UNIVERSISAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

MAESTRÍA EN CIENCIAS DE TECNOLOGÍA DE LA INFORMACIÓN Y COMUNICACIÓN

UNA ARQUITECTURA PARA LA REUTILIZACIÓN DE INFORMACIÓN DE PROCESOS DE SOFTWARE EN UN AMBIENTE MULTI-AGENTE

ASESORES

DR. SERGIO CRESPO

DR. CLIFTON CLUNIE

POR

MARIO ALEXANDER RAMOS MORÁN

2012

MARIO ALEXANDER RAMOS MORÁN

UNA ARQUITECTURA PARA LA REUTILIZACIÓN DE INFORMACIÓN DE PROCESOS DE SOFTWARE EN UN AMBIENTE MULTI-AGENTE

Universidad Tecnológica de Panamá
Facultad de Ingeniería de Sistemas Computacionales
Maestría en Ciencias de Tecnología de la Información y Comunicación

Panamá
2012

MARIO ALEXANDER RAMOS MORÁN

UNA ARQUITECTURA PARA LA REUTILIZACIÓN DE INFORMACIÓN DE PROCESOS DE SOFTWARE EN UN AMBIENTE MULTI-AGENTE

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

Maestría en Ciencias de Tecnología de Información y Comunicación

Asesores:

Dr. Sergio Crespo
Dr. Clifton Clunie

Panamá
2012

Esta tesis de está

dedicada a mi familia por el apoyo

brindado durante el todo el tiempo

que duró la maestría.

Gracias a la Secretaría Nacional de Ciencia y Tecnología por el apoyo brindado y al Dr. Sergio Crespo por la orientación brindada a lo largo de la maestría..

Resumen

La complejidad y magnitud de los proyectos de desarrollo de software se ha ido incrementando considerablemente en los últimos años, lo cual hace cada vez más evidente la necesidad de reutilizar componentes o conocimientos adquiridos en base a la experiencia, ya sea por medio de patrones de diseño, librerías, buenas prácticas del desarrollo de software o modelos del negocio, que faciliten y permitan desarrollar los proyectos de la mejor manera posible.

La necesidad de reutilizar software y(o) conocimiento ha convertido la reutilización de componentes en uno de los retos más comunes que afronta el desarrollo de software actual. Muchos de los enfoques abarcados han sido dedicados al reúso de componentes de software, por medio de librerías, componentes auto adheribles (plugins) o patrones de diseño, pero a excepción de los patrones de diseño, los demás componentes son dependientes de tecnologías específicas e incapaces de prevalecer a lo largo del tiempo y la evolución tecnológica. De la misma forma, muchas organizaciones dedicadas al desarrollo de software han adquirido gran experiencia y han creado repositorios donde almacenan las arquitecturas y los modelos utilizados en el desarrollo de proyectos de software, pero no cuentan con los mecanismos adecuados para aprovechar el conocimiento que han acumulado a lo largo del tiempo.

Este trabajo propone e implementa una arquitectura denominada ARIPSAMA que permite la aplicación de métodos de búsqueda y extracción de conceptos almacenados en modelos de procesos diagramados con el estándar UML, por medio de Agentes de Software y servicios Web los cuales son consumidos y mostrados al usuario final por medio de una interfaz web. De esta forma se facilita la búsqueda de modelos existentes los cuales se utilizarán como base para el diseño de nuevos sistemas maximizando la reutilización del conocimiento en el desarrollo de software.

Palabras Clave: Agentes de Software, Arquitectura de Software, Modelos, Servicios Web.

Abstract

The complexity and magnitude of software development projects has increased considerably in recent years, making it the need to reuse components or knowledge gained from experience, either through design patterns, libraries, best practices of software development or business models that facilitate and enable projects to develop the best possible way.

The need for software and (or) knowledge reuse has become the reuse of components in one of the most common challenges facing software development today. Many of the approaches covered have been dedicated to the reuse of software components, plugins or design patterns, but except for design patterns, the other components are dependent on specific technologies and unable to prevail over time and technological advantages. Likewise, many organizations dedicated to software development have gained great experience and created repositories where they store architectures and models used in developing software projects, but lack the mechanisms to exploit the knowledge they have accumulated over time.

This paper proposes and implements an architecture called ARIPSAMA that allows the application of methods of prospecting and mining of concepts stored in process models diagrammed with the standard UML, through Software Agents and Web services which are consumed and displayed to the end user through a web interface. This will facilitate the search for existing models which are used as the basis for the design of new systems to maximize the reuse of knowledge in software development.

KeyWords: Software Agents, Software Architecture, Models, Web Services.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Abreviaturas

ACL	Agent Communication Language
AMS	Agent Management System
AP	Agent Platform
API	Application Program Interface
ARIPSAMA	Arquitectura para la Reutilización de Información de Procesos de Software en un Ambiente Multi-Agente
BPM	Business Process Model
CA	Communicative act
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
JSF	Java Server Faces
MVC	Model View Controller
OMG	Object Management Group
SOAP	Simple Object Access Protocol
UDDI	Universal Distribution Discovery and Interoperability
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Glosario de Términos

Application Programming Interface (API): Application programming interface (por sus siglas en ingles) consiste en un conjunto de definiciones de la forma en la que un componente de software se comunica con otro. Es un método utilizado usualmente para implementar abstracción entre un nivel bajo de programación y un nivel complejo.

Arquitectura de Software: "La Arquitectura del Software es la estructura o estructuras de un sistema, que comprende los elementos o componentes del software, las propiedades visibles de esos componentes y las relaciones entre ellos" (Gomaa, 2011).

Patrón de Diseño: Un Patrón es una regla con 3 partes: cierto contexto, un Problema y su Solución. Describe una solución probada a un problema de diseño recurrente, haciendo particular énfasis en el contexto del problema y en las consecuencias e impactos de la solución (Gomaa, 2011).

Mecanismo de Arquitectura: Representa una solución común y concreta a problemas frecuentemente encontrados. Pueden ser patrones de estructura, patrones de comportamiento o ambos. En el Proceso Unificado de Rational (RUP), los mecanismos de arquitectura son usados para agrupar los mecanismos de análisis, diseño e implementación.

Vista de Arquitectura: Una vista de la arquitectura muestra la estructura del sistema desde una perspectiva dada. Enfocándose principalmente en estructura, modularidad, componentes esenciales y los flujos de control principales.

Componente: Una parte del sistema no trivial, casi independiente y remplazable, que completa por si sola una función clara en el contexto de una arquitectura bien definida. Conforma y proporciona una realización de un conjunto de interfaces (Gomaa, 2011).

Un componente es típicamente especificado por uno o más clasificadores (ejemplo: clases de implementación) que residen en él, y pueden ser implementados por uno o más artefactos (ejemplo: binarios, ejecutables o archivos de script).

Paquete: Es un mecanismo de propósito general para organización de elementos dentro de grupos. Los paquetes pueden ser anidados dentro de otros paquetes.

Interface: Es una colección de operaciones que son usadas para especificar un servicio de una clase o un componente. Un conjunto nombrado de operaciones que caracterizan el comportamiento de un elemento (Gomaa, 2011).

Framework (Marco de Trabajo): Un framework o marco de trabajo, es una infraestructura que soporta un conjunto de conceptos, valores y prácticas que facilitan la construcción de aplicaciones. Un framework provee una serie de herramientas y componentes que permiten modelar e implementar de manera natural la realidad.

Java 2 EE: J2EE es el acrónimo de Java 2 Enterprise Edition, que traducido literalmente al español quiere decir Java 2 Edición Empresarial.

1. Introducción

La complejidad y magnitud de los proyectos de desarrollo de software se ha ido incrementando considerablemente en los últimos años, lo cual hace cada vez más evidente la necesidad de reutilizar componentes o conocimientos adquiridos en base a la experiencia, ya sea por medio de patrones de diseño, librerías, buenas prácticas del desarrollo de software o modelos del negocio, que faciliten y permitan desarrollar los proyectos de la mejor manera posible.

