabilidad de metadatos mediante el diseño de mecanismos que dan soporte a las convenciones comunes de semántica, sintaxis y estructura. RDF hace uso de XML como sintaxis común para el intercambio y proceso de metadatos. Provee además un medio para publicar vocabularios legibles por los humanos y capaces de ser procesados por las máquinas, y éstos pueden ser reutilizados, extendidos o refinados para adaptarlos a los diferentes dominios específicos según sus requerimientos.

RDF basa su modelo en tres partes:

- Recursos (sujeto) que son todo aquello de lo que se puede decir algo y son referenciados por un identificador único de recursos (URI).
- Propiedades (predicados) que definen atributos, aspectos, características o representan una relación que describe a un recurso.
- Declaraciones (objeto) los cuales nos sirven para dar valores a las propiedades de un recurso específico. El objeto de un estamento puede ser un recurso o un literal, por ejemplo, un recurso especificado por un URI, o bien un string u otras primitivas de datos definidas por XML.

Los tres constituyen una tripleta (sujeto, predicado, objeto) que es la construcción básica en RDF. Un conjunto de dichas tripletas es llamado grafo RDF, el cual define la sintaxis abstracta de RDF.

RDF se puede emplear en cualquier campo: no se asocia a ningún dominio en particular. Cada organización o persona puede definir su propio vocabulario mediante lo que se conoce como RDF Schema, que se define en función de las sentencias RDF.

RDF Schema permite la definición de jerarquías de clases (relación de subclase y subpropiedad) de objetos y propiedades (relaciones binarias) y permite la creación de restricciones (rango y dominio). Este esquema se aproxima más al concepto de ontología ya que se dispone de relaciones diseñadas para especificar la jerarquía de la taxonomía de conceptos que define un conocimiento. Funciona como un modelo semántico de datos capaz de permitir preguntas referentes a su contenido y no a la estructura del documento.

RDF Schema incluye tres clases principales:

- rdfs:Resource. Se consideran instancias de esta clase los recursos, que son todas las entidades descritas por expresiones RDF.
- rdfs:Class. Las clases RDF pueden representar páginas web, organizaciones, personas, búsquedas en la Web, etc. Toda clase es miembro de rdfs:Class.Cuando se crea una nueva clase mediante un esquema RDF, la propiedad rdf:type del recurso representado por la clase adquiere como valor el recurso rdfs:Class o alguna subclase suya. Cuando un recurso tiene la propiedad rdf:type cuyo valor es una clase determinada, se dice que el recurso es una instancia de esa clase.
- rdf:Property. Representa el subconjunto de recursos RDF que son propiedades. El dominio y rango de estos recursos se describen mediante las propiedades rdfs:domain y rdfs:range respectivamente; las jerarquías de propiedades se describen mediante rdfs:subPropertyOf. Tanto rdfs:range como rdfs:domain son instancias de rdf:Property. El rango se usa para establecer que los valores de una propiedad son instancias de una o más clases. Y el dominio se emplea para establecer que un recurso con una cierta propiedad es instancia de una o más clases.

Una propiedad relevante de RDF Schema es rdfs:subClassOf, que describe una clase como subclase de otra. Solamente las instancias de rdfs:Class pueden tener dicha propiedad.

Para los metadatos, RDF Schema define varias propiedades:

- rdfs:comment proporciona la descripción en lengua natural de un recurso.
- rdfs:label proporciona una versión legible para los humanos del nombre del recurso.
- rdfs:seeAlso especifica un recurso que proporciona información adicional sobre el recurso principal.
- rdfs:isDefinedBy indica cuál es el recurso donde se define el recurso principal. Es una subpropiedad de rdfs:seeAlso.

OWL

OWL es una extensión de RDFS y emplea el modelo de tripletas de RDF. Con OWL se pueden definir clases mediante restricciones a otras clases, o con operaciones booleanas sobre otras clases, hay nuevas relaciones entre clases como la inclusión, disyunción y la equivalencia, se pueden definir restricciones de cardinalidad en propiedades o dar propiedades sobre las relaciones (transitiva, simetría) así como permitir clases enumeradas (McGuinness y van Harmelen, 2004).

OWL tiene mucha más capacidad expresiva que RDFS. Además, OWL facilita la importación y exportación de clases e incluye propiedades como sameAs, equivalentClass, equivalentProperty, differentFrom, etc. Por ejemplo, equivalentClass permite establecer que dos clases de distintas ontologías sean equivalentes, esto significa, que cada instancia de una clase será también instancia de la otra clase, y viceversa. La capacidad de expresar que dos clases son iguales resulta muy útil cuando se integran o mezclan ontologías y permite la interoperabilidad.

Una ontología en OWL es una secuencia de axiomas, hechos y referencias a otras ontologías, que se consideran incluidas en la ontología. Las ontologías OWL son documentos web, y pueden ser referenciados a través de una URI.

Se decidió separar OWL en tres niveles, atendiendo a su expresividad semántica:

- OWL Lite: La versión más simple para los programadores principiantes. Permite la jerarquía de clasificación y las restricciones simples.
- OWL DL: Esta versión ya tiene todo el vocabulario OWL completo. Las limitaciones son que las clases no son instancias ni tipos y los tipos no son ni instancias ni clases. No permite restricciones de cardinalidad en propiedades transitivas. Posee gran expresividad sin perder las propiedades de completitud y decidibilidad.
- OWL Full: Esta versión también incluye todo el vocabulario de OWL, pero interpretado de forma más amplia que en OWL DL, con la libertad proporcionada por RDF, en este caso no hay limitaciones para explotar todo su potencial. No tiene garantías computacionales.

Este lenguaje también posee funcionalidades de razonamiento para las ontologías.

SPARQL

SPARQL es un lenguaje de consulta para datos RDF y provee un protocolo de acceso de datos para la Web Semántica (DuCharme, 2011), el cual describe el proceso involucrado entre un cliente SPARQL y un procesador de consultas SPARQL (SPARQL EndPoint). Es una tecnología clave en el desarrollo de la Web Semántica que se constituyó como recomendación oficial del W3C el 15 de Enero de 2008.

Ofrece a los desarrolladores y usuarios finales un camino para presentar y utilizar los resultados de búsquedas a través de una gran variedad de información como puede ser datos personales, redes sociales y metadatos sobre recursos digitales como música e imágenes. SPARQL también proporciona un camino de integración sobre recursos diferentes.

Al igual que sucede con SQL, es necesario distiguir entre el lenguaje de consulta y el motor para el almacenamiento y recuperación de los datos. Por este motivo, existen múltiples implementaciones de SPARQL, generalmente ligados a entornos de desarrollo y plataforma tecnológicas.

El lenguaje SPARQL posee cuatro variaciones diferentes de consulta para diferentes propósitos:

- SELECT: Devuelve el resultado en forma tabular sobre XML, mostrando las vinculaciones encontradas de las variables.
- CONSTRUCT: Genera y devuelve el grafo RDF siguiendo el patrón definido en la consulta.
- ASK: Devuelve un resultado booleano dependiente de si existe una solución a la consulta.
- DESCRIBE: Devuelve información sobre un recurso especificado. Depende de la configuración del procesador.

Algunos de los beneficios que brinda SPARQL, según Yu (2011), son los siguientes:

- Consultar grafos RDF para obtener información específica.
- Consultar servidores RDF remotos y obtener los resultados requeridos.
- Ejecutar consultas regulares automatizadas sobre conjuntos de datos RDF para generar reportes.
- Habilitar el desarrollo de aplicaciones a alto nivel; es decir, la aplicación puede trabajar con los resultados de consulta SPARQL, no directamente con las sentencias RDF.

Jena Framework

Jena es un framework open source de Java para la implementación de aplicaciones de Web Semántica. Provee de un ambiente de programación para RDF, RDFS, OWL y SPARQL e incluye un motor de inferencias basado en reglas (Hebeler et al., 2009) (Yu, 2011).

La primera versión de Jena tenía capacidades de razonamiento muy limitadas y principalmente daba soporte a RDF. En la segunda versión ya se incluyó una API de Java para trabajar con ontologías y soporte del lenguaje OWL.

El framework Jena incluye:

- Una API para la lectura, procesamiento y escritura de datos RDF en XML, N-triples y formatos Turtle
- Una API de ontología para el manejo de ontologías OWL y RDFS
- Un motor de inferencias basado en reglas para el razonamiento con fuentes de datos RDF y OWL.
- Almacenamiento en memoria y persistente.
- Motor de consultas SPARQL: ARQ

2.4. Reutilización de Software

2.4.1. Concepto

La reutilización de software se define como el proceso de crear sistemas de software a partir de software existente; es el uso de cualquier tipo de artefacto o parte del mismo, creado con anterioridad, en un nuevo proyecto (Krueger, 1992).

Se define también, como la aplicación de una variedad de tipos de conocimientos acerca de un sistema en otro sistema con la finalidad de reducir el esfuerzo de desarrollo y mantenimiento del sistema.

La reutilización de software ha sido tema de investigación por muchos años dentro de la comunidad de software debido a los grandes beneficios que aporta, principalmente en cuanto a la reducción de tiempo y costo de desarrollo, y al aumento de la calidad de los sistemas de software resultantes ya que se utilizan artefactos validados (Robinson y Woo, 2004).

Un artefacto de software es cualquier producto tangible producido en algún punto durante el ciclo de vida de desarrollo de software. En principio, diferentes artefactos producidos durante este ciclo de vida pueden ser reutilizados. Algunos ejemplos típicos de artefactos reutilizables incluyen: código fuente, especificaciones de análisis y diseño, planeaciones (manejo de proyectos), data de pruebas, documentación, entre otros.

2.4.2. Beneficios y Dificultades Asociadas

La reutilización de software brinda importantes beneficios a la industria del software; sin embargo, también presenta ciertos inconvenientes.

Algunos de los principales beneficios que aporta la reutilización de software son:

- Reducción de los tiempos de desarrollo.
- Reducción de los costes de desarrollo.
- Aumento de la calidad de los productos.
- Incremento de la productividad del software, mediante la mejora de los tiempos en los que se desarrollan los nuevos proyectos informáticos.
- Mejoras en las actividades de mantenimiento y soporte de aplicación.

Dentro de las dificultades asociadas a la reutilización de software se presentan las siguientes:

- Resistencia a la adopción de la reutilización de software por parte de los equipos de desarrollo.
- Escasa formación y soporte metodológico que promueven la reutilización de software.
- Carencia de herramientas informáticas que propicien la reutilización de artefactos de software.

2.4.3. Reutilización en el Análisis de Requisitos

En la mayoría de las ocasiones, la reutilización de software es asociada únicamente con la reutilización de código fuente; sin embargo, tal como se indica en su definición, se puede reutilizar cualquier tipo de artefacto de software (Kim y Stohr, 1998). Aunado a esto, la reutilización de código fuente resulta ser problemática debido

a que las plataformas y tecnologías de desarrollo se encuentran en constante cambio. Por consiguiente, resulta necesario y conveniente aplicar la reutilización de software a artefactos creados en niveles más altos del proceso de desarrollo de software como es el caso del análisis de requisitos.

El análisis de requisitos es la primera fase que se realiza en el desarrollo de un software. Durante esta etapa se adquieren, reúnen y especifican las características funcionales y no funcionales que deberá cumplir el futuro sistema o software a desarrollar (Pressman et al., 2006).

Uno de los artefactos más importantes incluidos dentro de esta fase son los diagramas de casos de uso ya que muestran la relación entre los actores y los casos de uso o funcionalidades de un sistema.

Reutilizar diagramas de casos de uso brinda entonces la oportunidad de contar con definiciones formales y validadas de los requisitos funcionales de un sistema de software creado con anterioridad y de esta manera poder utilizarlos las veces que sean necesarios en diferentes proyectos de software. Además, se aplicaría reutilización sobre un nivel de abstracción más alto, evadiendo las limitaciones en cuanto a los cambios tecnológicos y de plataformas.

Capítulo 3

Trabajos Relacionados

Este capítulo presenta cuatro publicaciones científicos en el área de computación, seleccionados dentro de un conjunto considerables de trabajos que fueron consultados y analizados, ya que aportan ideas y conocimientos muy valiosos para los objetivos de nuestra propuesta.

3.1. Representación de los Modelos UML de Actividad como Ontologías

En este trabajo, Khan et al. (2008) presentan una solución que permite representar la información de los modelos UML de actividad como una ontología expresada en OWL.

De acuerdo con el método propuesto, un diagrama de actividad es modelado a través de una herramienta de soporte a UML. La herramienta exporta el diagrama a formato XMI el cual contendrá todo el conocimiento de comportamiento del diagrama en particular. Este archivo XMI junto con las reglas de mapeo escritas en forma de XSLT sirven como entrada para la aplicación de conversión. La salida es una ontología OWL que expresa el conocimiento representado por un diagrama de actividades. La Figura 3.1 muestra una visión general de la solución propuesta en este trabajo.

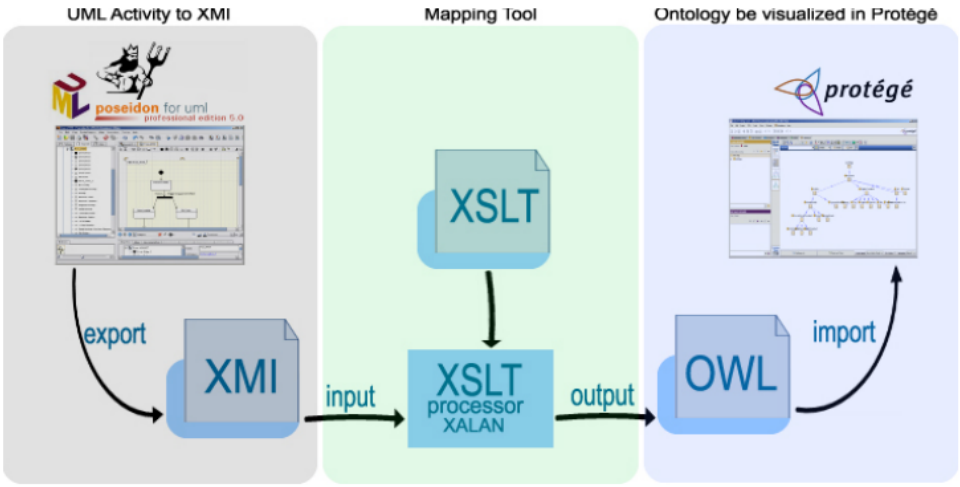


Figura 3.1: Visión general de la solución propuesta por Khan et al. (2008)

Se propone el empleo de una ontología base que permita capturar la información contenida en los diagramas de actividades. Utilizan la ontología OWL-S la cual soporta la representación de procesos genéricos y la modifican para obtener una ontología de procesos que se adapte a los propósitos de la investigación, dando como resultado la ontología UAD-OWL.

Con respecto a las reglas de mapeo proponen reglas de correspondencia entre los conceptos manejados en los diagramas de actividades y los conceptos de la ontología UAD-OWL. El cuadro 3.1 muestra estas correspondencias.

Type	**UML 2.0**	**UAD-OWL**
Construcción de Control	Fork	Fork
	Join	Join
	Transition	Choice
	Decision	Decision
	Merge	Merge
	State	Perform
Actividad	State	Action Task
	Model	Composite Activity

Tabla 3.1: Mapeo a nivel conceptual según la propuesta de Khan et al. (2008)

3.2. Recuperación Automática de Diagramas UML de Secuencia Reutilizables

En este trabajo propuesto por Robinson y Woo (2004), se proponen técnicas para la reutilización de cualquier artefacto UML, basando su atención en los diagramas de secuencia. Se introduce el proyecto REUSER el cual tiene como objetivo asistir a los analistas de software en la reutilización de artefactos de software.

REUSER está basado en un enfoque de reutilización dirigido por la comparación entre las relaciones estructurales de los diagramas las cuales se presentan en forma de grafos definidos por conceptos (conocidos como vértices) y relaciones (conocidos como aristas). La idea es realizar comparaciones globales en cuanto a la similitud entre grafos, por ejemplo, totalizando el número de diferencias entre vértices o aristas. De esta manera una herramienta calcula la cantidad de coincidencias basada en las estructuras de diseño representadas por grafos.

El enfoque se aplica de la siguiente manera:

- El analista define o selecciona una librería de artefactos
- Para consultar la librería, el analista brinda a la herramienta una estructura parcial del diagrama que desea encontrar.
- REUSER traslada el artefacto UML a su representación interna a través de la traducción de los elementos del metamodelo UML a un grafo directo. Los tipos en el metamodelo se traducen a tipos vértices y las asociaciones a aristas dentro del grafo.
- REUSER emplea el algoritmo SUBDUE para encontrar las coincidencias entre el grafo introducido y los grafos en la librería haciendo una evaluación de cada uno de los vértices y aristas. Para establecer la coincidencia entre grafos por lo menos la etiqueta de un nodo debe coincidir entre estos y dependiendo de la cantidad de coincidencias la puntuación de similitud se establece.
- REUSER devuelve las *n* mejores coincidencias según las puntuaciones obtenidas.

- El analista selecciona el artefacto devuelto que más coincide con su necesidad y lo adapta al contexto de su problema.

3.3. Reutilización de Especificaciones UML en un Dominio de Aplicación Restringido

En este trabajo Blok y Cybulski (1998) proponen un modelo para almacenar y recuperar componentes de diseño reutilizables para y desde una extensa colección de especificaciones UML. Describe un método para representar y comparar casos de uso en el modelo de dominio UML.

El método consiste en utilizar tanto un enfoque léxico (en el cual se infiere el significado de los objetos de un dominio por sus nombres) como un enfoque semántico (en el cual se infiere el significado de los objetos de un dominio a partir del propio modelo) para determinar la similitud entre artefactos. El enfoque semántico se utiliza cuando se conoce que los atributos de un artefacto se encuentran en el modelo de dominio y de esta manera se puede comparar la distancia semántica la cual es una medida que define la proximidad de los descriptores en la misma clase. El enfoque léxico se utiliza cuando los atributos del artefacto no son encontrados dentro del modelo de dominio y su afinidad solo puede ser deducida basado en las propiedades léxicas de sus nombres.

Para el propósito de comparar casos de uso se considera a los mismos como una colección de flujos de eventos. Por lo tanto, la similitud de casos de uso está basada completamente en la similitud de sus componentes en el flujo de eventos. La idea es agrupar los eventos en grupos descriptores de eventos dependiendo de la funcionalidad que defina el grupo. De esta manera cuando dos eventos se encuentran en el mismo grupo la distancia entre ellos es considerada cero. Para clasificar los eventos a través de grupos descriptores se utiliza un enfoque basado en las palabras que definen el evento; luego, para encontrar el grupo adecuado al que pertenece un evento en particular se toman en cuenta sinónimos y significados utilizando el lenguaje WordNet.