Para mejorar un trabajo se debe saber exactamente en qué consiste y, excepto en el caso de trabajos muy simples y cortos, rara vez se tiene la certeza de conocer todos los detalles de la tarea. Por lo tanto, se deben observar todos los detalles y registrarlos. Una forma de modelar los procesos complejos de una organización o un software, es mediante la utilización de diagramas que representen el proceso o modelo.

Los diagramas UML pueden representarse mediante archivos XMI los cuales a su vez, pueden ser manipulados con tecnologías XML, lo cual permite la aplicación de métodos de búsqueda y extracción de conceptos que permitan inferir la naturaleza funcional del modelo diagramado. De esta manera es posible obtener patrones con un comportamiento similar facilitando su almacenamiento en un esquema relacional que facilite la búsqueda y propicie la reutilización del conocimiento plasmado en los modelos.

1.1. Motivación

Herramientas tales como BPM (Business Process Modeling), SOA (Service Oriented Architecture) y workflow están tomando mayor fuerza, tanto en las organizaciones como en la industria del software. Esto ha llevado a que las organizaciones piensen en el desarrollo y la adquisición de aplicaciones de software adoptando el concepto de arquitecturas empresariales y haciendo de los procesos de negocio la plataforma para la estructuración de la organización, el planeamiento estratégico y la tecnología para el manejo de su información (Kohlbacher, Gruenwald, & Kreuzer, 2010).

Estas arquitecturas conducen al desarrollo de los procesos del negocio desde su concepción organizacional hasta su aplicación en soluciones, tales como SOA (Larentis, 2007) o flujos de trabajo (workflows) soportados por sistemas transaccionales. Para lograr esta transición, los procesos del negocio se pueden modelar por medio de diagramas de actividades/procesos UML y automatizar por medio de las tecnologías asociadas a la administración de flujos de trabajo.

Mientras los diagramas de actividades proveen los elementos de modelo que identifican las acciones que se realizan durante la ejecución de un proceso o actividad (Graig, 2003), los workflows proveen la representación de un proceso del negocio facilitando la comunicación y colaboración entre los integrantes del grupo de trabajo o coordinando la secuencia de sus actividades, dependiendo de las reglas de negocio que definan el proceso de la organización (Talib, Volz, & Jablonski, 2010).

En vista de la importancia que tiene el modelo de negocio para las diversas instituciones, es extremadamente importante el reconocimiento y reutilización del conocimiento previamente adquirido, con lo cual se reduce el tiempo de análisis y desarrollo de un proyecto determinado, así como los costos asociados al mismo (Cai, Zou, Wang, Xie, & Shao, 2011).

Es importante recalcar que en las últimas décadas, se ha propiciado un auge notable en las diversas tecnologías para el desarrollo de software, específicamente en los lenguajes de programación y ambientes de desarrollo, lo cual afecta de manera directa los componentes desarrollados como módulos reutilizables, ya que muchas veces se vuelven obsoletos o hay que realizar numerosos cambios para adaptarlos a la nueva versión del lenguaje o herramienta de desarrollo terminando con un producto de mala calidad y posibles fallas, al menos que se invierta tiempo y recursos considerables para adaptarlo a las nuevas tecnologías. Es por ello que muchas veces el componente es reconstruido en su totalidad utilizando las nuevas tecnologías, dejando de lado la migración o actualización del mismo. Partiendo de este hecho, se hace evidente la necesidad de reutilizar componentes independientes de la tecnología del momento, con lo cual se aprovecha el conocimiento y el esfuerzo aplicado en ellos. Este tipo de artefactos, son los modelos de diversas funcionalidades que pueden estar representados por medio de lenguajes de modelado tales como UML.

El principal motivo para la realización de este trabajo, es la definición e implementación de una arquitectura que facilite la búsqueda y reutilización de diagramas que representen procesos de software/negocios,

1.2. Problemática

Las empresas de desarrollo de software están en una constante búsqueda de procesos que optimicen el desarrollo de nuevas aplicaciones. En la actualidad existe un gran interés en desarrollar o usar técnicas que puedan integrar los procesos del negocio con el proceso de desarrollo y los productos de software que se construyen. Pero muchas veces el análisis de una problemática planteada se vuelve tedioso, al no tener un acceso eficiente al conocimiento previamente adquirido a través del desarrollo de proyectos similares. Debido a ello, el análisis de esta problemática tiende a realizarse casi desde cero, pudiendo haberse reducido a una simple consulta al conocimiento almacenado y recopilado durante el análisis y desarrollo de proyectos anteriores.

La gestión adecuada de un elevado volumen de información es el principal problema que es necesario solventar para que un proceso de desarrollo con miras en la reutilización, pueda convertirse en una alternativa viable en comparación a los procesos de desarrollo tradicionales. (Díaz, 2002).

1.3. Objetivos

El objetivo de este trabajo es diseñar una arquitectura que permita reutilizar el conocimiento plasmado en los diversos procesos de software o negocio que ha desarrollado una Organización. Estos procesos deben estar modelados mediante diagramas basados en UML que puedan representarse mediante estándares XML. Basados en la estructura estándar de los archivos XML, los diversos diagramas UML pueden ser procesados de forma tal que nos permita obtener información relevante sobre el modelo o la funcionalidad que representan.

Para cumplir con este objetivo, resulta necesario conocer y poder determinar la lógica plasmada en un diagrama de Actividad/Proceso, ¿qué funcionalidad está definiendo? es decir, el significado del diagrama.

Para cubrir esta necesidad esta investigación se enfocará en el mapeo de modelos UML a un esquema relacional que facilite la búsqueda y reutilización de los modelos.

1.4. Metodología

Para cumplir con los objetivos planteados en la sección 1.3 se ha establecido una metodología de trabajo adaptada de procesos de metodología de la investigación y el proceso Unificado Racional (RUP por sus siglas en inglés), que abarca los siguientes puntos:

- Identificación de la problemática basada en observaciones del proceso de desarrollo de software y la problemática de reutilización del conocimiento en un ambiente Real.
- Revisión de los principales componentes involucrados en el dominio del problema, mediante una revisión bibliográfica.
- Definición del alcance de la solución.
- Análisis de diversos trabajos realizados, cuyo enfoque brinda las bases que apoyan directa o indirectamente el objetivo de esta investigación.
- Definición de las principales tecnologías y herramientas que serán utilizadas como apoyo para la definición e implementación de la arquitectura ARIPSAMA.
- Definición conceptual de una Arquitectura para la Reutilización de Información de Procesos de Software (ARIPSAMA), aplicando técnicas de análisis y diseño orientado a objetos.
- Implementación de una herramienta que valide la arquitectura propuesta y ofrezca una interfaz amigable para la consulta de información de procesos de software/negocios a través de una aplicación web.
- Documentación del trabajo realizado.

1.5. Organización del Documento

El presente documento está dividido en 7 capítulos. El presente capítulo corresponde a una introducción de la propuesta de tesis, define la problemática, la motivación y los objetivos que llevaron al desarrollo de esta tesis.

- El capítulo dos incluye la revisión bibliográfica que define los principales conceptos aplicados en el desarrollo de este trabajo tales como, agentes de software, UML, servicios web, entre otros.

- El capítulo tres aborda los principales trabajos relacionados, tomando en consideración aquellos que realizan conversión de modelos, extracción de conocimiento de modelos, búsquedas en repositorios de software, entre otros.

- El capítulo cuatro define el alcance, las restricciones y las principales herramientas y tecnologías utilizadas para el desarrollo del proyecto.

- El capítulo cinco describe conceptualmente el trabajo propuesto, definiendo los modelos que ilustran la arquitectura ARIPSAMA, así como los patrones de diseño y mecanismos de arquitectura utilizados.

- El capítulo seis describe el caso de estudio de aplicación de la arquitectura ARIPSAMA, describiendo el proceso de implementación.

- El capítulo siete presenta las conclusiones y consideraciones finales del trabajo presentado, así como la presentación de posibles trabajos futuros.

2. Fundamentación Teórica

El presente capítulo describe los principales conceptos involucrados a lo largo del desarrollo de la investigación, representa un marco de referencia teórica necesario para el desarrollo del proyecto. Cada uno de los conceptos definidos a continuación se ven involucrados de forma directa o indirecta dentro del alcance de esta investigación.

2.1. Lenguaje Unificado de Modelado - UML

El Lenguaje Unificado de Modelado (UML) es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas de software, así como para el modelado del negocio y otros sistemas que no necesariamente sean de software (OMG, 2011).

UML se ha convertido en la notación visual estándar para el modelado orientado a objetos. Fue adoptado como estándar en el año 1997 por el OMG (Object Management Group), organización que promueve estándares para la industria (Graig, 2003).

El UML ha sido ampliamente aceptado, tanto por los fabricantes de herramientas de modelado como por desarrolladores de software, lo que le ha permitido ganar gran popularidad y ser uno de los estándares aplicados en muchas de las metodologías de desarrollo de software existentes.

El UML define un conjunto de diagramas separados para especificar gráficamente el diseño de software, de los cuales los diagramas de actividades, diagramas de clases y diagramas de componentes se describen en las siguientes secciones.

2.1.1. Diagramas de Actividades

Los diagramas de actividades muestran los aspectos dinámicos de un sistema, pueden describir procesos o casos de uso, además permiten elegir el orden en que pueden hacerse las cosas y establecer las reglas de secuencia a seguir (Booch,

Jacobson, & Rumbaugh, 1997). Estos diagramas se encuentran agrupados dentro de los diagramas UML que definen el comportamiento de un sistema.

Los diagramas de actividades son utilizados para modelar el comportamiento de los sistemas por ejemplo, los pasos lógicos que debe seguir un proceso basado en diferentes condiciones, procesamiento concurrente, acceso a datos, entre otros.

El elemento principal de un diagrama de actividad es la “actividad”, que se representa mediante un rectángulo de contorno redondeado y puede relacionarse con otras actividades a través de elementos como control de flujo, nodos de decisión, nodos de unión, entre otros. Además de las actividades, todo diagrama de actividad debe contar con un nodo inicial y un nodo final, los cuales sirven para indicar el inicio y fin de la actividad. La Figura 2.1 muestra un ejemplo de diagrama de actividad UML.

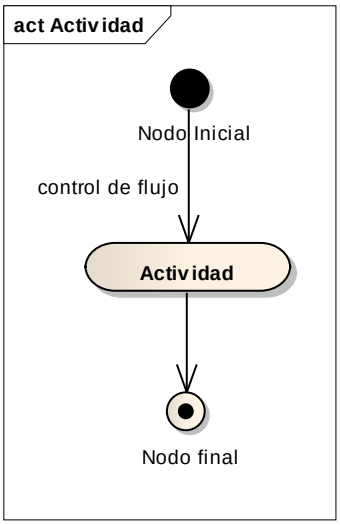


Figura 2.1: Ejemplo de diagrama de actividad UML.

2.1.2. Diagramas de Clases

Los diagramas de clases, son diagramas utilizados para estructurar los objetos, operaciones y atributos de esos objetos.

En UML, las clases son modeladas mediante un rectángulo que normalmente consta de tres compartimientos: el nombre de la clase, los atributos y los métodos (Graig, 2003). La Figura 2.2 muestra un ejemplo de diagrama de clases.

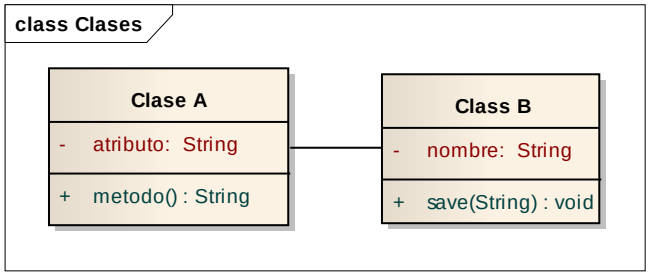


Figura 2.2: Ejemplo de Diagrama de Clases UML.

2.1.3. Diagrama de Componentes

Un componente es una parte de la estructura de un sistema que representa un empaquetamiento físico de elementos relacionados lógicamente (Cheesman & Daniels, 2001). Mientras que un diagrama de componentes captura la estructura física del software con el objetivo de organizar el código fuente y los ejecutables surgidos durante el desarrollo. El diagrama de componentes modela uno o más componentes de un sistema.

Los diagramas de componentes son utilizados para definir arquitecturas de software. En la cual se modelan las relaciones, dependencias, y composiciones de objetos. La Figura 2.3 muestra un ejemplo de diagrama de componentes.

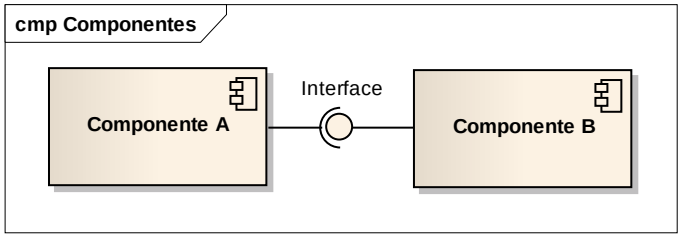


Figura 2.3: Ejemplo de Diagrama de Componentes UML.

2.2. XML - Extensible Markup Language

El Lenguaje de marcado extensible (XML por sus siglas en inglés) es un lenguaje de marcado en texto plano utilizado para la representación de información estructurada: documentos, datos, configuración, libros, transacciones, facturas, entre otros. Fue

concebido como un subconjunto derivado del Standard General Markup Language SGML (ISO 8879), con el fin de ser más adecuado para su uso Web (W3C, 2011).

XML es un lenguaje de marcado que hace un uso extensivo de etiquetas y atributos. Marcado se refiere a cualquier cosa que le proporciona o le añade información adicional a un documento.

XML consiste principalmente de dos piezas básicas: elementos y atributos. Los elementos son la unidad de contenido básico en XML. Están delimitados por etiquetas o marcas y pueden contener a otros elementos o información de caracteres.

Los atributos incorporan características o propiedades a los elementos de un documento. Los atributos se incluyen en la etiqueta de inicio de un elemento y están expresados como pares nombre-valor. Los documentos XML son procesados por un analizador sintáctico (parser), que valida cada instancia de un documento XML comparándola con una DTD. La Figura 2.4 muestra un ejemplo básico de un documento XML, en el cual los elementos **lastname**, **firstname** y **age** tienen el atributo **type** que define el tipo de dato que puede contener el elemento.

```
<?xml version="1.0" encoding="windows-1252" ?>
<people>
    <person>
        <lastname type="String">Brown</lastname>
        <firstname type="String">Jim</firstname>
        <age type="Integer">24</age>
    </person>
</people>
```

Figura 2.4: Ejemplo de archivo XML

2.2.1. XMI - XML Metadata Interchange

XMI es un estándar del Object Management Group (OMG) para el intercambio de información utilizando meta-data a través de XML. Los metadatos son información para describir la información. El objetivo del XMI es el intercambio de meta-información.

XMI no especifica un vocabulario XML, pero proporciona un algoritmo que genera vocabularios para metamodelos (IBM, 2011). En otras palabras XMI no define clases,

atributos, relaciones ni otros tags comunes de XML ó XML Schema, en contraposición XMI especifica como crear tags para conceptos en un metamodelo. Es por ello que XMI no debe verse como un vocabulario o un conjunto de tags (elementos y atributos), sino como un framework.