Una vez los eventos se encuentran clasificados dentro de grupos descriptores, un flujo de eventos completo para un caso de uso es representado como un vector de

descriptores. Estos vectores se encuentran almacenados y cada vez que un nuevo flujo de eventos es ingresado este se traduce en un vector y se realiza una comparación basada en lógica vectorial contra los vectores almacenados. De esta manera se pueden determinar los flujos de eventos similares al ingresado y por consiguiente, obtener los casos de uso asociados con lo cual se puede obtener la representación funcional completa de un sistema.

La Figura 3.2 muestra una visión general del método propuesto en este trabajo.

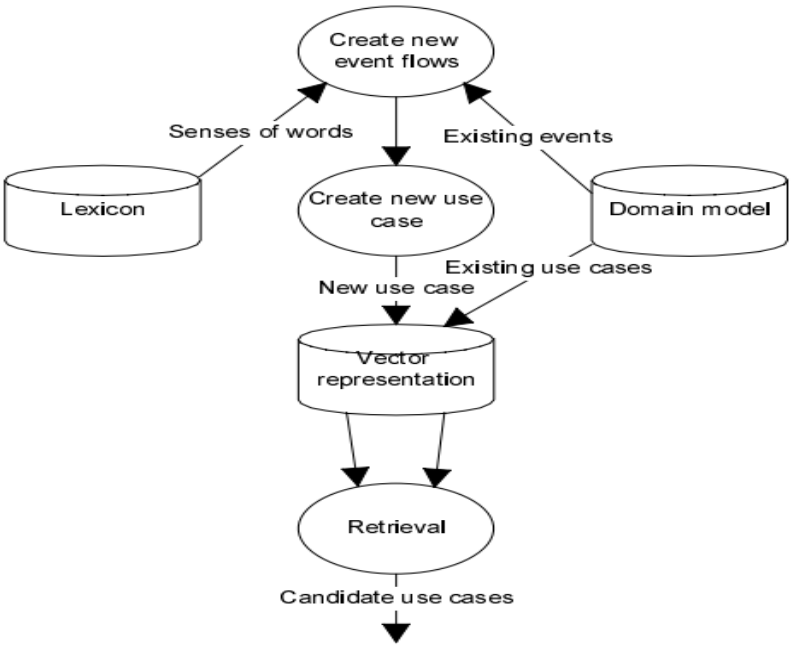


Figura 3.2: Visión general de la solución propuesta por Blok y Cybulski (1998)

3.4. KOntoR: Un Enfoque de Reutilización de Software basado en Ontologías

En este trabajo propuesto por Happel et al. (2006) se presenta KOntoR, una infraestructura para la reutilización de software empleando tecnologías de Web Semántica.

La arquitectura de KOntoR esta basada en dos elementos:

- Un componente respositorio para almacenar las descripciones en XML de los artefactos de software, independientemente de su formato original (WSDL, Corba IDL, UML).
- Un componente de conocimiento que abarca una infraestructura de ontología y razonamiento para aprovechar el conocimiento de fondo de los artefactos de software como tecnologías y/o lenguajes de programación, licencias de software, etc.

Para permitir el manejo de metadata arbitraria, el componente repositorio está basado en un meta-esquema particular propuesto que permite la inserción de información independientemente del formato en el cual se encuentre el artefacto. Cada conjunto de metadata es un documento XML que describe un artefacto de software a través de uno o más aspectos de información.

El segundo bloque de la arquitectura KOntoR, el componente de conocimiento, comprende una base de conocimiento la cual importa la metadata a una ontología; luego, el conocimiento estructural de los artefactos de software puede ser combinado con el conocimiento de fondo de los mismos ya que la ontología se puede extender a través del uso de otras ontologías de dominio como por ejemplo una ontología que defina las licencias de software estándar. El conocimiento de fondo contenido en la ontologías de dominios puede ser utilizado para asistir al usuario en un amplio rango de problemas de reutilización.

De esta manera se tiene una ontología que define tanto la esctructura propia de los artefactos de software como de la información de fondo relacionada.

Para la implementación del prototipo del componente de conocimiento, se utiliza una infraestructura basada en RDF/OWL. Se utiliza el API KAON2 para los procesos de almacenamiento y razonamiento de las ontologías. Las consultas semánticas sobre la base de conocimientos se apoyan en SPARQL.

El trabajo propone una interfaz en Java que permite buscar y recuperar artefactos de software que den solución a un problema basado principalmente en parámetros de información de fondo.

Capítulo 4

Trabajo Propuesto

En este capítulo será presentado el trabajo propuesto en esta investigación, el cual consiste en la implementación de una herramienta de software para la reutilización de diagramas de casos de uso en UML. De esta manera, se presentará, entre otras cosas, la visión general del sistema, el desarrollo de la ontología, la arquitectura del sistema y los detalles de su implementación.

4.1. Visión General

El trabajo propuesto consiste de una herramienta, bajo el lenguaje Java, que permite reutilizar diagramas de casos de uso en UML. La idea principal es manipular y almacenar la información relevante de los diagramas de casos de uso en una ontología en lenguaje OWL. Al contar con esta representación ontológica de la información propia de los diagramas de casos de uso dentro de un repositorio, la herramienta permite al usuario, a través de una interfaz gráfica, realizar consultas acerca de los diagramas que está interesado en obtener, a la vez que se aumenta la precisión en los resultados obtenidos gracias a las características taxonómicas, capacidad de inferencia y manejo de conceptos que poseen las ontologías OWL.

Para detallar mejor el concepto de este trabajo, se presenta la Figura 4.1 de la cual se pueden describir tres fases que definen el comportamiento de nuestra herramienta.

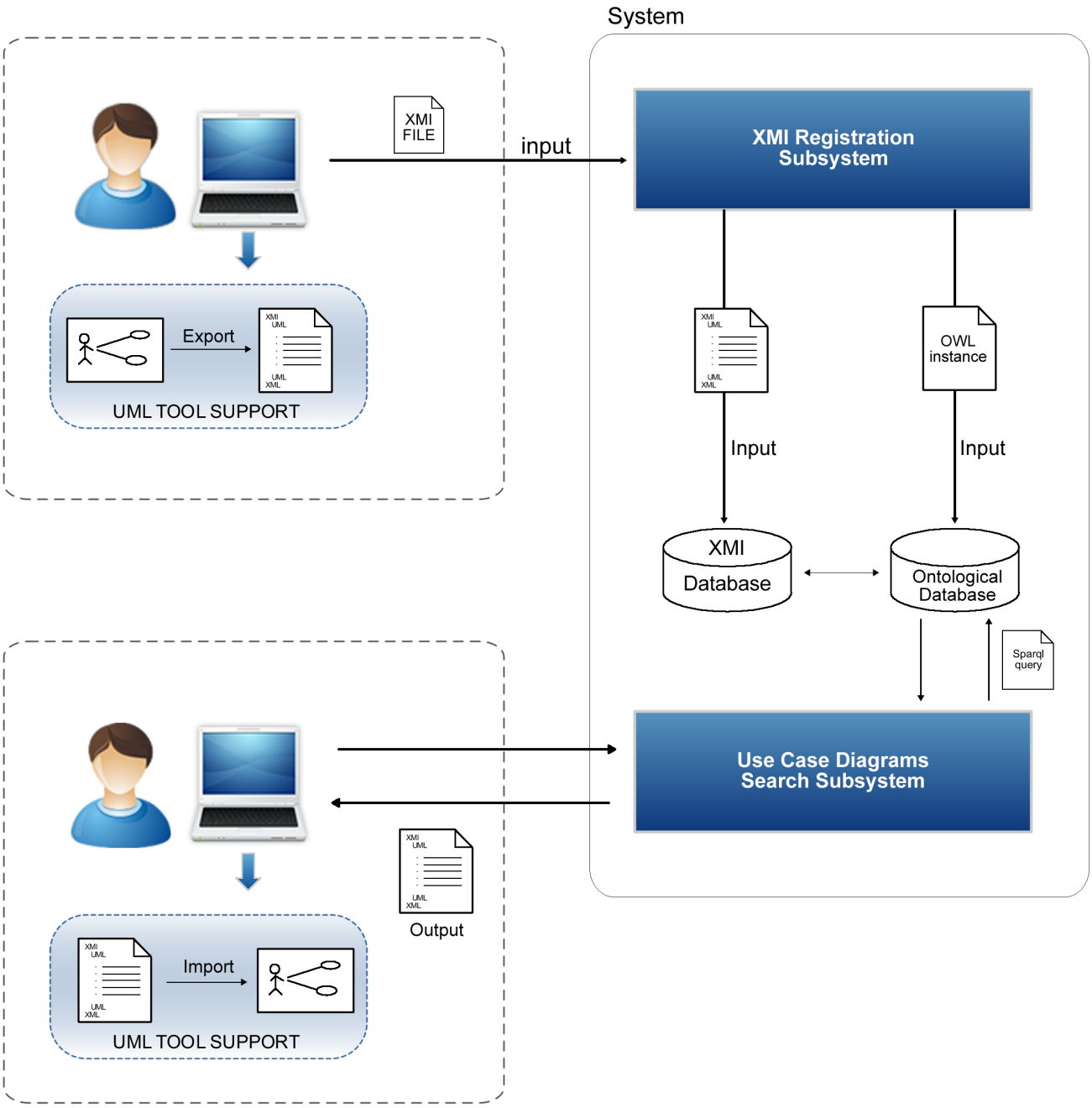


Figura 4.1: Vision General del Sistema Propuesto

Estas fases se definen como sigue:

a) Creación de diagramas de casos de uso en UML y exportación a formato XMI: Los diagramas de casos de uso son modelados utilizando una herramienta CA-

SE para UML como StarUML, Rational Rose, ArgoUML, entre otras. Una vez creados, se exportan a formato XMI a través de la opción de exportación que brindan la mayoría de estos programas.

b) Registro de los archivos XMI e información adicional de los diagramas de casos de uso en el sistema: A través de esta fase, el usuario procede a cargar el archivo XMI en el sistema e introducir información adicional correspondiente al diagrama. Una vez cargado el archivo e ingresada la información adicional, el sistema se encarga de extraer los datos necesarios para construir la instancia de la ontología OWL; esta instancia es almacenada de forma persistente en una base de datos ontológica. De forma paralela, el sistema guarda el archivo XMI en un directorio dentro del servidor dedicado al almacenamiento de estos archivos.

c) Búsqueda y recuperación de la información: A través de la herramienta, el usuario puede realizar búsquedas de diagramas de casos de uso que fueron almacenados con anterioridad. El usuario emplea una serie de parámetros para definir su consulta; la herramienta interpreta la solicitud del usuario como una consulta sobre la ontología OWL que se encuentra almacenada en la base de datos, se recuperan los individuos que concuerdan con la consulta y finalmente se presentan como resultados los archivos XMI asociados a estos individuos o instancias. El usuario puede proceder entonces a abrir este archivo XMI con cualquiera de las herramientas CASE para UML.

4.2. Tecnologías Utilizadas

En esta sección, se describen de manera breve las tecnologías y herramientas de software utilizadas durante el desarrollo de nuestra herramienta:

a) Unified Modelling Language (UML): UML es el lenguaje de modelado y representación de sistemas de software más empleado. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. La versión utilizada para este trabajo es la versión 1.3.

b) XML Metadata Interchange (XMI): XMI es un estándar del Object Management Group (OMG) para el intercambio de información a nivel de metadata a través

de XML. Es una especificación para el intercambio de diagramas. La versión utilizada para este trabajo es la versión 1.1.

c) Ontology Web Language (OWL): Es el lenguaje utilizado para crear la ontología que alojará la información de los diagramas de casos de uso. Se utiliza OWL 2, sublenguaje OWL Lite.

d) IDE Netbeans 6.9.1: NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo.

   Es empleado en nuestro proyecto para programar bajo lenguaje Java.

e) API JDOM 1.1.1: JDOM es un API para leer, crear y manipular documentos XML en Java.

f) Framework Jena y Lenguaje SPARQL: Para el desarrollo de la herramienta se utilizan el API de Jena en su versión 2.6.2 y ARQ versión 2.8.2.

g) MySQL 5.1.41: MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

   En nuestro trabajo, la base de datos solo se utilizará para almacenar la tripletas RDF de la ontología OWL de manera persistente, no así para establecer relaciones.

4.3. Ontología Propuesta

El sistema para la reutilización de diagramas de casos de uso que se propone en este trabajo utiliza una ontología que permite describir y almacenar los conceptos relacionados a los diagramas de casos de uso. Esta estructura de representación de conocimiento es la base para el desarrollo restante del sistema, ya que es sobre este modelo que se realizan las consultas y recuperación de información a través de la manipulación de los individuos de la ontología.

La idea de esta ontología no radica en representar en forma de mapeo los componentes o elementos de los diagramas de casos de uso como ontologías sino en

obtener a través de la ontología el significado de cada diagrama, qué problema define, quiénes participan, sobre qué área de conocimiento/ trabajo se define, entre otros aspectos de información que pueden obtenerse a través de estos diagramas.

La Figura 4.2 presenta un modelo conceptual definido para la ontología propuesta, donde se pueden apreciar las clases que hemos definido y relacionado a este dominio, siendo éstas: Diagrama, Actor, Caso de Uso, Proyecto y Categoría.

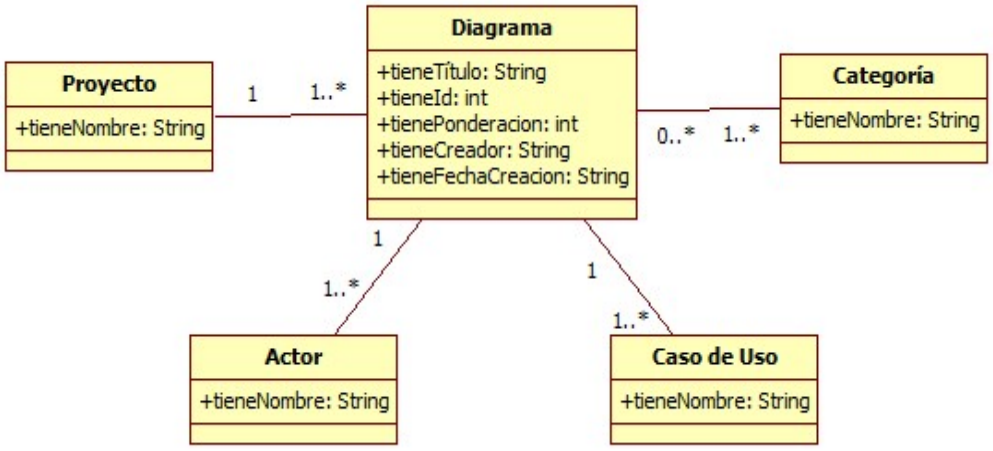


Figura 4.2: Modelo conceptual para la ontología propuesta

En cuanto a la metodología utilizada para la construcción de esta ontología se utilizó METHONTOLOGY. Esta elección se justifica en la simplicidad y facilidad de la metodología.

4.3.1. Dominio y Alcance

La ontología propuesta en este trabajo, representa los conceptos más significativos en la definición de los diagramas de casos de uso. Cabe destacar que esta ontología establece un conjunto de conceptos mínimos y necesarios para representar el conocimiento del dominio, no siendo considerados algunos conceptos correlacionados o complementarios como conceptos de diagramas asociados a los diagramas de casos de uso según UML.

Para el pleno entendimiento de los objetivos de este trabajo, se hace necesario que las consultas sobre la ontología respondan a cuestiones como:

- ¿Cuáles diagramas de casos de uso tienen ciertos actores dentro de su definición?
- ¿En cuáles diagramas de casos de uso se pueden encontrar definidos determinados casos de uso?
- ¿Cuáles diagramas de casos de uso pertenecen a una categoría determinada?
- ¿Para un proyecto de software determinado, cuáles diagramas de casos de uso se encuentran asociados?
- ¿Cuáles diagramas de casos de uso son los más reutilizados?

4.3.2. Clases

El modelo conceptual de la ontología para la manipulación de los diagramas de casos de uso fue establecido a partir de un levantamiento de aspectos relacionados a los diagramas de casos de uso. Posterior a esta recolección, estos aspectos fueron consolidados en un número mínimo y necesario de clases para representar el dominio en cuestión.

Para una mejor comprensión de las clases definidas para la ontología, la Tabla 4.1 presenta dichas clases y su descripción.

Clase	Descripción
Diagrama	Representa la clase principal de la ontología. Define los diagramas de caso de uso
Proyecto	Representa los proyectos de software que se desarrollan
Categoría	Representa las diferentes áreas de conocimiento o trabajo sobre las cuales se puede desarrollar un software
Actor	Representa los actores que interactúan con el sistema dentro de los diagramas de casos de uso
CasoUso	Representa los casos de uso que definen las funcionalidades del sistema de software

Tabla 4.1: Listado de Clases para la Ontología Propuesta

Cabe destacar que la clase Categoría poseerá subclases que representan las áreas de conocimiento dentro de las cuales puede surgir la necesidad de crear un sistema de software en un tiempo determinado. Se presentan como subclases debido a que existen diferentes individuos que comparten características comunes y pueden ser agrupados en alguna de estas subclases. No hemos agregado estas subclases al modelo conceptual de nuestra ontología debido a que son clases que se irán agregando a través de la herramienta. Más adelante en este capítulo se explicará a detalle como se va enriqueciendo la ontología al permitir al usuario de la herramienta agregar nuevas subclases a la clase Categoría.

4.3.3. Propiedades

Luego de la definición del modelo conceptual de la ontología, se establecieron las propiedades para cada concepto o clase. Estas permiten establecer las relaciones en la ontología. Como primer paso de esta etapa fueron enumeradas las propiedades necesarias para completar la representación de los diagramas de casos de uso en la ontología. Las propiedades son de tipo objeto, las cuales asocian diferentes individuos de la ontología, y de tipo dato, las cuales complementan la descripción de un determinado concepto.

La Tabla 4.2 presenta las propiedades de la ontología para la manipulación de diagramas de casos de uso. Para cada propiedad se establece el dominio, el rango o

tipo y una breve descripción.