La especificación para el intercambio de metadatos fue escrita para proveer una manera de compartir modelos UML entre diferentes herramientas de modelado. En versiones anteriores de UML se utilizaba un Schema XML para capturar los elementos utilizados en el diagrama; pero este Schema no decía nada acerca de cómo el modelo debía graficarse por ende el intercambio de modelos entre distintas herramientas de modelado era complicado. Un ejemplo de diagrama UML en formato XMI puede verse en la Figura 2.5.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<xmi:XMI xmi:version='2.1' xmlns:xmi='http://schema.omg.org/spec/XMI/2.1' xmlns:uml
  <documentation exporter='Oracle JDeveloper' exporterVersion='11.1.2.1.0'/>
  <uml:Package xmi:id='urn:uuid:15249904-c5f9-4811-abb2-b6ec5343a88d' name='model'>
    <packagedElement xmi:type='uml:Activity' xmi:id='urn:uuid:b1f529f7-926a-4e0a-aa
      <edge xmi:type='uml:ControlFlow' xmi:id='urn:uuid:5147e10a-0b2c-4741-931a-70a
        <guard xmi:type='uml:LiteralString' xmi:id='urn:uuid:526dd65c-7260-4fde-acc
        <weight xmi:type='uml:LiteralString' xmi:id='urn:uuid:48473bed-0c77-4365-9d
      </edge>
      <edge xmi:type='uml:ControlFlow' xmi:id='urn:uuid:ccb6c25a-555d-4e31-9c31-a58
        <guard xmi:type='uml:LiteralString' xmi:id='urn:uuid:8f9c77ca-82c8-4d45-943
        <weight xmi:type='uml:LiteralString' xmi:id='urn:uuid:8ed3f83b-4575-4155-8a
      </edge>
      <node xmi:type='uml:ActivityFinalNode' xmi:id='urn:uuid:0d67d09e-74b6-42e0-81
      <node xmi:type='uml:InitialNode' xmi:id='urn:uuid:9b3ef1a6-b394-4c7b-a7f7-6c6
      <node xmi:type='uml:OpaqueAction' xmi:id='urn:uuid:1b45feac-fa30-4c88-98f9-36
    </packagedElement>
  </uml:Package>
</xmi:XMI>
```

Figura 2.5: Extracto de modelo UML en formato XMI.

Aunque el XMI puede verse como un framework, desafortunadamente esto implica que dos herramientas distintas pueden no interpretar el framework de la misma forma (IBM, 2011). Por ende existen diferencias entre distintas herramientas basadas en la misma especificación XMI, al igual que existen diferencias entre la misma herramienta que utilicen especificaciones XMI distintas. Estas diferencias pueden ser poco significativas, pero pueden causar incompatibilidad entre modelos y herramientas de modelado.

La especificación XMI define una serie de elementos comunes entre herramientas de modelado y modelos, las cuales se describen a continuación:

- **XMI:** es el elemento raíz y debe contener un atributo llamado xmi.version. Las versiones válidas son 1.0, 1.1, 1.2 y 2.0
- **XMI.header:** almacena información sobre el modelo.
- **XMI.documentation:** almacena la información de usuario final, y contiene los siguientes sub elementos: XMI.owner, XMI.contact, XMI.longDescription, XMI.shortDescription, XMI.exporter, XMI.exporterVersion, XMI.exporterID, XMI.notice.
- **XMI.metamodel:** almacena el metamodelo al cual se le ha aplicado el algoritmo XMI.
- **XMI.content:** contiene el modelo actual.
- **XMI.id y XMI.idref:** el xmi.id es el identificador del modelo, el cual debe ser único; el xmi.idref es la referencia a un elemento dada por el identificador.

2.3. Reutilización de Software

La reutilización de software fue planteada en su momento como una vía complementaria para la mejora de los procesos de desarrollo de sistemas, con los objetivos de agilizar todas las tareas propias de estos procesos e incrementar la calidad de los sistemas obtenidos (Díaz, 2002).

2.3.1. Paradigmas de Reutilización de Software

El conocimiento toma un rol cada vez más relevante para la toma de decisiones en las organizaciones. Esto demanda que el conocimiento que se genera, almacena y transfiere con el apoyo de medios tecnológicos y organizacionales, sea efectivamente reutilizado para el logro de los objetivos del negocio.

Para representar la lógica del negocio en una organización, pueden utilizarse diversas tecnologías, entre las cuales podemos mencionar los diagramas de actividades/procesos los cuales representan el comportamiento interno de una operación o de un caso de uso, bajo la forma de un desarrollo por etapas, agrupadas secuencialmente (Gomaa, 2011).

En la actualidad el software se hace cada vez más esencial para el desarrollo comercial, educativo y social. Mientras por una parte, las organizaciones tratan de controlar gastos crecientes del desarrollo de software; por otra parte, ellos deben responder rápidamente al cambio de necesidades comerciales y nuevas tecnologías y deben producir sistemas de software cada vez más complejos en un tiempo relativamente corto y más barato a fin de permanecer competitivos.

Es obvio que la reutilización de software puede reducir enormemente los gastos de desarrollo de aplicaciones de software, minimizar el tiempo de desarrollo, y reducir el riesgo de fracaso, así como mejorar la calidad y la fiabilidad de productos de software (Lorenz & Rosenan, 2011).

Las soluciones actuales para la reutilización de software pueden ser clasificadas en cuatro categorías (Rahim, 2003):

- Reutilización a nivel de diseño: conformada por la descripción de un proceso o patrón a alto nivel, sin prestar atención a la implementación. Ejemplo: patrones de diseño.

- Reutilización a nivel de herencia: comprende la propagación de características y comportamiento de un componente reusable a otro. Ejemplo: programación orientada a objetos.

- Reutilización a nivel de componentes: comprende componentes autónomos que ofrecen servicios comunes en forma de paquetes o funciones, que pueden ser invocados por otros componentes. Ejemplo: Microsoft COM, java EJBs.

- Reutilización a nivel de código: es el paradigma comúnmente utilizado por muchos desarrolladores, que consiste en copiar funcionalidades, paquetes o

clases desde una aplicación a otra. Se le conoce popularmente como “cut and paste” ó “copiar y pegar”.

En las últimas décadas se ha llevado un enfoque de reutilización basado en reutilización del software a nivel de código o componentes (Lorenz & Rosenan, 2011). Sin embargo, reutilizar código fuente puede convertirse en una tarea realmente problemática, esto se debe a la rápida evolución de las tecnologías y lenguajes para el desarrollo de software, de este modo muchos de los componentes desarrollados en el pasado, son prácticamente obsoletos en la actualidad, por lo que es muy probable que los componentes desarrollados actualmente, sean prácticamente inútiles en un futuro cercano.

Por ende debemos enfocar los esfuerzos por promover la reutilización de software en los niveles más altos del ciclo de vida del desarrollo de software. Es decir, enfocarnos en los niveles de diseño, cuanto más arriba subamos en el ciclo de vida, más valor tiene el activo a reutilizar, puesto que se ha requerido un mayor poder de abstracción para crearlo. Por consiguiente ganamos un nivel mayor de independencia de la plataforma tecnológica, lo que reduce el riesgo de producir componentes con un período de vida corto. Partiendo de este enfoque queda claro que no debemos tener restricciones en cuanto a lo que se desea reutilizar o buscar, es por ello que podemos enfocarnos en requisitos, diagramas UML, pruebas, manuales, código fuente, riesgos, planificaciones de proyectos, reglas de negocio, entre otros.

2.4. Agentes de Software