Nombre	Dominio	Rango/Tipo	Descripción
tieneActor	Diagrama	Actor	Actor que pertenece al diagrama
tieneCasoUso	Diagrama	CasoUso	Caso de uso que pertenece al diagrama
tieneCategoria	Diagrama	Categoría	Categoría dada al diagrama de casos de uso
tieneProyecto	Diagrama	Proyecto	Proyecto de software al que pertenece el diagrama de casos de uso
tieneCreador	Diagrama	String	Creador del diagrama de casos de uso
tieneId	Diagrama	int	Identificador único para el diagrama como individuo dentro de la ontología
tienePonderacion	Diagrama	int	Ponderación dada al diagrama dependiendo de la cantidad de veces que haya sido reutilizado
tieneTitulo	Diagrama	String	Título del diagrama de casos de uso
tieneFechaCreacion	Diagrama	String	Fecha en la cual fue creado el diagrama

Tabla 4.2: Listado de Propiedades para la Ontología Propuesta

Por otro lado, se emplean propiedades incorporadas en OWL para establecer relaciones entre una propiedad definida y una clase, o entre los propios individuos dentro de la ontología. Al realizar estas relaciones se plantean axiomas que permitirán posteriormente llevar a cabo inferencias. Las propiedades incorporadas más utilizadas en la definición de la ontología propuesta son:

- rfds:domain
- rdfs:range
- owl:sameAs

4.3.4. Implementación de la Ontología Propuesta

La ontología propuesta en este trabajo fue implementada siguiendo el modelo de especificación OWL accesible a través de los recursos disponibles en Jena para la manipulación de ontologías.

Jena gestiona una ontología a través de modelos para ontologías los cuales son una extensión de modelo RDF de Jena con capacidades extras para la manipulación de ontologías. Estos modelos pertenecen a la clase OntModel y son creados a través de la clase ModelFactory la cual provee métodos para la creación de tipos estándar de modelos.

La Figura 4.3 muestra un componente *Ontology* creado para encapsular las actividades de gestión de la ontología propuesta. Este componente posee una estructura *Model* que expone una interface para la gestión tanto del modelo como de los individuos de la ontología.

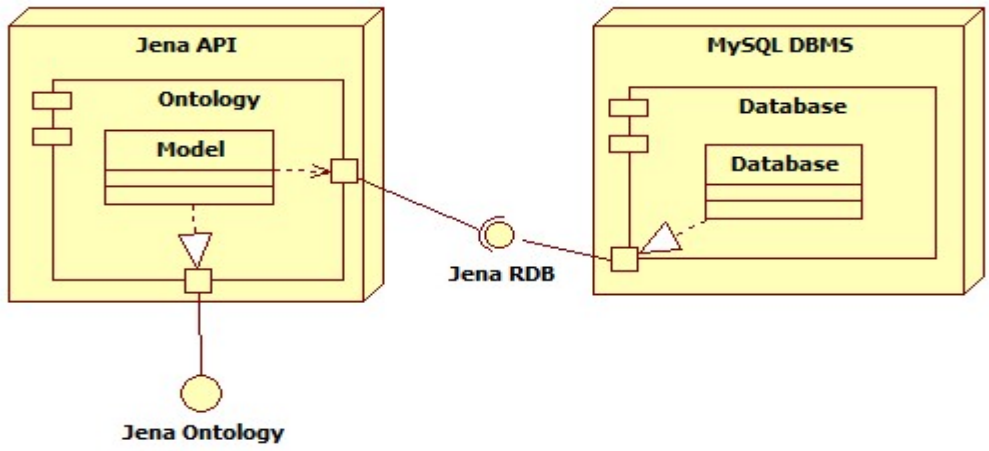


Figura 4.3: Componente Jena para la manipulación de la ontología

Conociendo las clases y métodos que Jena proporciona para la creación de una ontología, se puede proceder entonces a realizar la implementación necesaria para crear la ontología. La Figura 4.4 muestra parte del código Java del método *CrearOntologia* la cual como su nombre indica es utilizada para crear la ontología base que

utilizará nuestro sistema de reutilización para guardar y estructurar la información de los diagramas de casos de uso.

```
public static void CrearOntologia(){
        OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF);
        String NS = "diagramas";
        model.setNsPrefix(NS, "http://www.owl-ontologies.com/ontologia baseb.owl");
        OntClass Diagrama = model.createClass(NS + ":" + "Diagrama");
        OntClass CasoUso = model.createClass(NS + ":" + "CasoUso");
        OntClass Actor = model.createClass(NS + ":" + "Actor");
        OntClass Categoria = model.createClass(NS + ":" + "Categoria");
        OntClass Proyecto = model.createClass(NS + ":" + "Proyecto");

        //Creamos la propiedad tieneCasoUso
        ObjectProperty tieneCasoUso = model.createObjectProperty(NS + ":" + "tieneCasoUso");
        tieneCasoUso.addDomain(Diagrama);//Clase a la que pertenece
        tieneCasoUso.addRange(CasoUso);//Tipo de la propiedad


        DatatypeProperty tieneCreador = model.createDatatypeProperty(NS + ":" + "tieneCreador");
        tieneCreador.addDomain(Diagrama);//Clase a la que pertenece
        tieneCreador.addRange(XSD.xstring);//Tipo de la propiedad
        tieneCreador.convertToFunctionalProperty();//Para que solo acepte un valor.
                .
                .
                .
}
```

Figura 4.4: Código para la creación de la ontología base

Una clase simple es representada en Jena a través de un objeto *OntClass*. Podemos apreciar en el fragmento de código que se utiliza *OntClass* para definir las clases propuestas para nuestra ontología: Diagrama, Actor, Categoria, CasoUso y Proyecto. De igual manera sucede para las propiedades con *ObjectProperty* que nos permitirá definir las propiedades: tieneActor, tieneCasoUso, tieneProyecto y tieneCategoria, y *DatatypeProperty*, las propiedades: tieneId, tienePonderacion, tieneCreador, tieneTitulo, tieneFechaCreacion.

La ontología creada es almacenada en un archivo llamado ontologia_base.owl. La ontología creada se presenta en el Anexo 1 en formato RDF/XML.

Como se pudo notar también, la Figura 4.3 presenta un componente *Database* como parte de un nodo que representa el sistema de gestión de base de datos (SGBD)

MySQL. Esto es debido a que se decidió almacenar la ontología de manera persistente ya que esta acción garantiza un mejor desempeño en las consultas SPARQL que serán realizadas sobre la ontología, además de que se provee mayor seguridad en cuanto a pérdida de información.

Para la persistencia de la ontología, también se utilizan los recursos de Jena, lo cual posibilita el almacenamiento de la ontología en un banco de datos.

Para los propósitos de almacenar una ontología de manera persistente, Jena emplea el modelo "database-backed". Este modelo se identifica a través de la clase llamada *ModelRDB*; el método *createModelRDBMaker()* llamado en la clase *ModelFactory* responde a un objeto *ModelMaker* que entiende como manejar modelos RDF y posteriormente transformarlos en grafos persistentes en una base de datos. Como parámetro se le envía la conexión para que el objeto *ModelMaker* pueda operar sobre la base de datos especificada.

La Figura 4.5 muestra el fragmento de código correspondiente al método *GuardarModelo* el cual se encarga de almacenar de manera persistente la ontología en la base de datos en MySQL.

```
public static boolean GuardarModelo(String dbmodelname, OntModel ontmodel) {
    boolean guardado = false;
    IDBConnection con = null;
    con = new DBConnection(DB_URL, DB_USER, DB_PASSWD, DB);
    ModelMaker maker = ModelFactory.createModelRDBMaker(con);

    ModelRDB RDBModel = (ModelRDB) maker.openModel(dbmodelname);
    OntModel ontmodel2 = null;
    ontmodel2 = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF, RDBModel);
    ontmodel2.add(ontmodel);
    ontmodel2.commit();
    guardado = true;
    try {
        ontmodel2.close();
        ontmodel.close();
    } catch (Exception exc) {
        return guardado;
    }
```

Figura 4.5: Código para el almacenamiento persistente de la ontología en MySQL

Al almacenar la ontología en MySQL, Jena crea automáticamente siete tablas, siendo las más relevantes:

- jena_g1t1_stmt : Contiene las sentencias/ tripletas contenidas en la ontología.
- jena_graph : Contiene los modelos en la base de datos.

Con todo esto se ha logrado la creación de la ontología base y su almacenamiento persistente en una base de datos MySQL.

4.3.5. Consultas SPARQL asociadas al Sistema

A continuación se presentan una serie de consultas SPARQL que responden tanto a las preguntas establecidas en la sección 4.3.1 como a otras de interés y que serán la base para la recuperación de los diagramas de casos de uso. Los resultados que devuelvan estas consultas dependerán de la información que esté almacenada dentro de la ontología propuesta.

Estas consultas serán empleadas como parte de la implementación de la herramienta de apoyo, y se presentan en esta sección puesto que estamos abordando el tema ontológico.

La Figura 4.6 muestra la consulta SPARQL que permite recuperar todas las categorías almacenadas en la ontología.

```
PREFIX vcard:<diagramas:>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?x
WHERE {?x rdfs:subClassOf vcard:Categoria};
```

Figura 4.6: Consulta que retorna las categorías almacenadas en la ontología

Por su parte la Figura 4.7 muestra la consulta SPARQL que permite recuperar las subcategorías por categoría, que no son más que individuos de las subclases derivadas de la clase Categoría.

```
PREFIX vcard:<diagramas:>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?x
WHERE {?x rdf:type vcard:CategoriaEspecifica}
```

Figura 4.7: Consulta que retorna las subcategorías de una categoría

La palabra en rojo indica que este nombre varía dependiendo de la categoría que vayamos a consultar.

La Figura 4.8 presenta la consulta SPARQL que retorna los proyectos almacenados en la ontología.

```
PREFIX vcard:<diagramas:>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?x
WHERE {?x rdf:type vcard:Proyecto};
```

Figura 4.8: Consulta que retorna los proyectos almacenados en la ontología

Finalmente, la Figura 4.9 muestra la consulta SPARQL que permite recuperar los diagramas de casos de uso que coinciden con los parámetros de búsqueda establecidos, facilitando información como id, título, comentarios y ponderación de los diagramas.

```
PREFIX vcard:<diagramas:>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?x ?t ?co ?id ?pon
WHERE {
?x vcard:tieneId ?id .
?x rdfs:comment ?co .
?x vcard:tienePonderacion ?pon .
?x vcard:tieneActor ?y . FILTER (regex(str(?y),"actor","i")) .
?x vcard:tieneCasoUso ?z . FILTER (regex(str(?z),"casouso","i")).
?x vcard:tieneCategoria ?c . FILTER (?c = vcard:categoria).
?x vcard:tieneTitulo ?t . FILTER (regex(str(?t), "titulo","i")).
?x vcard:tieneProyecto ?p . FILTER (?p = vcard:proyecto).
?x vcard:tieneCreador ?cr . FILTER (regex(str(?cr), "creador","i")).
?x vcard:tienefechaCreacion ?f . FILTER (regex(str(?f), "fecha","i"))
}
ORDER BY DESC (?pon) DESC(?t);
```

Figura 4.9: Consulta que retorna los diagramas de casos de uso según la búsqueda realizada

Una variación de esta consulta ocurre cuando no se establecen parámetros de búsqueda y por lo tanto se hace una consulta de todos los individuos que definen los diagramas de casos de uso en la ontología. La Figura 4.10 muestra la consulta SPARQL correspondiente.

```
PREFIX vcard:<diagramas:>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?x ?t ?co ?id ?pon
WHERE {?x vcard:tieneId ?id . ?x rdfs:comment ?co .
?x vcard:tienePonderacion ?pon .?x vcard:tieneActor ?y .
?x vcard:tieneCasoUso ?z . ?x vcard:tieneCategoria ?c .
?x vcard:tieneTitulo ?t . ?x vcard:tieneProyecto ?p .
?x vcard:tieneCreador ?cr . ?x vcard:tienefechaCreacion ?f }
ORDER BY DESC(?pon) DESC(?t)
```

Figura 4.10: Consulta que retorna todos diagramas de casos de uso almacenados en la ontología

4.4. Especificaciones Funcionales del Sistema

En la sección 4.1 se describieron de manera general las funcionalidades que definen el comportamiento del sistema para la reutilización de diagramas de casos de uso. En esta sección vamos formalizar esa definición.

La Figura 4.11 muestra un diagrama de casos de uso que presenta las funcionalidades del sistema.

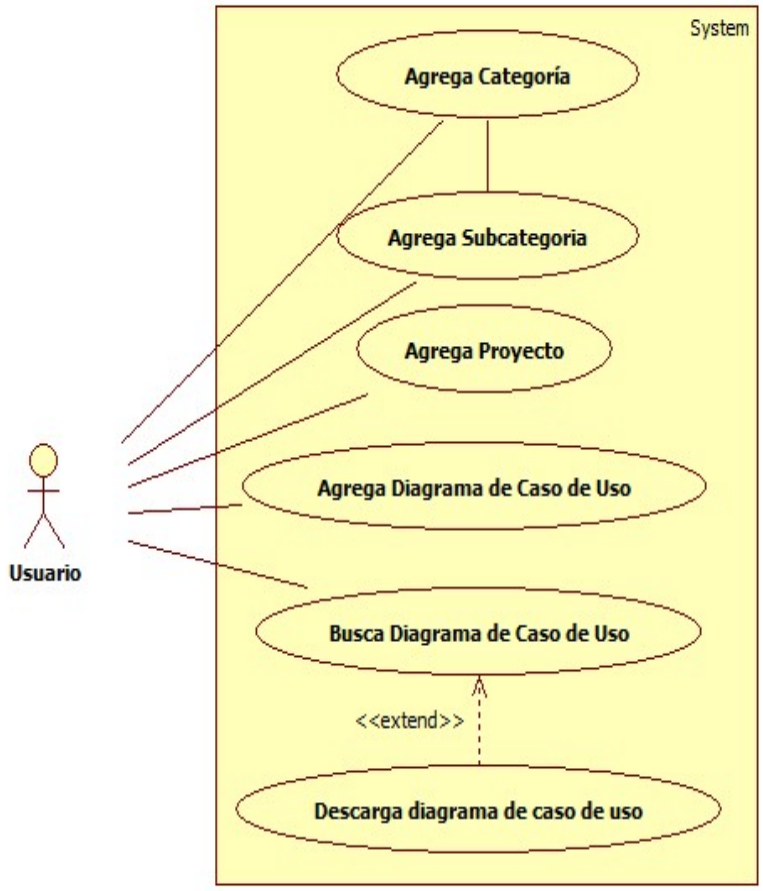


Figura 4.11: Diagrama de Casos de Uso para el Sistema Propuesto

A continuación se presentan las especificaciones para los casos de uso establecidos en la Figura 4.11.

Nombre	Agrega Proyecto
Actores	Usuario
Descripción	Permite agregar nombres de proyectos de software
Precondiciones	El actor ha accedido al sistema
Flujo de Eventos	▪ El actor selecciona la opción Agregar Proyecto ▪ El actor ingresa el nombre del proyecto ▪ El actor presiona la opción Guardar ▪ El sistema almacena el nuevo proyecto en la base de datos

Tabla 4.3: Especificación de Caso de Uso: Agrega Proyecto

Nombre	Agrega Categoría
Actores	Usuario
Descripción	Permite agregar categorías para clasificar los diagramas de casos de uso
Precondiciones	El actor ha accedido al sistema
Flujo de Eventos	1. El actor selecciona la opción Agregar Categoría 2. El actor ingresa el nombre de la categoría 3. El actor presiona la opción Guardar 4. El sistema almacena la nueva categoría en la base de datos

Tabla 4.4: Especificación de Caso de Uso: Agrega Categoría

Nombre	Agrega Subcategoría
Actores	Usuario
Descripción	Permite agregar subcategorías para clasificar los diagramas de casos de uso
Precondiciones	1. El actor ha accedido al sistema 2. Se ha creado por lo menos una categoría para la subcategoría a agregar
Flujo de Eventos	1. El actor selecciona la opción Agregar Subcategoría 2. El actor ingresa el nombre de la subcategoría 3. El actor selecciona las categorías a las que pertenece la subcategoría 4. El actor establece relaciones entre subcategorías 5. El actor presiona la opción Guardar 6. El sistema almacena la nueva subcategoría en la base de datos

Tabla 4.5: Especificación de Caso de Uso: Agrega Subcategoría

Nombre	Agrega Diagrama de Caso de Uso
Actores	Usuario
Descripción	Permite agregar la información de los diagramas de casos de uso al sistema
Precondiciones	1. El actor ha accedido al sistema 2. Se ha creado por lo menos una categoría para el diagrama de caso de uso 3. Se ha creado por lo menos una subcategoría para el diagrama de caso de uso 4. Se ha creado por lo menos un proyecto asociado al diagrama de caso de uso 5. Se ha creado por lo menos una categoría para el diagrama de caso de uso 6. Contar con el diagrama de casos de uso en formato XMI estándar.
Flujo de Eventos	1. El actor selecciona la opción Cargar Archivo XMI 2. El actor selecciona el archivo XMI a cargar 3. El actor ingresa la información adicional para los diagramas de casos de uso 4. El actor presiona la opción Guardar 5. El sistema guarda la información completa del diagrama de caso de uso en la ontología almacenada en la base de datos

Tabla 4.6: Especificación de Caso de Uso: Agrega Diagrama de Caso de Uso

Nombre	Busca Diagrama de Caso de Uso
Actores	Usuario
Descripción	Permite realizar búsquedas de diagramas de casos de uso
Precondiciones	1. El actor ha accedido al sistema 2. Se ha insertado por lo menos la información de un diagrama de casos de uso al sistema
Flujo de Eventos	1. El actor selecciona la opción Buscar Diagrama de Caso de Uso 2. El actor ingresa los parámetros de su búsqueda 3. El actor presiona la opción Buscar 4. El sistema presenta los resultados al actor

Tabla 4.7: Especificación de Caso de Uso: Busca Diagrama de Caso de Uso

Nombre	Descarga diagrama de caso de uso
Actores	Usuario
Descripción	Permite descargar los archivos XMI tras la búsqueda de los diagramas de casos de uso
Precondiciones	1. El actor ha accedido al sistema 2. Se ha realizado un proceso de búsqueda previo 3. Los resultados se han presentado de forma correcta
Flujo de Eventos	1. El actor presiona la opción Descargar del resultado de su elección 2. El sistema solicita ubicación donde guardar el archivo XMI 3. El actor selecciona la ubicación donde desea guardar el archivo XMI 4. El actor presiona la opción Guardar

Tabla 4.8: Especificación de Caso de Uso: Descarga diagrama de caso de uso

Con estas especificaciones de caso de uso quedan claras las funcionalidades que contempla el sistema a implementar.

4.5. Arquitectura del Sistema

Tomando en consideración las especificaciones de casos de uso establecidas en la sección anterior, podemos definir la arquitectura del sistema a través de un diagrama de componentes tal como se muestra en la Figura 4.12.

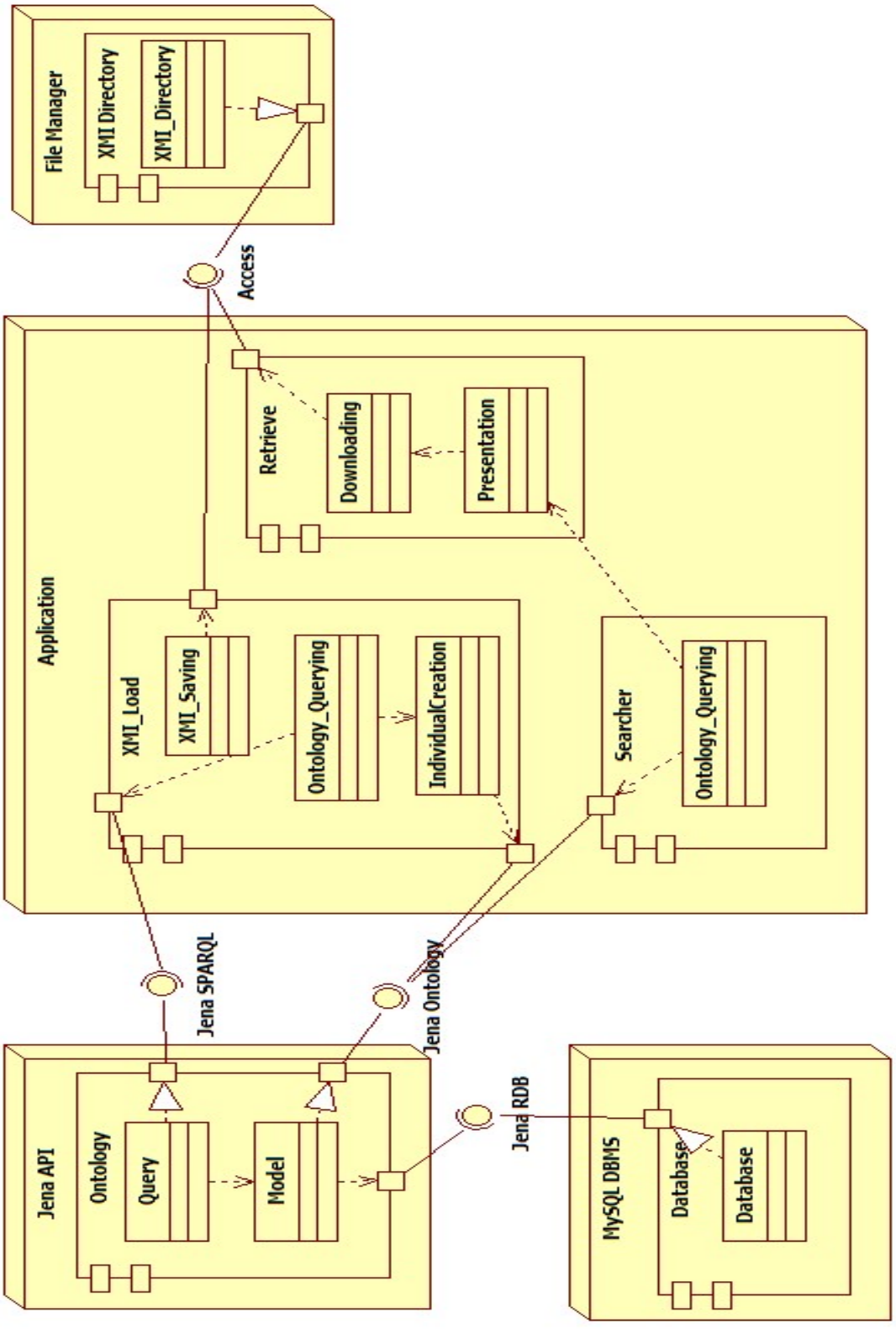


Figura 4.12: Arquitectura del sistema propuesto

Como se define en la arquitectura, el sistema trabaja con cuatro nodos principales: Jena API, MySQL DBMS, Application y File Manager. Cada uno posee un conjunto de componentes y clases que definen la implementación del sistema propuesto. A continuación vamos a definir cada uno de los componentes y sus clases:

a) Componente ***Ontology***: Tal como se planteó en la sección 4.3.4, este componente es creado para encapsular las actividades de gestión de la ontología propuesta. Posee la clase o estructura *Model* que permite gestionar tanto del modelo como los individuos de la ontología a través de la interface *Jena Ontology*, así como una clase Query la cual permite realizar las consultas SPARQL requeridas y manipular sus resultados a través de la interface *Jena Ontology*.

b) Componente ***Database***: Posee una clase llamada *Database* que representa la estructura de la base de datos donde se almacenarán los individuos de la ontología. Provee una interface *Jena RDB* a través de la cual se permite que la base de datos MySQL manipule la ontología como un modelo RDB y pueda almacenar la información en forma de tripletas RDF a través de las siete tablas predefinidas por Jena.

c) Componente ***XMI Directory***: Representa el directorio de archivos XMI. Posee una clase *XMI_Directory* que representa la estructura de almacenamiento interno dentro del repositorio y una interface *Acces* que permite el acceso al mismo.

d) Componente ***XMI_Load***: Este componente se encarga de la gestión correspondiente al registro de los archivos XMI asociados a los diagramas de casos de uso. Posee tres clases principales. La clase Ontology_Querying se encarga de cargar los individuos correspondientes a Proyectos y Categorías que permitirán agregar información adicional a los diagramas de casos de uso; utiliza la interface *Jena SPARQL* para realizar las consultas necesarias. La clase XMI_Saving se encarga de almacenar el archivo XMI en el repositorio central de tal manera que esté disponible para su recuperación posterior; utiliza la interface *Access* para acceder al repositorio. La clase IndividualCreation se encarga de crear los individuos de la ontología a partir de la información proveniente del archivo XMI y la información adicional ingresada a través de la interfaz de usuario; utiliza la interface *Jena Ontology* para la creación de los modelos y el acceso posterior a la base de datos para ejecutar las inserciones correspondientes.

e) Componente ***Searcher***: Se encarga, a través de la clase *Ontology_Querying*, de manipular los parámetros introducidos por el usuario para armar las consultas SPARQL. Utiliza la interface *Jena Ontology* para acceder a los individuos de la ontología almacenados en la base de datos. Almacena en memoria los resultados obtenidos tras ejecutar la consulta.

f) Componente ***Retrieve***: En general se encarga de presentar la información al usuario. La clase *Presentation* organiza los resultados procesados en el componente ***Searcher*** para su manipulación por el usuario y la clase *Downloading* permite descargar el archivo XMI asociado a los individuos recuperados tras ejecutar la consulta a través del acceso al componente ***XMI Directory***

4.6. Prototipo de Interfaces del Sistema

Considerando la implementación de la ontología base y de las consultas SPARQL necesarias para responder a las preguntas de dominio y alcance, además de las especificación de las funcionalidades del sistema, se hace necesario entonces especificar las interfaces de la aplicación.

Nuestra aplicación consta de dos interfaces gráficas principales creadas con ayuda de la biblioteca gráfica Swing: una para el registro de la información de los diagramas de casos de uso y otra para la búsqueda y recuperación de la información.

La Figura 4.13 muestra la interfaz gráfica por medio de la cual el usuario carga un archivo XMI a través de la herramienta e ingresa información adicional y relevante asociada al diagrama de caso de uso.

Como se aprecia la interfaz permite agregar información adicional como:

- Título: Permite introducir el nombre del diagrama de caso de uso
- Creador: Indica que analista de software diseñó el diagrama de caso de uso
- Fecha de creación: Indica la fecha de creación del diagrama
- Categorías: Permite establecer las categorías a las cuales pertenece el diagrama

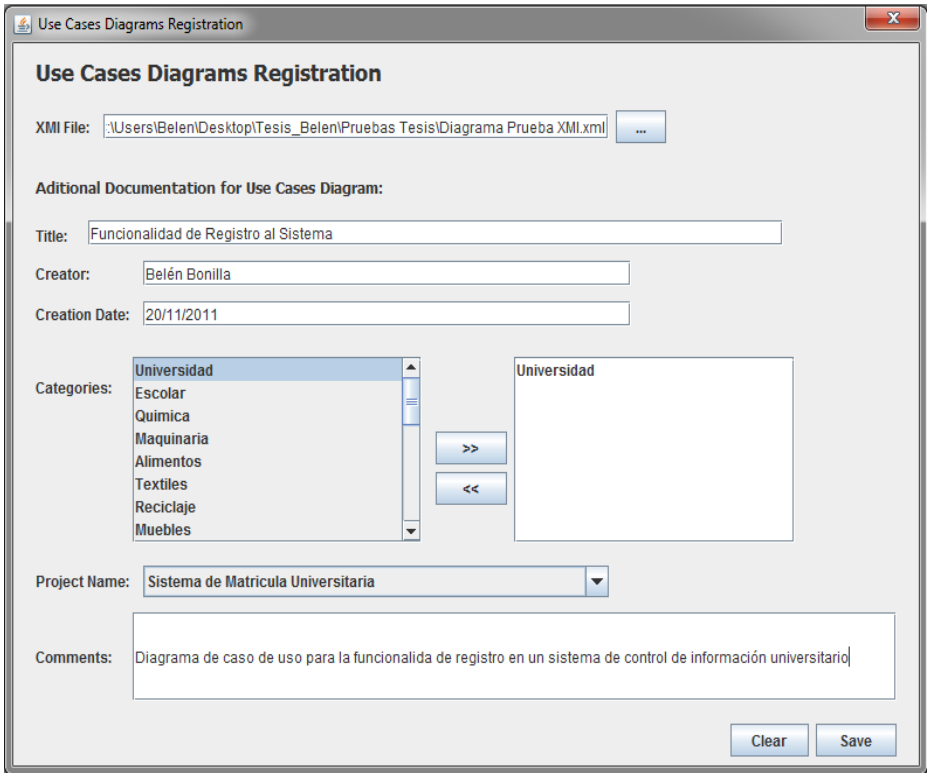


Figura 4.13: Interfaz Gráfica para el Registro de Diagramas de Casos de Uso

- Nombre del Proyecto: Indica el proyecto de software al que pertenece el diagrama
- Comentarios: Permite introducir algunos comentarios extras para definir el diagrama

Por su parte, la Figura 4.14 muestra la interfaz gráfica que permite al usuario realizar las búsquedas de los diagramas de casos de uso de su interés.

Esta interfaz contiene los siguientes parámetros/campos que permiten definir la búsqueda del usuario:

- Actor: Permite introducir uno o varios nombres de actores
- Caso Uso : Permite introducir uno o varios nombres de casos de uso
- Categorías: Permite filtrar por categorías predefinidas

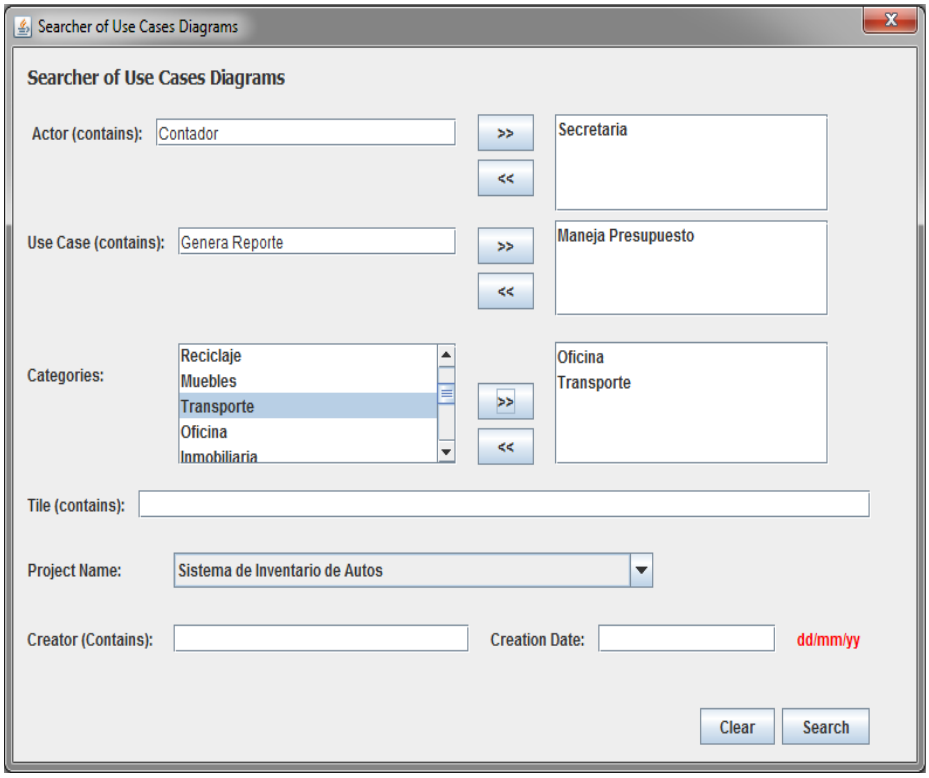


Figura 4.14: Interfaz Gráfica para la Búsqueda de Diagramas de Casos de Uso

- Título: Permite introducir una cadena que indica el título del diagrama de casos de uso
- Nombre del Proyecto: Permite escoger el nombre del proyecto de software
- Creador: Permite ingresar y filtrar por el nombre de un analista creador de diagramas de casos de uso
- Fecha de Creación: Permite filtrar por la fecha de creación de los diagramas de casos de uso

La Figura 4.15 muestra la interfaz de usuario que presenta los resultados tras ejecutar una búsqueda.

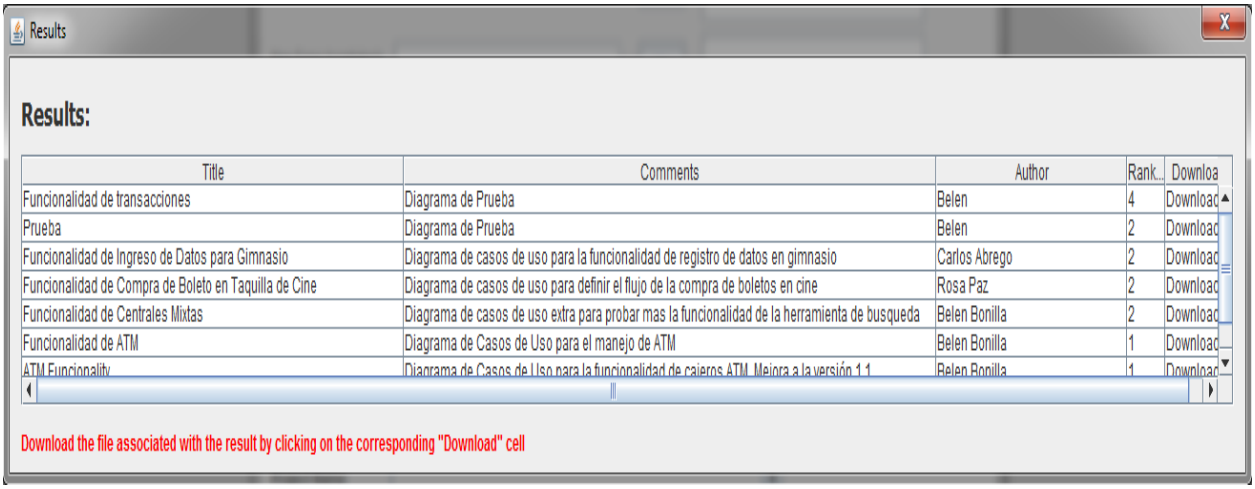


Figura 4.15: Interfaz Gráfica para la presentación de resultados

Otras interfaces son complementarias y permiten agregar individuos auxiliares a la ontología como son: categorías, subcategorías y proyectos.

Las Figuras 4.16, 4.17 y 4.18 muestran las interfaces para agregar categorías, subcategorías y proyectos respectivamente.

Figura 4.16: Interfaz Gráfica para agregar categorías

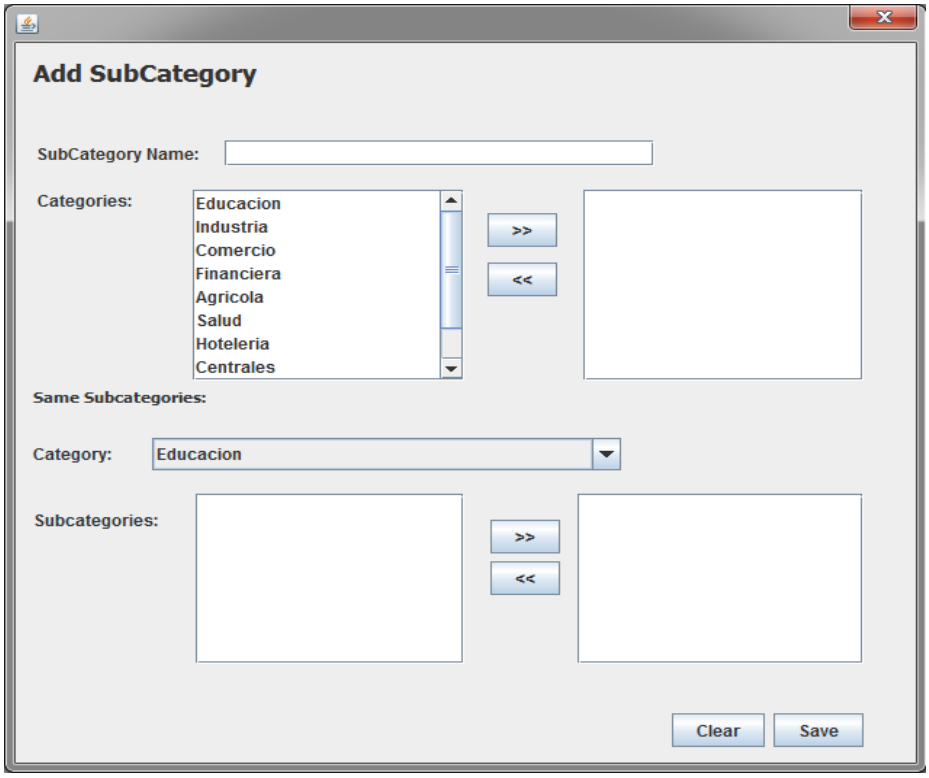


Figura 4.17: Interfaz Gráfica para agregar subcategorías

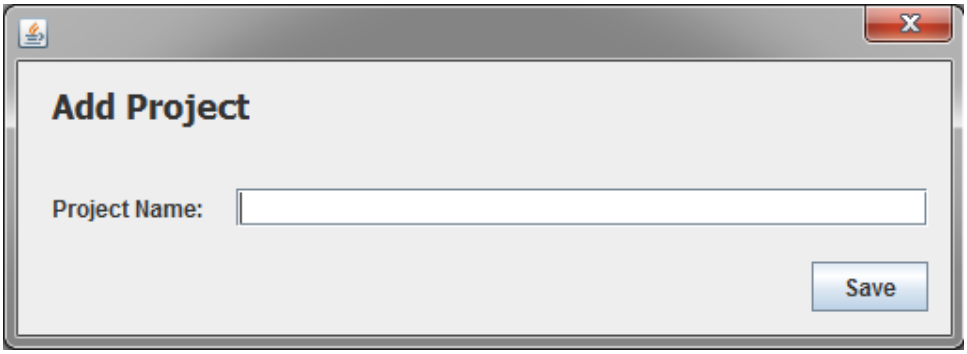


Figura 4.18: Interfaz Gráfica para agregar proyectos

4.7. Análisis Comparativo con Trabajos Relacionados

Esta sección tiene la finalidad de establecer las similitudes y diferencias entre nuestro trabajo y los trabajos relacionados expuestos en el capítulo 3. Para tales efectos se presenta el cuadro 4.9.