Los agentes son considerados uno de los más importantes paradigmas que pueden mejorar los métodos actuales para la conceptualización, diseño e implementación de sistemas de software y pueden ser una solución al problema de integración del software heredado (Bellifemine, Caire, & Greenwood, 2007).

Aunque no existe una definición formal y precisa de lo que es un agente, éstos son por lo general vistos como entidades inteligentes que existen dentro de cierto contexto o ambiente, y que se pueden comunicar a través de un mecanismo de comunicación, usualmente un sistema de red, utilizando protocolos de comunicación.

Una de las definiciones más aceptadas sobre los agentes de software los define como los sistemas computacionales situados en algún ambiente, capaces de realizar acciones de forma autónoma dentro de este ambiente con el fin de atender objetivos específicos (Silva, 2010).

Otra de las definiciones comúnmente utilizada define los agentes de software como componentes especiales de software que poseen autonomía y proporcionan una interfaz interoperable para un sistema arbitrario con un comportamiento similar a un agente humano trabajando para algunos clientes en cumplimiento de sus tareas y objetivos (Bellifemine, Caire, & Greenwood, 2007). Un ejemplo ilustrativo de la relación entre un agente de software y su entorno es mostrado en la Figura 2.6.

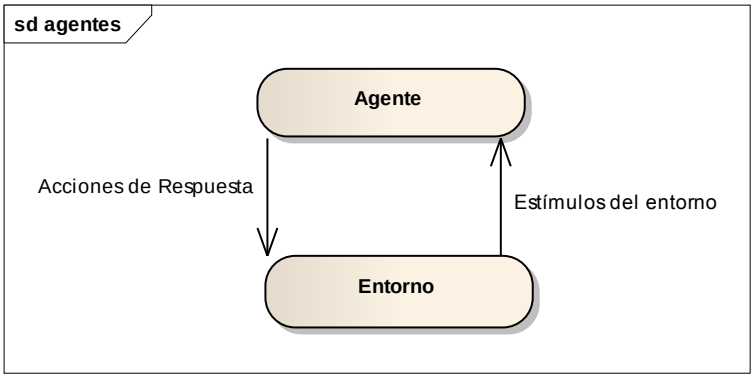


Figura 2.6: Relación Agente-Ambiente

Un agente puede ser clasificado por su rol o papel en una sociedad de agentes (SMA- Sistema Multi-Agente al que pertenece), por su especialidad o actividad en la que es experto, por sus objetivos o metas en el SMA, por su funcionalidad o lo que es capaz de hacer, por sus creencias, por su capacidad de comunicación y por su capacidad de aprendizaje (Tonconi, 2009).

2.4.1. Características

Una forma de conocer lo que constituye un agente es mediante el conocimiento de las características que debe tener una entidad de software para ser considerado un agente. Estas características (Bellifemine, Caire, & Greenwood, 2007; Dall'oglio, 2010; Martins, 2007) se describen a continuación:

- **Autonomía:** la capacidad de operar sin la intervención directa de humanos u otros agentes, teniendo el control sobre sus acciones y estados internos.

- **Sociabilidad:** capacidad de cooperar con otros agentes utilizando como medio algún lenguaje de común para la comunicación.

- **Reactividad:** el agente debe ser capaz de percibir estímulos de su entorno y responder a él en un tiempo determinado con el fin de satisfacer sus objetivos.

- **Proactividad:** no actúa simplemente en respuesta a los estímulos recibidos de su entorno, es capaz de exhibir un comportamiento dirigido por sus objetivos que le permiten mostrar iniciativa.

Las definiciones anteriormente descritas y las características listadas en esta sección tienen diversos puntos en común, en base a los cuales se propone la siguiente definición: “Un agente de software es un componente de software que exhibe autonomía, proactividad, sociabilidad y posee la capacidad de comunicación e interacción con otros agentes así como con el ambiente donde reside”

2.4.2. The Foundation for Intelligent, Physical Agents (FIPA)

FIPA fue establecida en 1996 como una asociación internacional para el desarrollo de un conjunto de estándares relacionados a la tecnología de agentes de software (Bellifemine, Caire, & Greenwood, 2007).

Los miembros iniciales fueron un conjunto de organizaciones académicas e industriales que dieron inicio a las especificaciones estándar para la tecnología de agentes de software.

Este trabajo no pretende ser una guía introductoria ni especializada de la especificación FIPA, por ende solo se describirán los conceptos más relevantes y necesarios para el desarrollo de este proyecto.

A continuación se describen los principales conceptos manejados por el core FIPA:

Agent Communication

Los agentes son fundamentalmente una forma de código de procesos distribuidos, basados en la noción clásica de la computación distribuida, compuesto de dos partes: componentes y conectores. Los componentes son los consumidores, productores, mediadores de los mensajes de comunicación intercambiados vía conectores.

La especificación FIPA define una estructura estándar para el intercambio de mensajes:

- **FIPA-ACL Message Structure Specification (SC00061):** Un mensaje FIPA-ACL contiene un conjunto de uno o más parámetros, en donde los parámetros requeridos varian de acuerdo a la situación, siendo el parámetro "performative" el único parámetro obligatorio en todo mensaje ACL. Adicional al parámetro "performative", los mensajes ACL pueden contener los parámetros "sender", "receiver" y "content". Los parámetros válidos en un mensaje ACL se resumen en la Tabla 2-1.

Parámetro	Descripción
Performative	Tipo de acción comunicativa del mensaje
Sender	Identifica al remitente del mensaje
Receiver	Identifica el receptor del mensaje
reply-to	Indica a que agente se hará una replica en respuesta a un mensaje
Content	Contenido del mensaje
Language	Lenguaje con el cual el contenido del mensaje es descrito
Encoding	Especifica la codificación usada en el contenido del mensaje
Ontology	Referencia a una ontología existente utilizada en el mensaje
Protocol	Protocolo de interacción utilizado para estructurar la conversación
conversation-id	Identificador único de la conversación
reply-with	Expresión utilizada por un agente para identificar un mensaje de respuesta

in-reply-to	Referencia a una respuesta a acción previa
reply-by	Indica la fecha y hora en que una respuesta puede ser recibida

Tabla 2-1: Parámetros de un mensaje ACL.

- **FIPA-ACL Communicative Act Library Specification (SC00037):** FIPA ACL define la comunicación en términos de función o acción, denominándola "communicative act" o "CA". Este estándar define una librería con todos los CA especificados por FIPA, un extracto de estos CA se listan en la Tabla 2-2.

FIPA communicative act	Descripción
Accept Proposal	La acción de aceptar un propuesta enviada para realizar una acción.
Agree	La acción de estar de acuerdo para realizar alguna acción
Cancel	La acción de un agente de notificar a otro agente de no realizar alguna acción
Inform	El remitente informa al receptor que la proposición dada es cierta
Not Understood	Indica que la acción a realizar no ha sido comprendida
Request	El remitente envía una petición al receptor para realizar alguna acción

Tabla 2-2: Acciones de comunicación FIPA

Agent Management

En adición a la comunicación, el segundo aspecto fundamental de un sistema de agentes según la especificación FIPA es el "Agent Management" o administración del agente. Esta administración establece el modelo de referencia lógico para la creación, registro, localización, comunicación, migración y operación de los agentes (Bellifemine, Caire, & Greenwood, 2007).

Los componentes principales de un administrador o manejador de agentes son:

- **Agent Platform (AP)**: provee la infraestructura física en la cual son desplegados los agentes de software.

- **Agente**: Un agente es un proceso computacional que habita en un AP y ofrece servicios computacionales que pueden ser publicados como una descripción de servicio.

- **Directory Facilitator (DF)**: el DF es un componente opcional de una AP que provee las páginas amarillas de servicios de agentes de software.

- **Agent Management System (AMS):** el AMS es el componente obligatorio de una AP y es el responsable de la administración y operación de la plataforma de agentes, incluyendo la creación, eliminación y migración de agentes de software desde y hacia la plataforma de Agentes.

- **Message Transport Service (MTS):** El MTS es un servicio proporcionado por una AP para transportar mensajes ACL entre agentes en una misma AP o en diferentes AP.

2.5. Servicios Web

Para que dos sistemas distintos puedan interactuar y compartir información es necesario que ambos se comuniquen por medio de una interfaz común, la cual pueda ser procesada por ambos sistemas sin la necesidad de intermediarios.

La creciente necesidad de comunicación entre distintos sistemas produjo un cambio en la forma en que los sistemas son analizados y desarrollados. Los requerimientos de software actuales procuran de sistemas interoperables entre plataformas y con la capacidad de comunicarse con sistemas existentes o con aquellos que están por desarrollarse.

La capacidad de comunicación e interacción de sistemas como Amazon, Facebook y Twitter con otros sistemas es posible gracias a la existencia de protocolos de comunicación estándar conocidos comúnmente en el mundo del desarrollo web como “Web Services” (James, 2010).

De manera más formal, se puede definir un servicio web (Del inglés Web Service) como un mecanismo o patrón de interoperabilidad entre diferentes aplicaciones de software ejecutadas en iguales o distintas plataformas (W3C, 2011).

Los servicios web son aplicaciones modulares e independientes que pueden ser descritos, publicados e invocados sobre una red, generalmente sobre internet. Son componentes interoperables, independientes del sistema operativo, independientes del lenguaje de programación con el cual son construidos y tienen la capacidad de comunicarse entre ellos (Larentis, 2007; Su, Shyang Wong, & Fen Soo, 2007).

Los servicios web combinan los mejores aspectos del desarrollo basado en componentes, permitiendo crear funcionalidades reutilizables, escalables y con bajo costo de mantenimiento, lo que los ha convertido en parte integral de desarrollo de software y como la base fundamental de la Arquitectura Orientada a Servicios (SOA).

2.5.1. Arquitectura de Servicios Web

La arquitectura de los servicios web está formada por tres participantes como se muestra en la Figura 2.7: el cliente o solicitante del servicio, el proveedor de servicio y el registro o directorio de servicios (Martins, 2007).

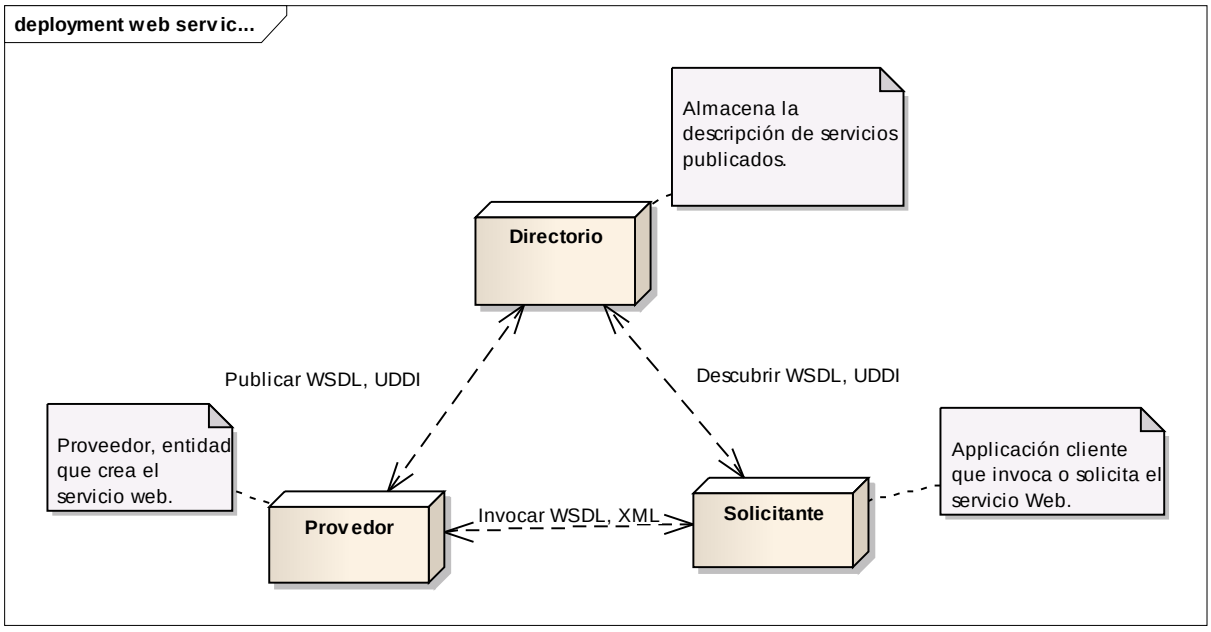


Figura 2.7: Arquitectura de Servicios Web. Adaptada de (Larentis, 2007).

El proveedor del servicio es el encargado de crear el servicio web y publicar la descripción del mismo en el directorio o registro de servicios web.

El registro o directorio de servicios web es el lugar donde los proveedores publican las descripciones de los servicios web, haciendo posible la búsqueda y descubrimiento del servicio. También almacena las instrucciones necesarias para la invocación del servicio brindando la descripción de sus operaciones, parámetros y resultados de operaciones.

El cliente o solicitante del servicio puede ser cualquier aplicación que invoca un servicio web. El cliente puede ser una aplicación web, una aplicación de escritorio, una clase sin interfaz de usuario, otro servicio web, entre otros.

La interacción descrita en Figura 2.7, muestra los componentes de la arquitectura de servicios web y la comunicación existente entre ellos. Para que esta comunicación exista de manera transparente, la arquitectura de servicios web hace uso de los siguientes estándares: SOAP (Simple Object Access Protocol), Web Services Description Language (WSDL), and Universal Description Discovery and Integration (UDDI) (Su, Shyang Wong, & Fen Soo, 2007). Cada uno de estos estándares está basado en el estándar XML descrito en la sección 2.1.3.

SOAP: Simple Object Access Protocol

SOAP es un protocolo de acceso a objetos simplificado que permite intercambiar información en un ambiente de servicios web (James, 2010). SOAP permite la comunicación entre distintas aplicaciones en ambientes distribuidos y descentralizados utilizando el intercambio de mensajes en formato XML, los cuales incluyen los parámetros requeridos y los datos del resultado.

SOAP es considerado el sucesor del protocolo XML-RPC, el cual es utilizado para el intercambio de mensajes entres servicios web, y ofrece mayor funcionalidad permitiendo la construcción de los servicios web basados completamente en SOAP.

WSDL: Web Services Description Language

WSDL es un lenguaje utilizado para describir los distintos métodos o funciones que están disponibles en un servicio web, así como su estructura, es decir, el número de

argumentos o parámetros que se les debe pasar, y el tipo de dato que devolverá la función como resultado.

WSDL describe un servicio como una colección de operaciones que pueden ser accesadas a través de mensajes. Para cada servicio web puede existir un archivo WSDL escrito en XML cuya función es describir las operaciones que el servicio web realiza (Larentis, 2007).

UDDI: Universal Description Discovery and Integration

La especificación UDDI tiene como objetivo crear un patrón para el descubrimiento de servicios web, tal como su nombre lo indica, UDDI constituye una especificación técnica para describir, descubrir e integrar servicios web.

El UDDI está constituido por dos elementos principales (Larentis, 2007; Martins, 2007):

- UDDI Project: es una especificación técnica utilizada para construir y distribuir servicios web, el cual permite que la información del servicio sea almacenada en un formato XML específico.

- UDDI Business Registry: que consiste en una implementación operacional completa de la especificación UDDI.

3. Trabajos Relacionados

Este capítulo muestra un resumen de los principales trabajos relacionados con el objeto de estudio propuesto en esta investigación. Los trabajos descritos están directamente relacionados con los objetivos propuestos en esta investigación, y abarcan diversas temáticas como la conversión de modelos, reutilización de software, búsqueda automática de modelos, entre otros.

El objetivo de este capítulo es presentar diversos trabajos que aportan contribuciones y soluciones a algunos de los problemas planteados en esta investigación. De igual forma son de utilidad para proporcionar una perspectiva comparativa del presente trabajo en relación a otros trabajos en áreas similares.

3.1. Conversión de Modelos

(Routledge, Bird, & Goodchild, 2002) Proponen un método para el diseño de esquemas XML usando el Lenguaje Unificado de Modelado (UML). El UML fue escogido principalmente porque su uso es generalizado y creciente. En segundo lugar, el UML es extensible, por lo que la nueva notación escrita es totalmente compatible con las herramientas UML existentes.

Se propuso una Arquitectura de tres niveles, modelo conceptual, modelo lógico y modelo físico, debido a que es la metodología seguida por muchos modeladores de datos. Este enfoque permite al modelador de datos centrarse en cuestiones de dominio conceptual de modelos en lugar de cuestiones de aplicación. Debido a que cada modelo de nivel conceptual tiene muchos posibles modelos a nivel lógico, es necesaria la aplicación de un algoritmo de asignación o mapeo, que utiliza técnicas de diseño de datos sensibles para traducir de una a otra.