Trabajo	Similitudes	Diferencias
T1	■ Se trabaja con diagramas UML de comportamiento ■ Se utiliza una ontología para almacenar la información del diagrama tratado ■ Se utiliza el estándar XMI para trasladar los conceptos desde el modelo UML a la ontología	■ El objetivo no es la reutilización de artefactos de software sino simplemente la representación ontológica del diagrama ■ Se utilizan diagramas de actividades, no de casos de uso
T2	■ Se trabaja con diagramas UML de comportamiento ■ El objetivo es la reutilización de artefactos de software generados en la fase de análisis del desarrollo de software	■ La metodología para lograr la reutilización no emplea un enfoque basado en ontologías ■ Se utilizan diagramas de secuencia, no de casos de uso ■ La herramienta de apoyo al método no está basado en tecnologías de Web Semántica

T3	▪ Se trabaja con información asociada a los diagramas de casos de uso ▪ El objetivo es la reutilización de artefactos de software generados en la fase de análisis del desarrollo de software	▪ Se trabaja específicamente con los flujos de eventos de los casos de uso no con la información propiamente derivada de los diagramas de casos de uso ▪ La metodología para lograr la reutilización no emplea un enfoque basado en ontologías ▪ La herramienta de apoyo al método no está basado en tecnologías de Web Semántica
T4	▪ El objetivo es la reutilización de artefactos de software ▪ Se emplea un enfoque basado en ontologías y tecnologías de Web Semántica	▪ No está orientado a la reutilización propia de diagramas de casos de uso ▪ La herramienta de apoyo al método no está implementada con Jena sino con el API KAON2
T1: Representación de los Modelos UML de Actividad como Ontologías T2: Recuperación Automática de Diagramas UML de Secuencia Reutilizables T3: Reutilización de Especificaciones UML en un Dominio de Aplicación Restringido T4: KOntoR: Un Enfoque de Reutilización de Software basado en Ontologías		

Tabla 4.9: Comparación con los trabajos relacionados

Capítulo 5

Pruebas y Evaluación

En este capítulo será presentada la validación del sistema propuesto a través de una serie de pruebas.

Esta fase de pruebas tiene como objetivo verificar el sistema de software para comprobar si este cumple sus requisitos. En este caso, dado que el sistema desarrollado consiste en una aplicación con interfaz gráfica, se deben realizar pruebas funcionales que verifiquen que el sistema de software ofrece las funcionalidades establecidas en su especificación.

La prueba consiste en trabajar con un conjunto de diagramas de casos de uso que definen funcionalidades de diversas áreas de conocimientos, registrarlos en el sistema de reutilización y posteriormente realizar diferentes búsquedas y verificar si se obtienen los resultados lógicamente deseables.

5.1. Definición de Escenarios

La muestra de diagramas de casos de uso a utilizar para el caso de estudio es de seis diagramas sencillos de diferentes áreas de conocimiento. Consideramos que no es necesario emplear una cantidad elevada de diagramas para estas pruebas puesto que como se definió en el alcance de este trabajo la idea es validar la funcionalidad del sistema.

A continuación se muestran unos cuadros que presentan y especifican los diagramas de prueba y la información adicional a ser ingresada en el sistema:

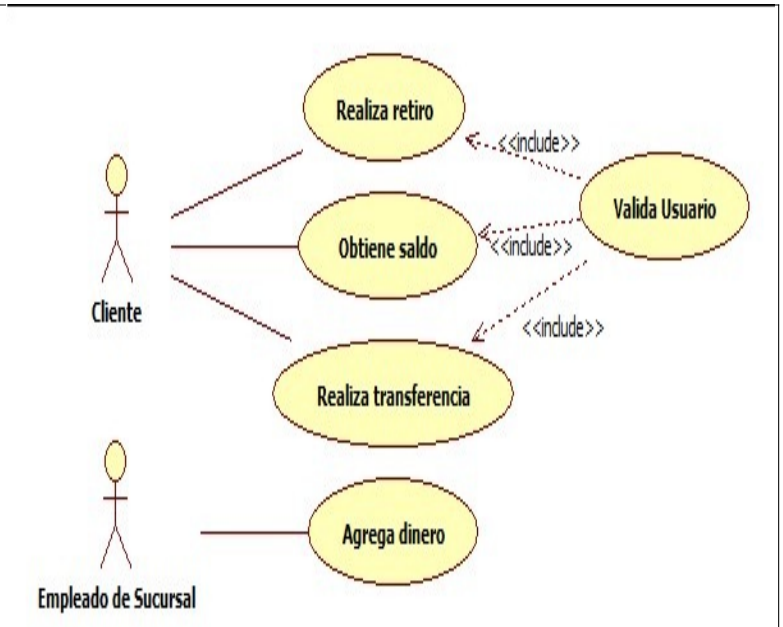
Diagrama	
Título	Interacción con Cajero Automático
Creador	Luis Andrades
Fecha de Creación	01/02/2012
Categorías	Banca
Proyecto	Sistema de Transacciones Bancarias
Comentarios	Diagrama de Casos de Uso para describir la funcionalidad de interacción entre un cajero automático y los clientes

Tabla 5.1: Diagrama de Prueba 1

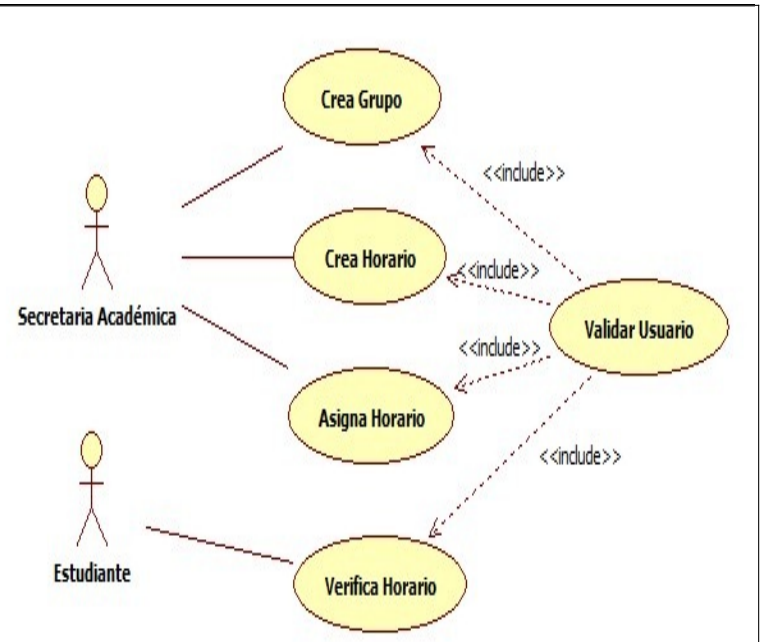
Diagrama	
Título	Gestión de Horarios Académicos
Creador	Sandra Vélez
Fecha de Creación	02/02/2012
Categorías	Universidad, Escolar
Proyecto	Sistema de Gestión Universitaria
Comentarios	Diagrama de Casos de Uso para describir la funcionalidad de gestión de horarios académicos en universidades

Tabla 5.2: Diagrama de Prueba 2

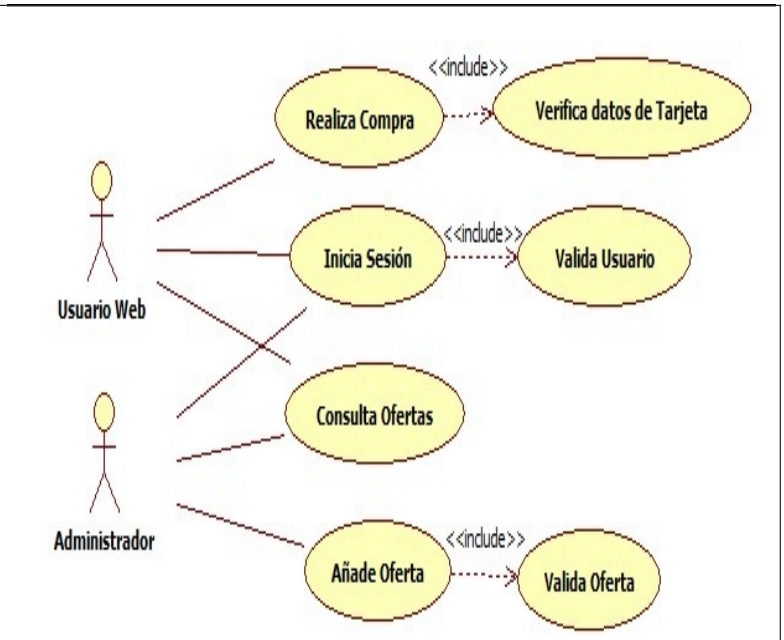
Diagrama	
Título	Compras Online
Creador	Sandra Vélez
Fecha de Creación	06/02/2012
Categorías	Comercio Electrónico, Maquinaria, Alimentos, Muebles, Hotel
Proyecto	Sistema de Ventas Online de Muebles Importados
Comentarios	Diagrama de Casos de Uso para describir la funcionalidad de compras online aplicable en varios sectores económicos

Tabla 5.3: Diagrama de Prueba 3

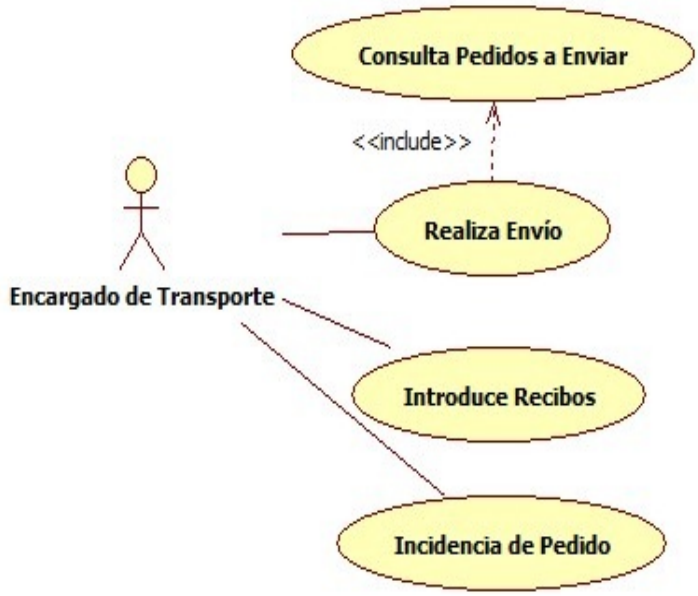
Diagrama	
Título	Funcionalidad de Envíos por Transporte
Creador	Héctor Perez
Fecha de Creación	07/01/2012
Categorías	Transporte, Textiles, Muebles
Proyecto	Sistema de Inventario y Despacho de Productos
Comentarios	Diagrama de Casos de Uso para describir la funcionalidad de envíos de mercancía a través de transporte

Tabla 5.4: Diagrama de Prueba 4

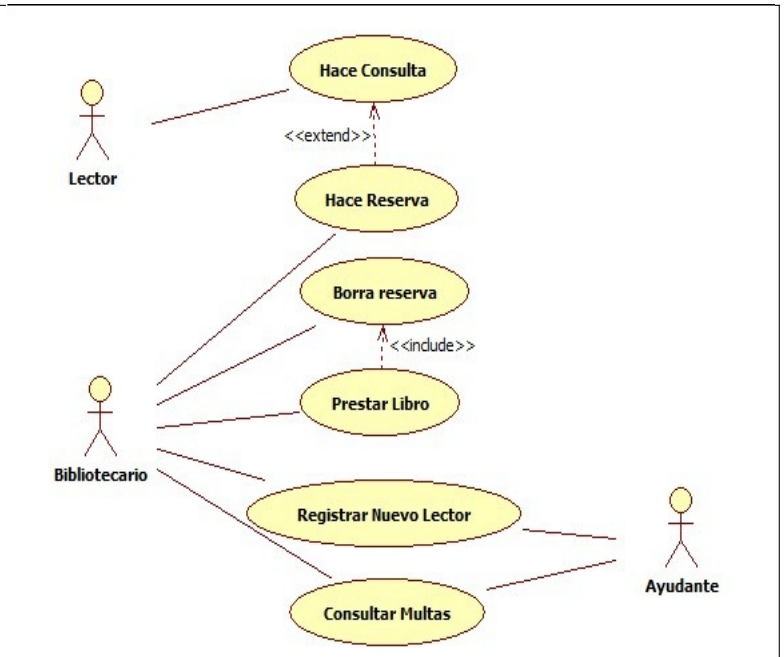
Diagrama	
Título	Funcionalidad de Préstamo de Libros
Creador	Luis Andrades
Fecha de Creación	07/02/2012
Categorías	Universidad, Escolar, Oficina
Proyecto	Sistema de Biblioteca
Comentarios	Diagrama de Casos de Uso para describir la funcionalidad de préstamo de libros en una biblioteca

Tabla 5.5: Diagrama de Prueba 5

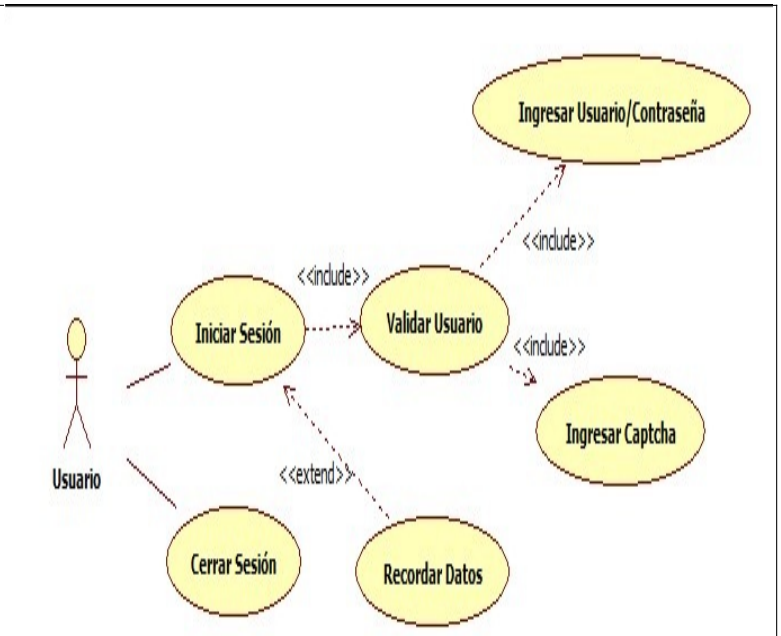
Diagrama	
Título	Funcionalidad de Inicio de Sesión
Creador	Luis Andrades
Fecha de Creación	14/02/2012
Categorías	Universidad, Escolar, Oficina, Transporte, Comercio Electrónico
Proyecto	Sistema de Biblioteca
Comentarios	Diagrama de Casos de Uso para describir la funcionalidad de inicio de sesión en un sistema

Tabla 5.6: Diagrama de Prueba 6

5.2. Ejecución de Pruebas y Análisis de Resultados

Los seis diagramas fueron ingresados a través de la interfaz de usuario para el registro de diagramas de casos de uso. Para verificar el correcto funcionamiento de los procesos que se llevan a cabo durante el registro de los diagramas de caso de uso en el sistema de reutilización, vamos a emplear el cuadro 5.7 que muestra los resultados obtenidos por proceso.

Proceso / Diagrama	Diagrama 1	Diagrama 2	Diagrama 3	Diagrama 4	Diagrama 5	Diagrama 6
Validación de Archivo XMI	Correcto	Correcto	Correcto	Correcto	Correcto	Correcto
Registro de Diagrama	Correcto	Correcto	Correcto	Correcto	Correcto	Correcto
Actualización de Ontología en Base de Datos	Correcto	Correcto	Correcto	Correcto	Correcto	Correcto

Tabla 5.7: Resultados por proceso obtenidos para los diagramas de prueba

Posterior a esta verificación de la funcionalidad de registro de los diagramas en el sistema, se procede a realizar pruebas a la funcionalidad de búsquedas de diagramas de casos de uso a través de la interfaz de usuario dedicada a este propósito. Para ello hemos establecido algunos casos de prueba.

a) Enunciados

- **Caso de Prueba 1**: El usuario busca diagramas de casos de uso que contengan un actor “usuario” o parecido. Ver Figura 5.1.
- **Caso de Prueba 2**: El usuario busca diagramas de caso de uso que contengan un caso de uso “valida usuario” o parecido. Ver Figura 5.2
- **Caso de Prueba 3**: El usuario busca diagramas de caso de uso que pertenezcan a las categorías “comercio electrónico” y “escolar” o parecido. Ver Figura 5.3
- **Caso de Prueba 4**: El usuario busca diagramas de caso de uso contengan en su título la frase “Inicio de sesión” o parecido. Ver Figura 5.4
- **Caso de Prueba 5**: El usuario busca diagramas de caso de uso contengan como actor “usuario” y/o “cliente” o parecido, como caso de uso “validar usuario” y/o “iniciar sesión” o parecido, y como creador alguien con apellido “Andrades” o parecido. Ver Figura 5.5

b) Resultado: El sistema muestra el panel de resultados con la lista de diagramas de casos de uso para cada búsqueda.

c) Conclusión: Pruebas superadas con éxito. El sistema devuelve los resultados que se ajustan a los parámetros introducidos por el usuario y realiza las funciones de inferencia en los casos que aplica de manera correcta.