Para realizar su cometido Routledge, Bird y Goodchild plantearon una metodología de desarrollo basada en cuatro pasos fundamentales que son: generar definiciones de tipos, determinar las agrupaciones de clase, construir el anidamiento de tipos complejos y la creación del elemento raíz. Al finalizar esta secuencia de pasos, se obtiene un XML-Schema que representa el diagrama de clases UML que ha sido transformado.

Narayanan y Ramaswamy se centran en las aplicaciones XML que utilizan una estructura estándar especificada por sus esquemas correspondientes. La salida de la actividad de diseño para tales aplicaciones es típicamente, un modelo genérico que representa el esquema de una manera abstracta. UML es el lenguaje de elección para el modelado de sistemas de software. A menudo se utiliza en el modelado de aplicaciones XML. Sin embargo, el mapeo de modelos UML a los esquemas XML está lejos de ser perfecto (Narayanan & Ramaswamy, 2005).

Narayanan y Ramaswamy proponen una serie de especificaciones estándar para cada elemento del XML Schema. Para ello listan especificaciones para las clases, atributos, asociaciones, tipos de datos, composiciones e incluso las notas de comentario. El trabajo consiste mayormente en definir un conjunto equivalente en el esquema XML, para cada elemento UML. La Tabla 3-1 muestra la especificación de mapeo propuesta para clases UML.

UML entity to be mapped	Class
Mapped XML entity	XML element and a matching Complex Type declaration
Exceptions	None
Extensions	If a class has an <<abstract>> stereotype associated with it, or is italicized, it's abstract attribute is set to true.
Assumptions	None
Comments	None

Tabla 3-1: Especificación de mapeo de clases UML

Otro trabajo que afronta la problemática de conversión UML a XMLSchema es el presentado por Mun-Young, Lim y Kyung-Soo donde proponen un generador de XML-Schemas basado en diagramas de clases UML. Para ello se definen una serie de 10 reglas importantes, previas a la conversión del modelo. Entre estas reglas podemos mencionar las siguientes: no se permiten elementos o atributos vacíos, restringir el tamaño de los campos "string" y "decimal", incluir claves que indiquen la unicidad de los campos y el modelo y añadir bloques por defecto, tanto al inicio como al final. Estas reglas fueron usadas como base para la creación de una aplicación en Java que genera XML-Schemas a partir de diagramas de clases UML (Mun-Young, Lim, & Kyung-Soo, 2005).

3.2. Recomendación de Componentes Reutilizables.

Este trabajo fue concebido para brindar una solución a un problema común en la reutilización de software, el cual consiste en elegir la mejor opción de un conjunto de componentes con características similares que resuelven una problemática en común.

Cai, Zou y otros; parten de la concepción de una organización estructurada para manejar componentes de software reutilizables, que cuenta con un repositorio de componentes. En la cual el proceso de recuperación de los componentes alojados en el repositorio, presenta el problema de elección de componentes de los candidatos que ofrecen funcionalidades similares.

Para abordar el problema, este trabajo propone un método semi-supervisado para recomendar los componentes del repositorio a los desarrolladores (Cai, Zou, Wang, Xie, & Shao, 2011). El enfoque propuesto calcula la probabilidad de recomendación de los componentes para identificar los componentes recomendables en función de sus citas en Internet.

La arquitectura propuesta plantea tres partes funcionales: el recuperador, el rastreador y el motor de recomendación. Tal como se muestra en la Figura 3.1.

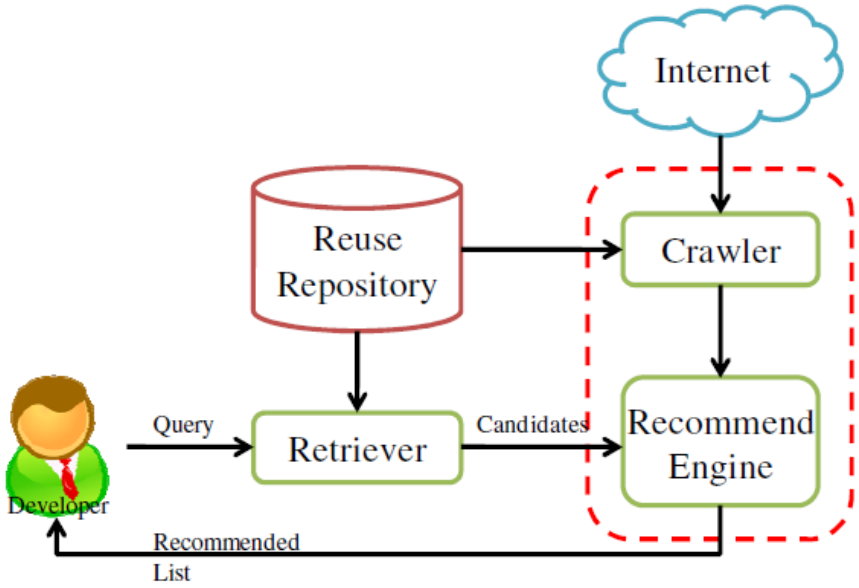


Figura 3.1: Arquitectura de Recomendación de Componentes

El recuperador obtiene los componentes candidatos relevantes desde el repositorio de reutilización de acuerdo con la consulta enviada por los desarrolladores. El rastreador obtiene los hosts que involucran a los componentes en el repositorio y crea las asociaciones entre los componentes y los hosts. El motor de recomendaciones calcula probabilidad de recomendación para cada componente utilizando las asociaciones generadas por el rastreador, y produce los componentes recomendables sobre la base de los candidatos recuperados y los devuelve como resultados de la consulta.

Cada componente tiene una probabilidad recomendación (llamado "probabilidad de recomendación de componente"), lo que significa el grado que se recomienda el componente. También establece un peso (llamado "probabilidad de recomendación del host") a cada host (Cai, Zou, Wang, Xie, & Shao, 2011).

La evaluación de los métodos propuestos se realizó en base a los datos recabados del repositorio SourceForge. Se seleccionó la categoría "Desarrollo de Software", como base de evaluación. Y se procedió a realizar consultas con el motor de SourceForge y el motor propuesto por los autores.

La Tabla 3-2 muestra un resumen de los resultados obtenidos en el proceso de evaluación, la columna query corresponde al criterio de búsqueda ingresado, la columna retrieved al número de resultados obtenidos y la columna recommended al número de recomendaciones realizadas. Las recomendaciones indicadas por SourceForge se encuentran dentro del paréntesis.

Query	Recommended	Retrieved
XML Parser	3(5)	14
Data Encryption	1(1)	7
Logging	9(11)	12
Math	5(5)	12
Statistics	5(5)	7
Data Compression	1(1)	10
Email	1(1)	6
File Upload	2(4)	14

Configuration File	3(3)	11
Network Utility	1(1)	11
IO Utility	2(2)	14

Tabla 3-2: Recomendaciones de SourceForge vs Método Basado en citas.

3.3. Búsqueda de Diagramas de Secuencia Reutilizables

El trabajo presentado por William N. Robinson and Han G. Woo, “Finding Reusable UML Sequence Diagrams Automatically” tiene como objetivo proponer un método de búsqueda que ofrezca el modelo más adecuado para solventar una problemática dado un modelo parcial de la solución deseada.

Robinson y Woo proponen una serie de técnicas para la reutilización de artefactos UML, específicamente los diagramas de secuencia. El punto más importantes de su propuesta se basa en encontrar el modelo que mejor se adapte a las características o funcionalidades deseadas, lo que ellos plantean como el primer y más importante paso para la reutilización de modelos (Robinson & Woo, 2004). Entre los criterios de búsqueda planteados para cumplir con este objetivo están los siguientes:

- Búsqueda por comparación de Conceptos: este método es la manera más común de encontrar el mejor modelo de acuerdo a los criterios buscados. Se basa en la comparación de un criterio deseado con el conjunto de conceptos almacenados en el modelo, ya sea por atributos propios del modelo o por anotaciones de documentación agregadas para explicar el mismo.

- Búsqueda por relaciones: esta perspectiva extiende la búsqueda de conceptos para incluir la búsqueda por relaciones entre conceptos. Se basa en la representación de un modelo como un grafo en el cual se comparan conceptos, relaciones, estructura de vértices, entre otros. Para este método cada concepto corresponde a un nodo o vértice de un grafo y cada relación equivale a una relación entre vértices.

Para realizar la búsqueda automática se hace uso del algoritmo de Clasificación de **SUBDUE** ver Figura 3.2. El cual recibe como entrada el criterio de consulta en forma de grafo y lo compara con la representación en grafo de los modelos existentes en la librería o repositorio. Al menos una etiqueta entre el modelo buscado y los modelos almacenados debe coincidir.