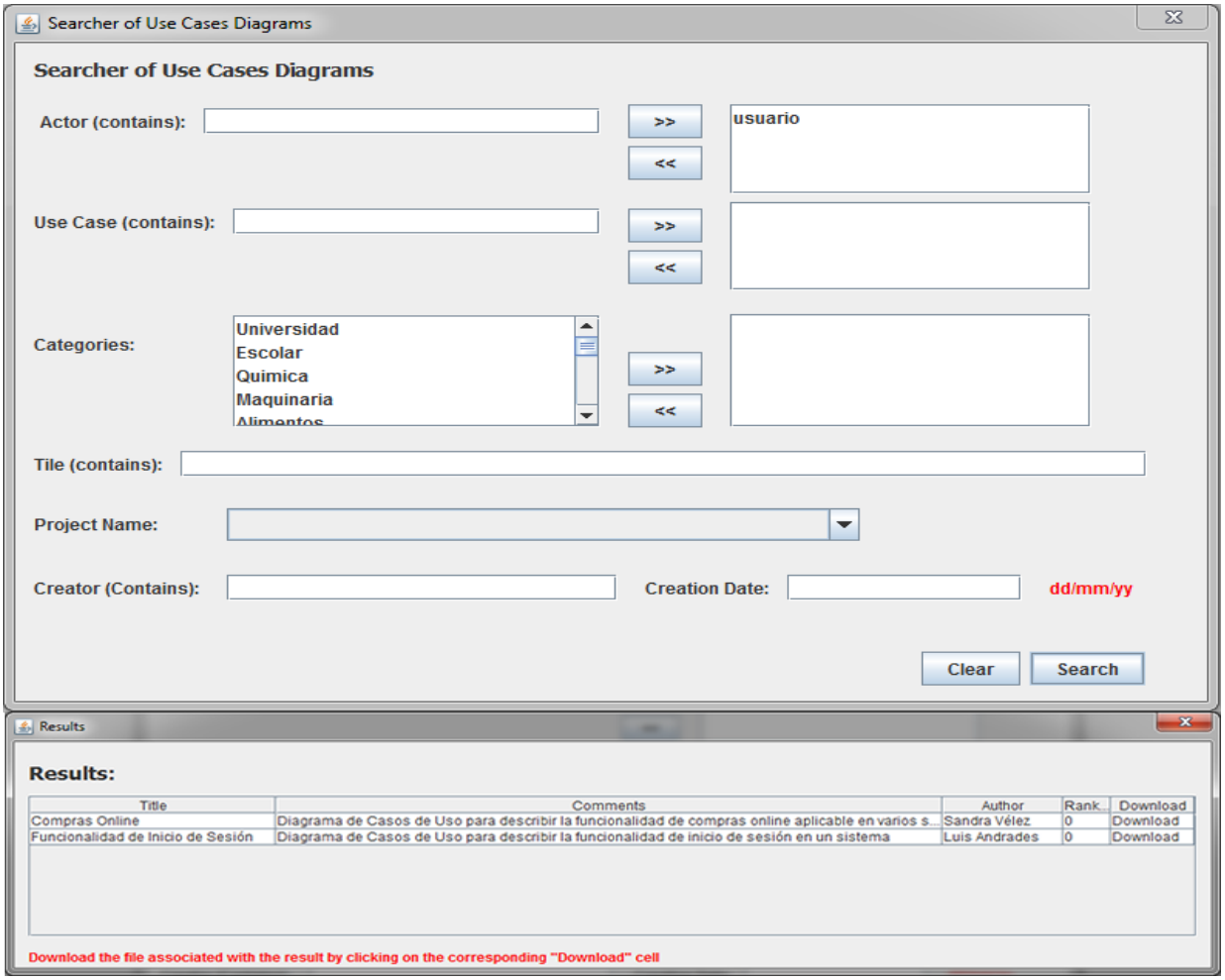


Figura 5.1: Caso de Prueba 1

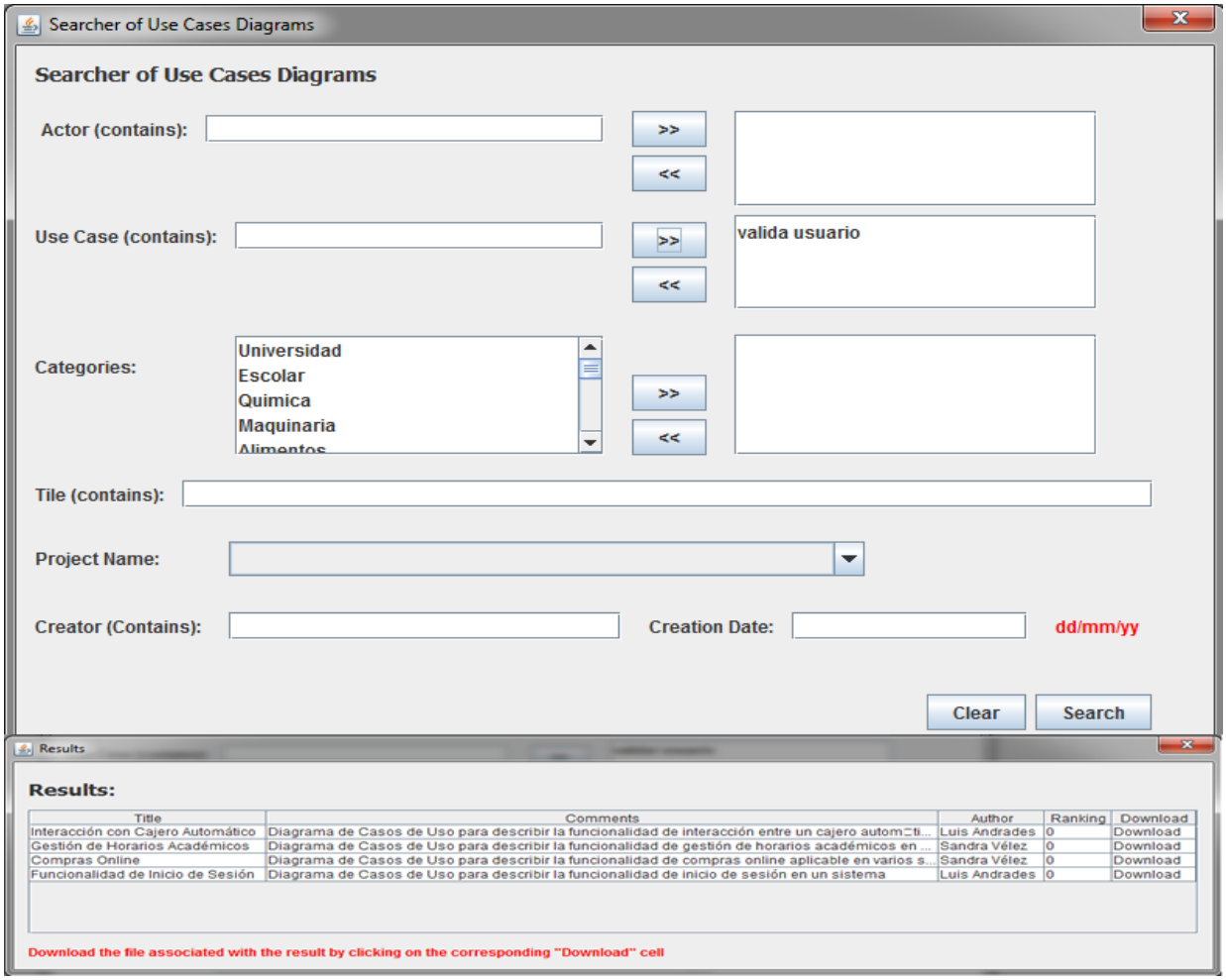


Figura 5.2: Caso de Prueba 2

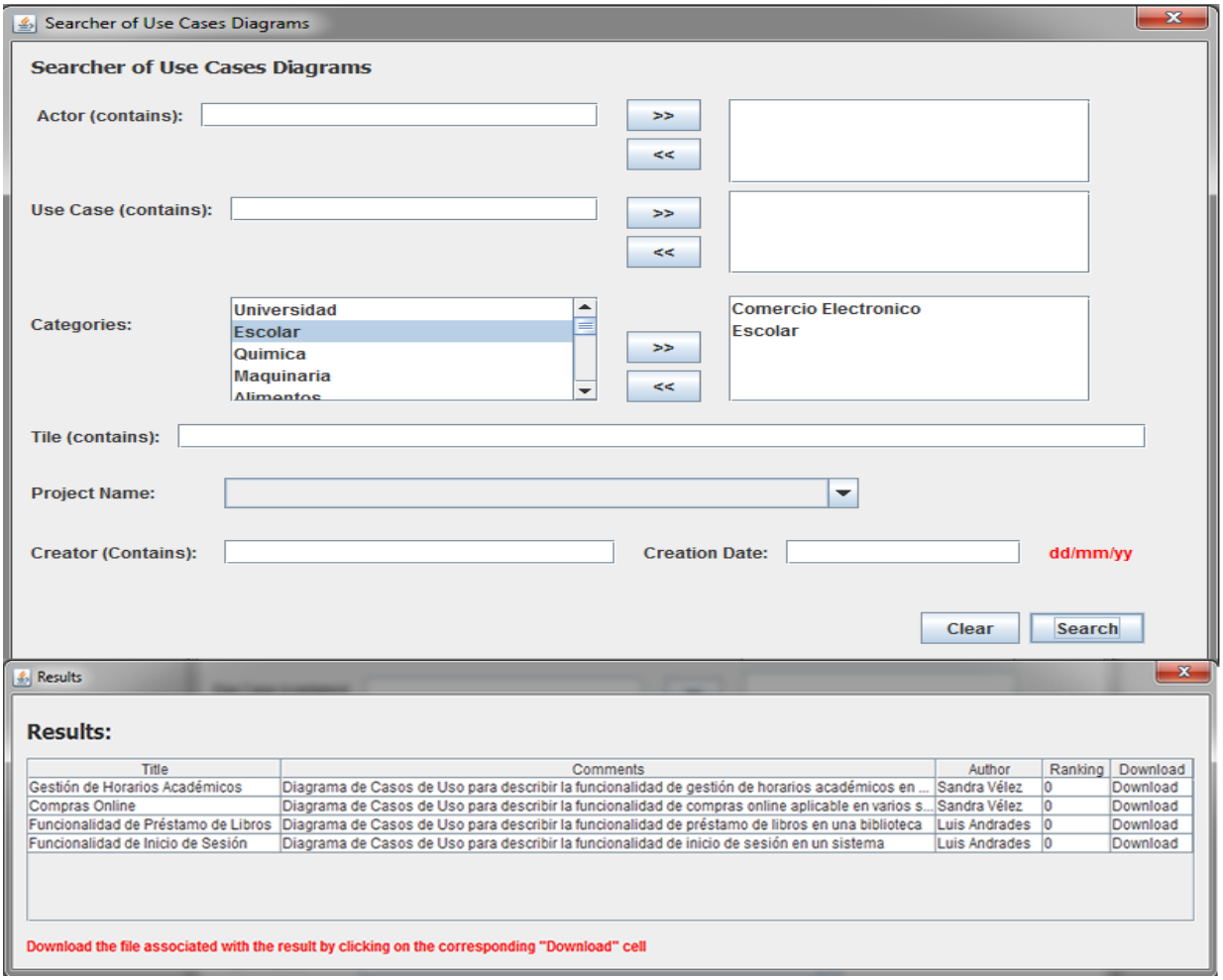


Figura 5.3: Caso de Prueba 3

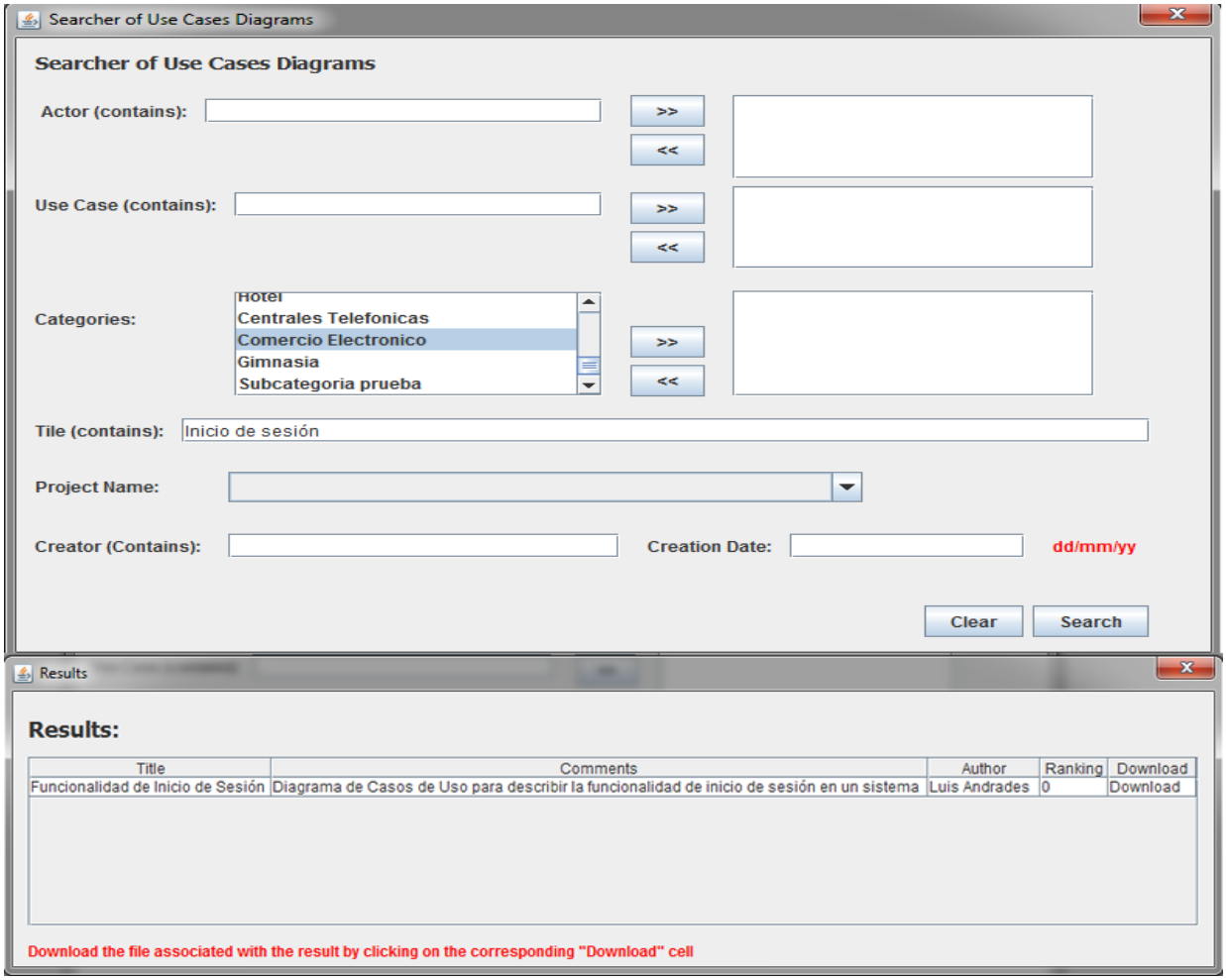


Figura 5.4: Caso de Prueba 4

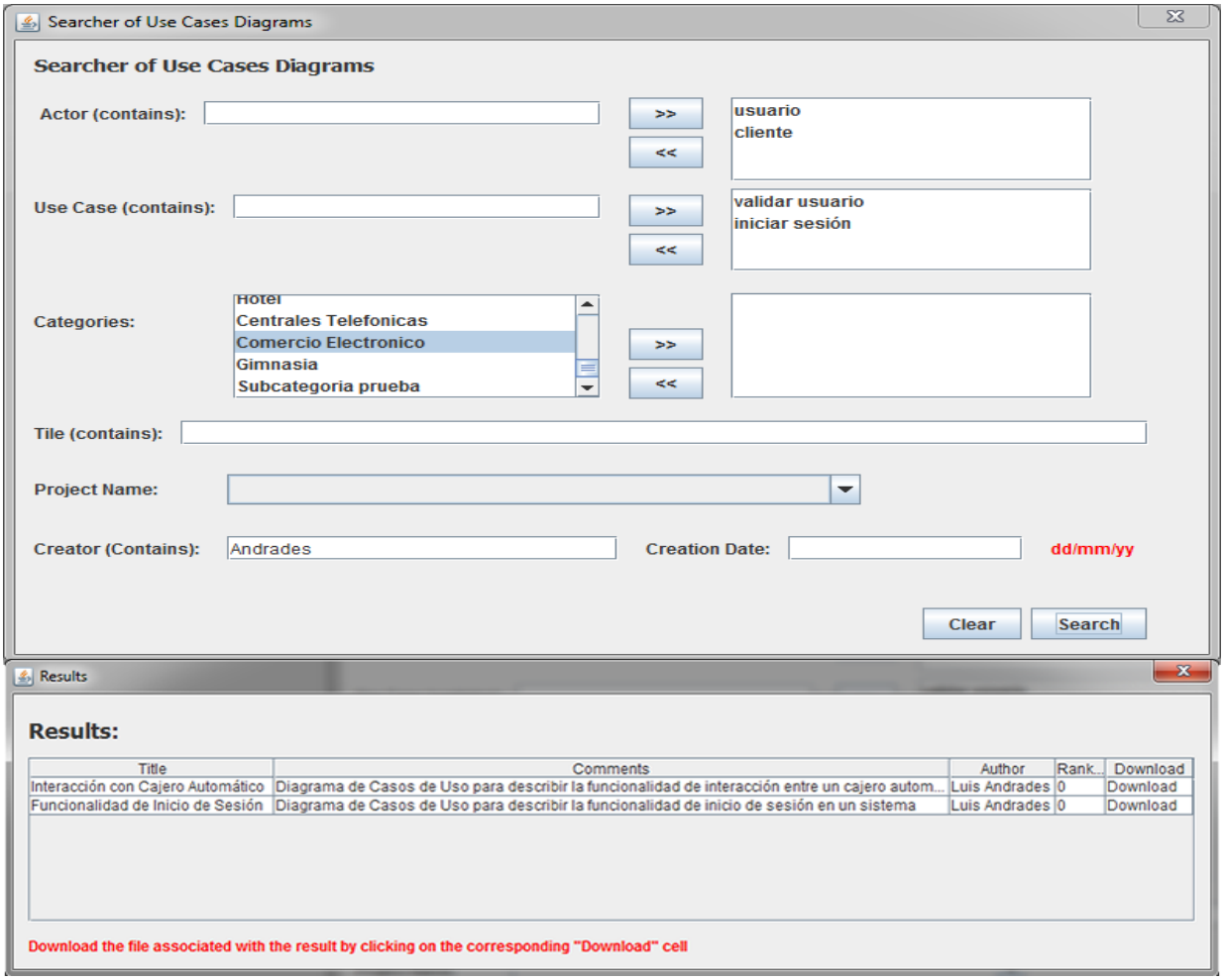


Figura 5.5: Caso de Prueba 5

Con respecto a la verificación de la funcionalidad de descarga de los archivos tras la ejecución de la búsqueda, las Figuras 5.6 y 5.7 muestran el proceso de esta funcionalidad para los resultados del caso de prueba 5. La Figura 5.6 muestra como al seleccionar Download para uno de los resultados, se presenta la ventana que permite escoger una ubicación en el directorio de archivos de la computadora donde se va a proceder a guardar la copia del archivo XMI asociado al resultado, y la Figura 5.7 muestra como tras ejecutar nuevamente una búsqueda que implique al diagrama de caso de uso en formato XMI descargado, el valor de Ranking aumenta agregándole un valor de preferencia en la nueva búsqueda pues ha sido reutilizado con satisfacción.

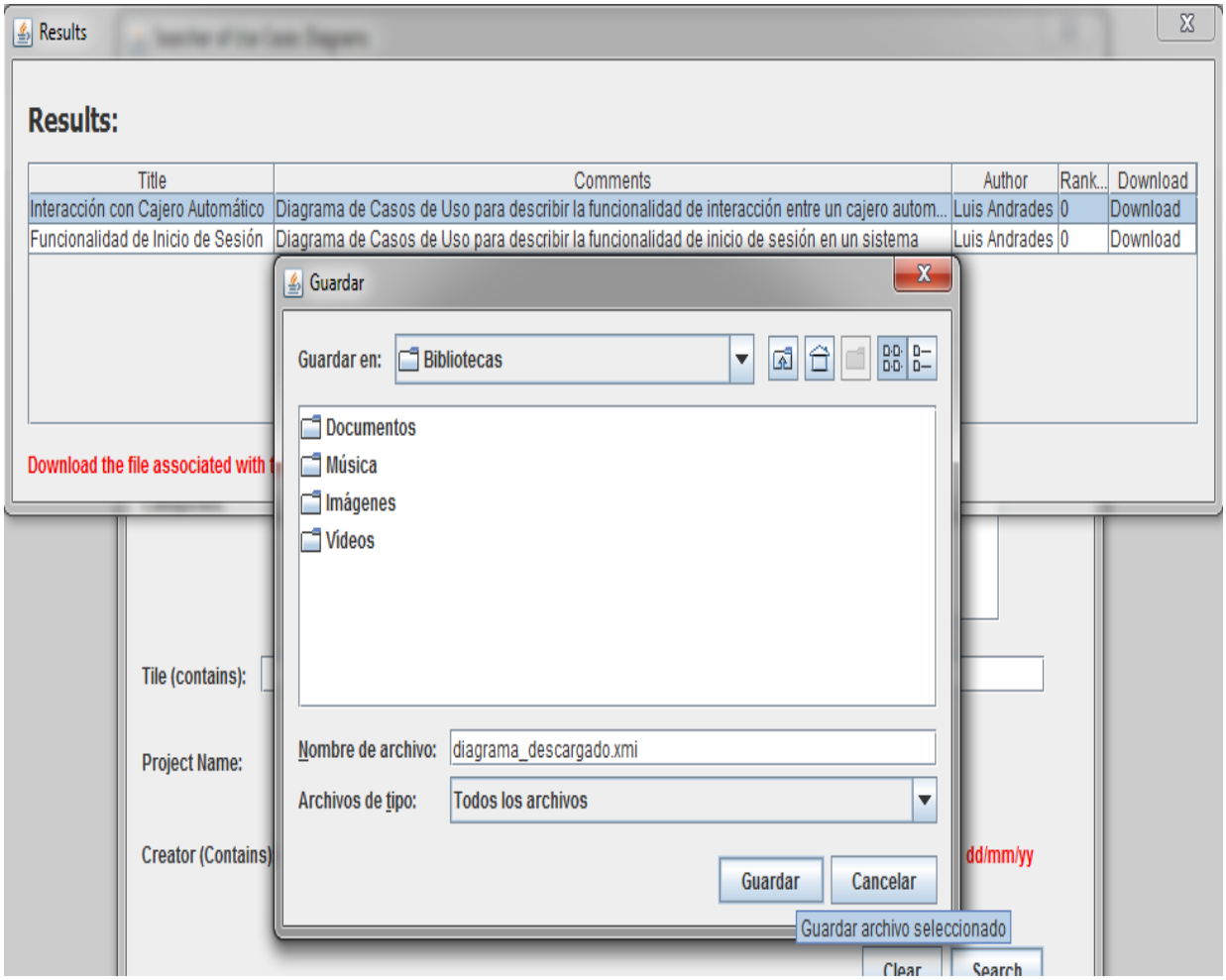


Figura 5.6: Descarga de resultado para el caso de prueba 5

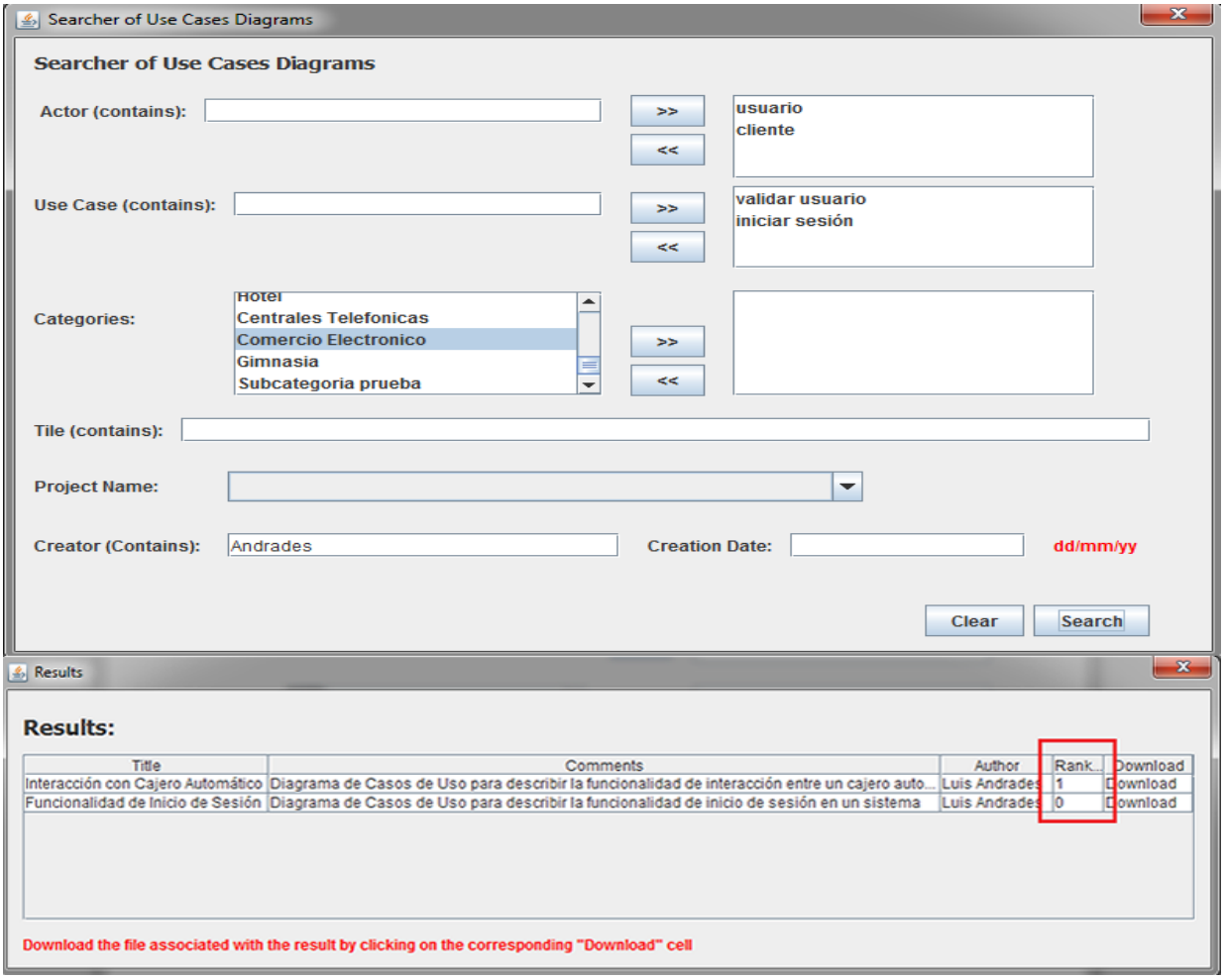


Figura 5.7: Verificación el el aumento de la ponderación para el diagrama de caso de uso descargado

Finalmente, el diagrama descargado puede ser abierto y ejecutado con el apoyo de cualquier herramienta de modelado UML que soporte XMI versión 1.1.

En las Figuras 5.8, 5.9 y 5.10 se puede apreciar como el diagrama de caso de uso descargado para los efectos de esta prueba se puede abrir tanto en StarUML, Rational Rose y ArgoUML respectivamente. Cabe destacar que ArgoUML no muestra el diagrama de forma gráfica pero si se despliegan correctamente los elementos que lo conforman.

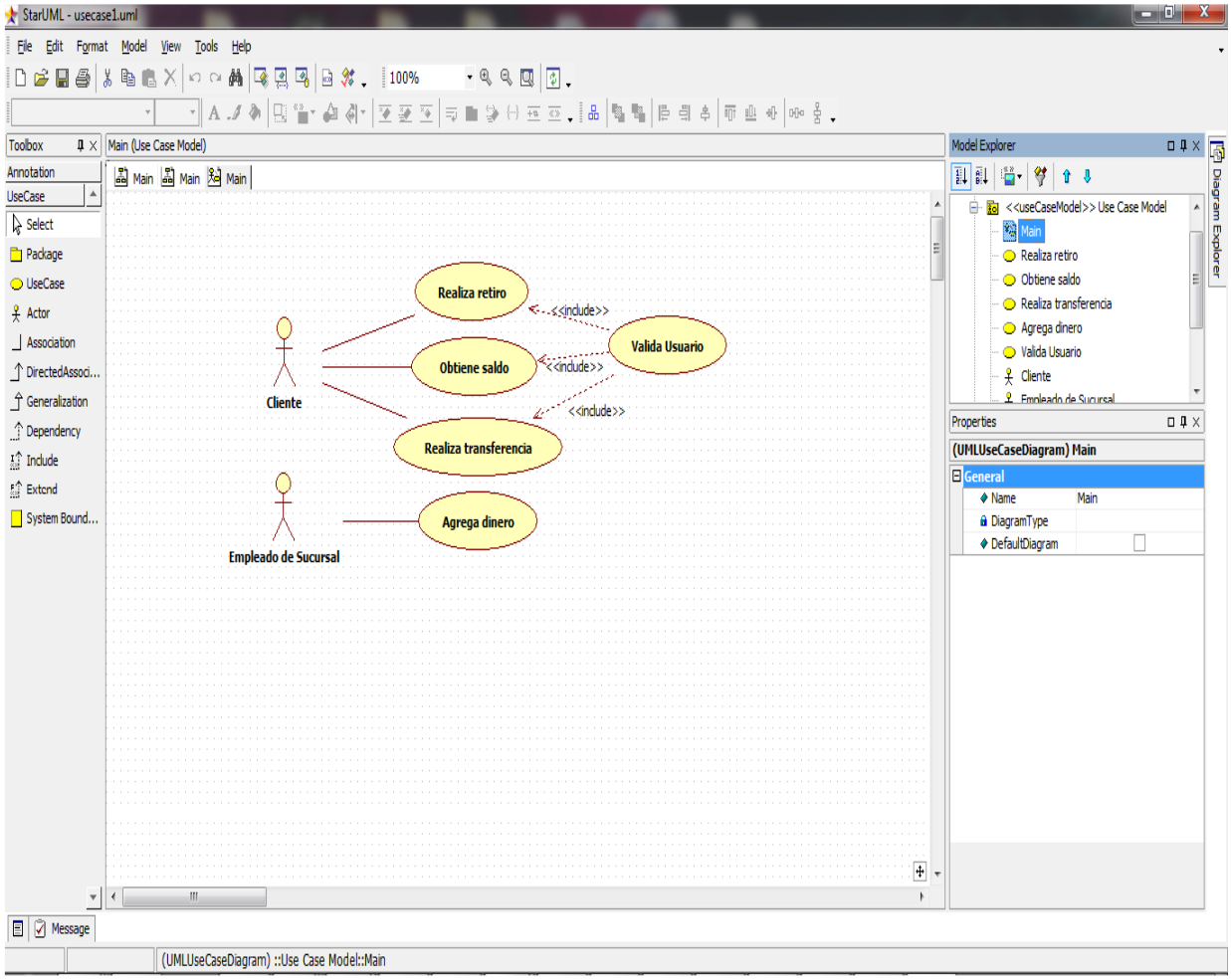


Figura 5.8: Visualización del contenido de archivo XMI recuperado en StarUML

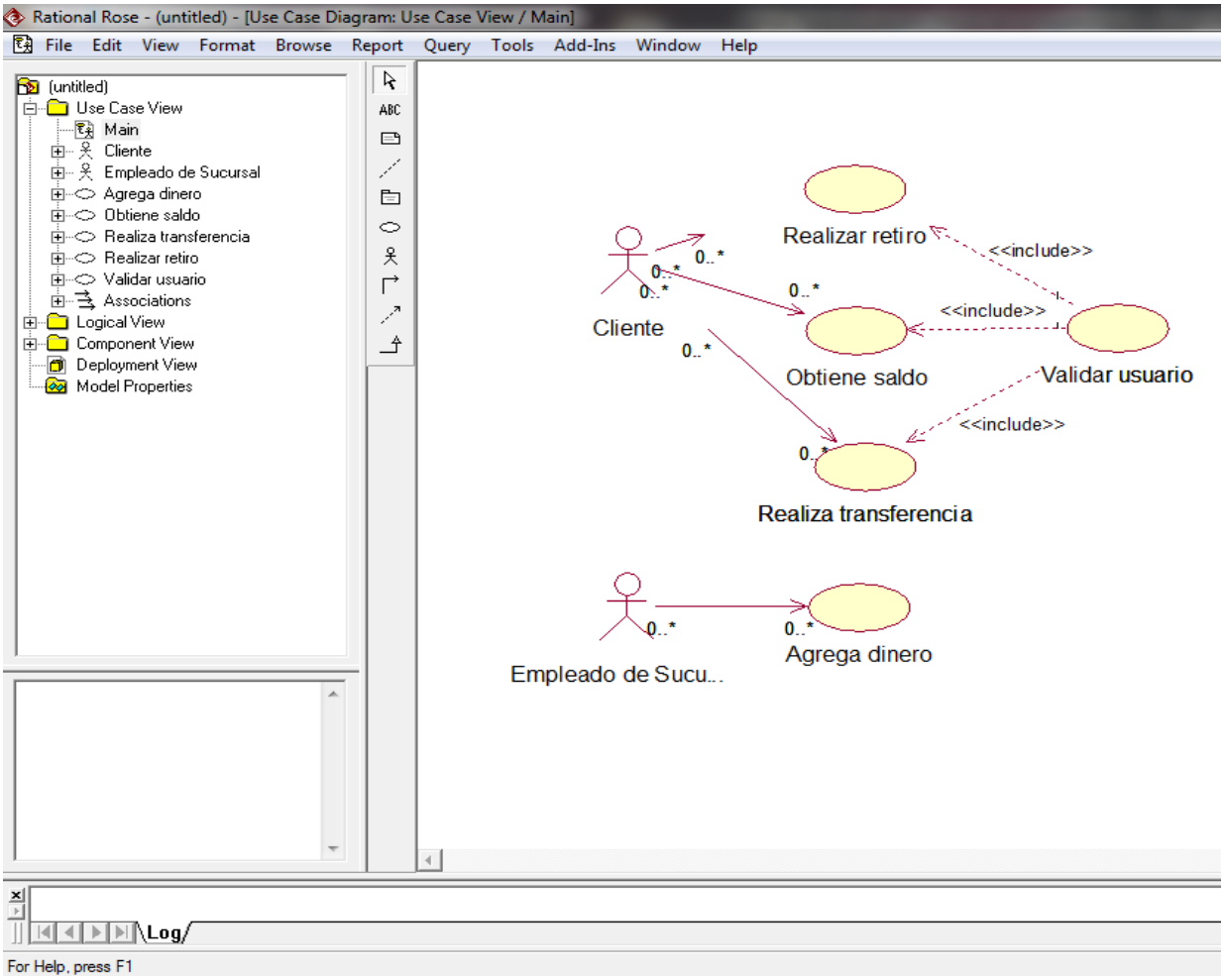


Figura 5.9: Visualización del contenido de archivo XMI recuperado en Rational Rose

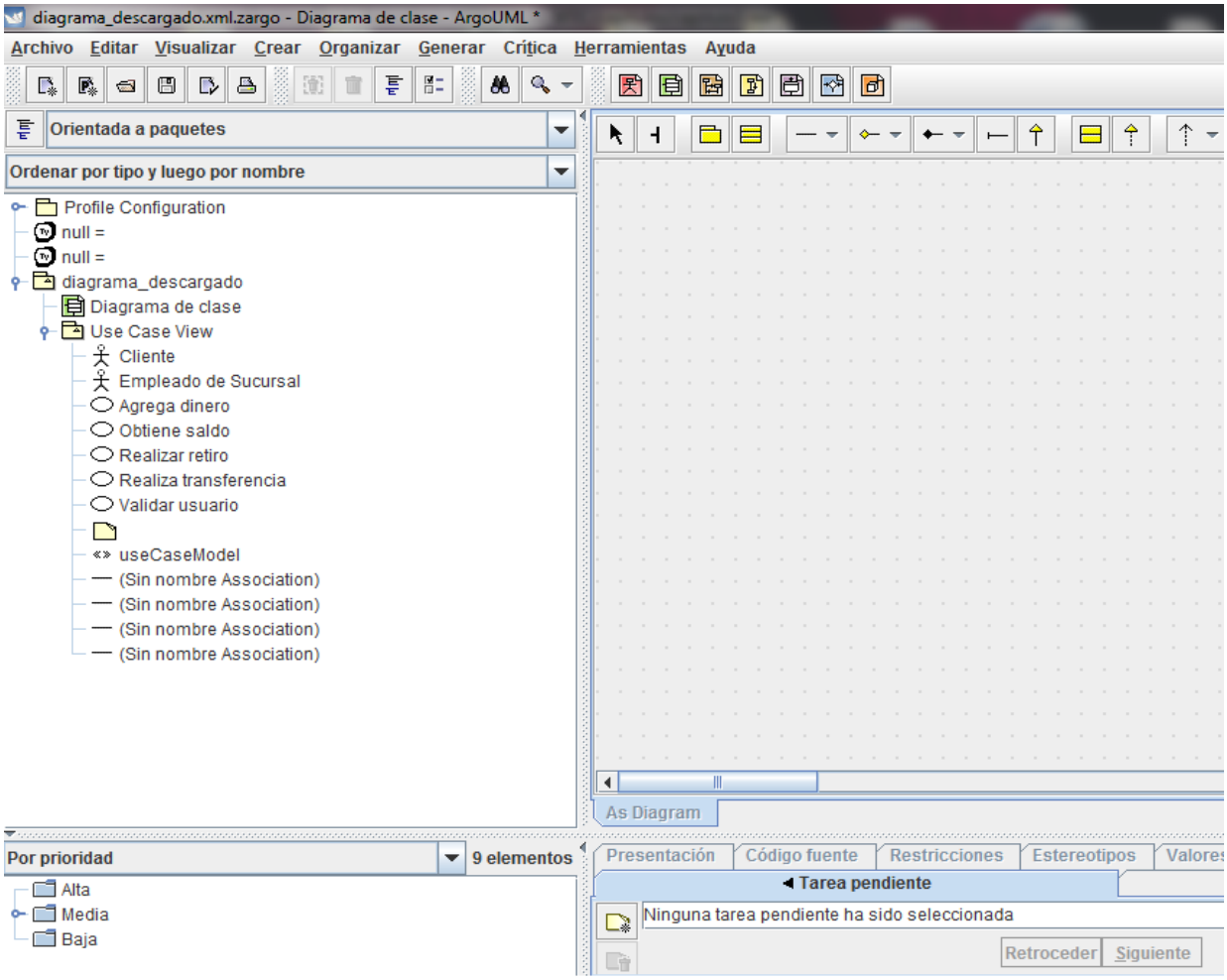


Figura 5.10: Visualización del contenido de archivo XMI recuperado en ArgoUML

A través de las pruebas ejecutadas para determinar el correcto funcionamiento del sistema implementado se puede concluir que el sistema contempla todas las funcionalidades definidas en la especificación de requisitos. Dichas funcionalidades ejecutan un correcto funcionamiento de sus procesos.

En lo que respecta a la ontología, la misma permitió almacenar y organizar la información de cada uno de los diagramas de casos de uso de manera correcta. Aunque en este trabajo no estamos evaluando tiempos de respuesta, cabe destacar que los tiempos de respuesta a las consultas hechas sobre la ontología para nuestra muestra son altamente aceptables, sin embargo, se puede verificar que a medida que el tamaño de la ontología aumenta por la creación de nuevos individuos y se llevan a cabo procesos de inferencia, los tiempos de respuesta en consulta van aumentando levemente.

Con respecto a las salidas obtenidas tras ejecutar las diferentes búsquedas, fueron obtenidos los resultados esperados. De igual forma, se pudo verificar que los archivos XMI para los diagramas de casos de uso que son descargados pueden ser manipulados a través de una herramienta para modelado UML permitiendo un trabajo directo con el diagrama de manera visual.

Capítulo 6

Conclusiones

El sistema de reutilización de diagramas de casos de uso presentado en este trabajo mostró ser viable y en conformidad con los objetivos inicialmente planteados. Esta conclusión es fundamentada en el análisis de los resultados obtenidos en la actividad de pruebas y evaluación donde se verificaron las funcionalidades del sistema y el apoyo que brinda a la reutilización de artefactos de software involucrando un mejoramiento en los tiempos y costos implicados en la etapa de análisis para el desarrollo de un software.

En lo que se refiere a la ontología, caracterizada por mapear los conceptos mínimos y necesarios para representar el dominio de conocimiento de los diagramas de casos de uso, se puede concluir que resulta viable el uso de ontologías en esta y demás aplicaciones orientadas a la ingeniería de software. Sin embargo, conviene observar la dificultad para establecer modelos que atiendan las necesidades de aplicación. Aún empleando metodologías para la construcción de ontologías, es común la confusión con modelos de tipo entidad-relación cuando la ontología es creada en su mayoría para fines taxonómicos, lo que demanda una serie de procesos incrementales hasta obtener el modelo final. Las tecnologías OWL, SPARQL y Jena 2 mostraron ser robustas, pero la curva de aprendizaje de estas herramientas no es rápida, hay pocos ejemplos prácticos y son necesarios buenos conocimientos de orientación a objetos, Java y XML.

Con respecto a la base de datos ontológica se puede concluir que almacenar la ontología y sus individuos de manera persistente garantiza un mejor desempeño en las

consultas SPARQL que son realizadas sobre la ontología, esto se valida a través de las bases teóricas que indican que el almacenamiento persistente en bases de datos supone grandes ventajas sobre el almacenamiento en memoria y el almacenamiento tradicional en el sistema de ficheros. Además, el uso de una base de datos para almacenar la ontología provee mayor garantía de disponibilidad de la información y mayor seguridad en cuanto a pérdida de información en comparación a los enfoques en memoria o ficheros sobre los cuales no se pueden establecer políticas específicas de acceso y manipulación. Cabe destacar también los beneficios obtenidos al utilizar Jena 2, el cual facilita la creación del modelo en la base de datos al definir las tablas necesarias para el almacenamiento de la información de la ontología en formato de tripletas RDF; con esta característica no hay necesidad de manipular la estructura de la base de datos por lo cual se evitan problemas de integridad e inconsistencia. En el ámbito tecnológico hubo perfecta interacción entre Jena 2, SPARQL y MySQL para esta tarea de persistencia.

Finalmente, en lo que se refiere a los prototipos de interfaces para la herramienta del sistema propuesto se puede concluir que las mismas poseen los criterios mínimos y necesarios para ejecutar las funcionalidades de registro, búsqueda y recuperación de diagramas de casos de uso. Cabe destacar que para su diseño e implementación no fueron tomados en consideración criterios de usabilidad y ergonomía, pero como una interface elaborada no está dentro de los objetivos de este trabajo no hubo impacto sobre los resultados generados. Otra consideración a resaltar es que los tiempos de respuesta asociados a las consultas SPARQL que implican inferencia pueden afectar la rapidez con que se presenta en determinado momento una interfaz de usuario, sin embargo, el mejoramiento de tiempos de respuesta tampoco está dentro de los objetivos de este trabajo por lo cual no hay afectación sobre los resultados generados, pero si se considerara la posibilidad de construcción de un producto de software propiamente dicho en su momento habría que evaluar el nivel de impacto que tendría el factor tiempo de respuesta sobre los objetivos de reutilización del producto y determinar su viabilidad.

6.1. Trabajos Futuros

Con el objetivo de dar continuidad al objeto de investigación presentado en este trabajo, se planea llevar a cabo un trabajo de campo extenso dentro de una fábrica de software, de tal manera que se pueda aplicar el sistema que hemos propuesto y verificar su viabilidad en un contexto de desarrollo de software más real. Con este trabajo de campo se podría obtener estadísticas concretas de comportamiento en cuanto a tiempo y esfuerzo invertidos durante la etapa de análisis de software haciendo uso del sistema propuesto versus el enfoque tradicional sin sistema de apoyo a la reutilización.

Por otro lado, y dependiendo de los resultados obtenidos a través del trabajo de campo, se propone la expansión del sistema para que permita trabajar con todos los artefactos UML derivados del desarrollo de un software, por lo cual el enfoque sería ampliado a nivel de proyecto y no solo de un artefacto de software en particular. La ontología sería reestructurada para brindar soporte a los nuevos conceptos a manejar y sería necesario la introducción de nuevas funcionalidades e interfaces de usuario.

Referencias

BALABAN, M. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, vol. 15, páginas 15–19, 1995.

BERNARAS, A., LARESGOITI, I. y CORERA, J. Building and reusing ontologies for electrical network applications. En *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI'96)* (editado por W. Wahlster), páginas 298–302. John Wiley and Sons, Chichester, UK, 1996. ISBN 0471968099.

BERNERS-LEE, T., HENDLER, J. y LASSILA, O. The semantic web. *Scientific American*, vol. 284(5), páginas 34–43, 2001.

BLOK, M. C. y CYBULSKI, J. L. Reusing uml specifications in a constrained application domain. En *Proceedings of the Fifth Asia Pacific Software Engineering Conference*, APSEC '98, páginas 196–. IEEE Computer Society, Washington, DC, USA, 1998. ISBN 0-8186-9183-2.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E. y YERGEAU, F. Extensible markup language (XML) 1.0 (third edition), W3C recommendation. Informe técnico, W3C, 2004.

CHAUDHRI, V. K., FARQUHAR, A., FIKES, R., KARP, P. D. y RICE, J. P. Okbc: a programmatic foundation for knowledge base interoperability. En *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, páginas 600–607. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1998. ISBN 0-262-51098-7.

CORCHO, O., FERNANDEZ-LOPEZ, M. y GÓMEZ-PÉREZ, A. Ontological engineering-principles, methods, tools and languages. *Ontologies for Software Engineering and Software Technology*, página 48, 2006.

DUCHARME, B. *Learning SPARQL*. O'Reilly Media, 2011. ISBN 9781449306595.

FARQUHAR, A., FIKES, R. y RICE, J. The ontolingua server: A tool for collaborative ontology construction. 1996.

FERNANDEZ-LOPEZ, M., GOMEZ-PEREZ, A. y JURISTO, N. Methontology: from ontological art towards ontological engineering. En *Proceedings of the AAAI97 Spring Symposium*, páginas 33–40. Stanford, USA, 1997.

GRÜNINGER, M. y FOX, M. S. Methodology for the design and evaluation of ontologies. En *International Joint Conference on Artificial Inteligence (IJCAI95), Workshop on Basic Ontological Issues in Knowledge Sharing*. 1995.

GRUBER, T. R. A translation approach to portable ontology specifications. *Knowl. Acquis.*, vol. 5, páginas 199–220, 1993. ISSN 1042-8143.

GUARINO, N. Formal ontology and information systems. páginas 3–15. IOS Press, 1998.

GUHA, R. V. y BRICKLEY, D. RDF Vocabulary Description Language 1.0: RDF Schema. W3C recommendation, W3C, 2004.

HAPPEL, H.-J., KORTHAUS, A., SEEDORF, S. y TOMCZYK, P. Kontor: An ontology-enabled approach to software reuse. En *IN: PROC. OF THE 18TH INT. CONF. ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING*. 2006.

HEBELER, J., FISHER, M., BLACE, R., PEREZ-LOPEZ, A. y DEAN, M. *Semantic Web Programming*. John Wiley & Sons Inc., Chichester, West Sussex, Hoboken, NJ, 2009. ISBN 978-0-470-41801-7.

HEFLIN, J. Owl web ontology language - use cases and requirements. 2010. `http://www.w3.org/TR/webont-req/`.

VAN HEIJST, G., SCHREIBER, A. T. y WIELINGA, B. J. Using explicit ontologies in kbs development. *Int. J. Hum.-Comput. Stud.*, vol. 46, páginas 183–292, 1997. ISSN 1071-5819.

KHAN, A. H., MINHAS, A. A. y NIAZI, M. F. Representation of uml activity models as ontology. En *In Proceedings of 5th International Conference on Innovations in Information Technology*. 2008.

KIM, Y. y STOHR, E. A. Software reuse: survey and research directions. *J. Manage. Inf. Syst.*, vol. 14, páginas 113–147, 1998. ISSN 0742-1222.

KRUEGER, C. W. Software reuse. *ACM Comput. Surv.*, vol. 24, páginas 131–183, 1992. ISSN 0360-0300.

MANOLA, F. y MILLER, E. RDF Primer. 2004.

MCGUINNESS, D. L., FIKES, R., HENDLER, J. y STEIN, L. A. Daml+oil: An ontology language for the semantic web. *IEEE Intelligent Systems*, vol. 17, páginas 72–80, 2002. ISSN 1541-1672.

MCGUINNESS, D. L. y VAN HARMELEN, F. Owl web ontology language overview. Informe Técnico REC-owl-features-20040210, W3C, 2004.

MIZOGUCHI, R., VANWELKENHUYSEN, J. y IKEDA, M. *Task ontology for reuse of problem solving knowledge*, páginas 46–57. IOS Press, 1995.

MOTTA, E. An overview of the ocml modelling language. En *In Proceedings KEML'98: 8th Workshop on Knowledge Engineering Methods and Languages*, páginas 21–22. 1998.

ÖVERGAARD, G. y PALMKVIST, K. A formal approach to use cases and their relationships. En *Selected papers from the First International Workshop on The Unified Modeling Language «UML»'98: Beyond the Notation*, páginas 406–418. Springer-Verlag, London, UK, 1999. ISBN 3-540-66252-9.

PRESSMAN, R., MURRIETA, J., ROJAS, E. y OLGUÍN, V. *Ingeniería del software: un enfoque práctico*. McGraw-Hill, 2006. ISBN 9789701054734.

ROBINSON, W. N. y WOO, H. G. Finding reusable uml sequence diagrams automatically. *IEEE Softw.*, vol. 21, páginas 60–67, 2004. ISSN 0740-7459.

SWARTOUT, B., PATIL, R., KNIGHT, K. y RUSS, T. Towards distributed use of large-scale ontologies. En *Proceedings of the 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff, Canada, 1996.

USCHOLD, M. y KING, M. Towards a methodology for building ontologies. En *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*. Montreal, Canada, 1995.

W3C. 2001. http://www.w3.org/2001/sw/.

YU, L. *A Developers Guide to the Semantic Web*. Springer Publishing Company, Incorporated, 1st edición, 2011. ISBN 3642159699, 9783642159695.

Anexo 1

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY diagramas "diagramas:" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY diagramas2 "file:/C:/Users/Belen/Desktop/ontologia_base.owl
    #diagramas:" >
]>
<rdf:RDF xmlns="file:/C:/Users/Belen/Desktop/ontologia_base.owl#"
     xml:base="file:/C:/Users/Belen/Desktop/ontologia_base.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:diagramas="diagramas:"
     xmlns:diagramas2="file:/C:/Users/Belen/Desktop/ontologia_base.owl
     #diagramas:">
    <owl:Ontology rdf:about=""/>
    <!--
    ////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ////////////////////////////////////////////////////
    <!-- diagramas:tieneActor -->
    <owl:ObjectProperty rdf:about="diagramas:tieneActor">
        <rdfs:range rdf:resource="diagramas:Actor"/>
        <rdfs:domain rdf:resource="diagramas:Diagrama"/>
```

```
</owl:ObjectProperty>

<!-- diagramas:tieneCasoUso -->

<owl:ObjectProperty rdf:about="diagramas:tieneCasoUso">
    <rdfs:range rdf:resource="diagramas:CasoUso"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
</owl:ObjectProperty>

<!-- diagramas:tieneCategoria -->

<owl:ObjectProperty rdf:about="diagramas:tieneCategoria">
    <rdfs:range rdf:resource="diagramas:Categoria"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
</owl:ObjectProperty>

<!-- diagramas:tieneProyecto -->

<owl:ObjectProperty rdf:about="diagramas:tieneProyecto">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
    <rdfs:range rdf:resource="diagramas:Proyecto"/>
</owl:ObjectProperty>
<!--
///////////////////////////////////////////////////////
//
// Data properties
//
///////////////////////////////////////////////////////
 -->
<!-- diagramas:tieneCreador -->

<owl:DatatypeProperty rdf:about="diagramas:tieneCreador">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
```

```
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- diagramas:tieneId -->

<owl:DatatypeProperty rdf:about="diagramas:tieneId">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
    <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<!-- diagramas:tienePonderacion -->

<owl:DatatypeProperty rdf:about="diagramas:tienePonderacion">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
    <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>

<!-- diagramas:tieneTitulo -->

<owl:DatatypeProperty rdf:about="diagramas:tieneTitulo">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- diagramas:tienefechaCreacion -->

<owl:DatatypeProperty rdf:about="diagramas:tienefechaCreacion">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="diagramas:Diagrama"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
    <!-- file:/C:/Users/Belen/Desktop/ontologia_base.owl#
    diagramas:tieneNombre -->

    <owl:DatatypeProperty rdf:about="#diagramas:tieneNombre">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="diagramas:Actor"/>
        <rdfs:domain rdf:resource="diagramas:CasoUso"/>
        <rdfs:domain rdf:resource="diagramas:Categoria"/>
        <rdfs:domain rdf:resource="diagramas:Diagrama"/>
        <rdfs:domain rdf:resource="diagramas:Proyecto"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <!--
    ///////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////
     -->
    <!-- diagramas:Actor -->
    <owl:Class rdf:about="diagramas:Actor"/>

    <!-- diagramas:CasoUso -->
    <owl:Class rdf:about="diagramas:CasoUso"/>

    <!-- diagramas:Categoria -->
    <owl:Class rdf:about="diagramas:Categoria"/>

    <!-- diagramas:Diagrama -->
    <owl:Class rdf:about="diagramas:Diagrama"/>

    <!-- diagramas:Proyecto -->
    <owl:Class rdf:about="diagramas:Proyecto"/>
</rdf:RDF>
     -->
```