```
Subdue ( graph G, int Beam, int Limit )
  queue Q = { v | v has a unique label in G }
  bestSub = first substructure in Q
  repeat
    newQ = {}
    for each S ∈ Q
      newSubs = S extended by an adjacent edge from G in all
        possible ways
      newQ = newQ » newSubs
      Limit = Limit - 1
    evaluate substructures in newQ by compression of G
    Q = first Beam substructures in newQ in decreasing order
        of value
    if best substructure in Q better than bestSub
    then bestSub = first substructure in Q
  until Q is empty or Limit ≤ 0
return bestSub
```

Figura 3.2: Algoritmo de Clasificación de Subdue

Para corroborar su propuesta Robinson y Woo proponen la creación de una herramienta denominada **Reuse** la cual permite realizar búsquedas en los modelos, planteando la siguiente secuencia de actividades:

- Un analista define o selecciona una colección de artefactos.
- La herramienta calcula las ponderaciones o puntuaciones de la estructura de los artefactos en la biblioteca.
- Para consultar la biblioteca, un analista proporciona una estructura parcialmente terminada. (La herramienta calcula la puntuación de la estructura y de la estructura parcial, busca en la colección de artefactos, y devuelve el artefacto que mejor se adapte.)

- El analista adapta el artefacto recuperado en el contexto del problema actual.

La combinación de la búsqueda conceptual y la búsqueda por relaciones según los autores se obtiene el modelo mejor adaptado a las características deseadas.

3.4. Reutilización de Especificaciones UML.

Este artículo describe un método de reutilización de modelos de software diseñados en UML en un dominio restringido. Específicamente los diagramas de caso de Uso. La principal ventaja del método es la posibilidad de reutilización del software en las primeras etapas del ciclo de vida del desarrollo, la especificación de casos de uso y sus flujos de eventos.

Block y Cybulski brindan detalles de las técnicas de representación utilizado para almacenar y recuperar el diseño reutilizable de una gran colección de especificaciones UML. En él se describen los enfoques para reducir la complejidad de tratar con los modelos de dominio muy grande y se describe el método de evaluación de la similitud conceptual entre los flujos de eventos y casos de uso. Para ellos, una herramienta que permita buscar el artefacto requerido debe ser capaz de:

- Representar y almacenar las propiedades esenciales de todos los modelos de un dominio.
- Formular la consulta del modelo sobre el dominio.
- Buscar el modelo que concuerde con los criterios introducidos.
- Ordenar los resultados de acuerdo al grado de proximidad de los artefactos con el criterio de búsqueda introducido.
- Devolver los artefactos seleccionados.

Para cumplir con los requerimientos previamente listados Blok y Cybulski proponen dos métodos para describir las propiedades de los objetos UML, el primer método se basa en la sintaxis y el significado léxico del dominio del modelo, el segundo método se basa se basa en la inferencia semántica de la estructura del propio modelo.

Las tres metas evaluadas al momento de ofrecer un resultado son: la afinidad del modelo al criterio buscado, la distancia semántica existente y la similitud entre la consulta y el resultado. Utilizando el método léxico, pueden evaluar la afinidad del

modelo, haciendo una comparación léxica de atributos del modelo con el criterio de búsqueda. El método semántico les permite calcular la distancia semántica. Y la combinación de ambos métodos les permite obtener la similitud del modelo buscado. Una representación del método semántico se observa en la Figura 3.3.

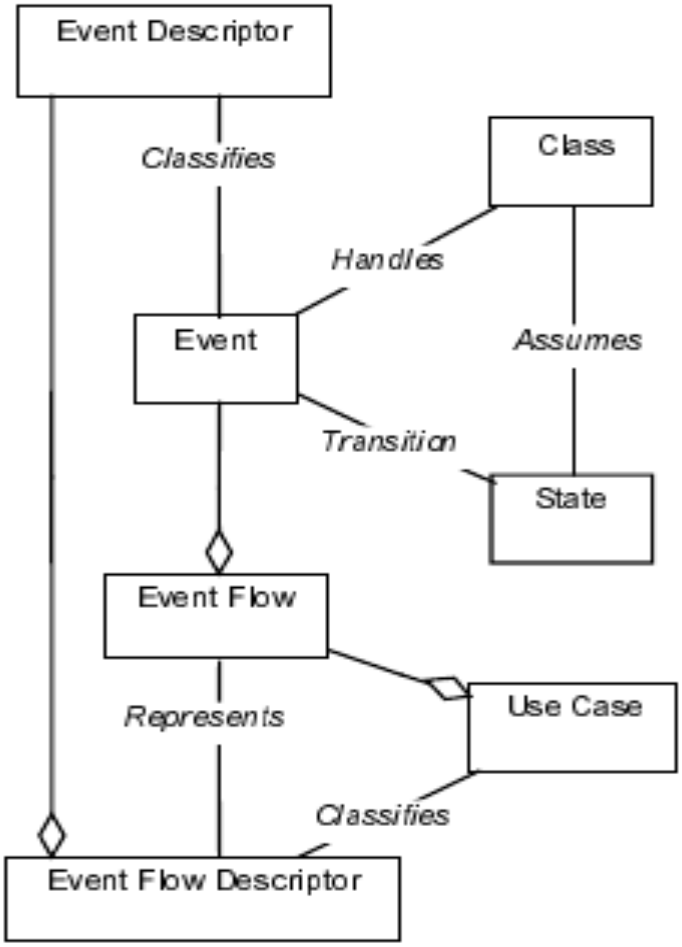


Figura 3.3: Clasificación Semántica de Modelos UML

Para manejar las comparaciones entre modelos, se propone la representación de cada modelo como un flujo de eventos. De esta forma la similitud de modelos de caso de uso se basa enteramente en la similitud de estos flujos (Blok & Cybulski, 1998).

La Figura 3.4 muestra un resumen del proceso propuesto,

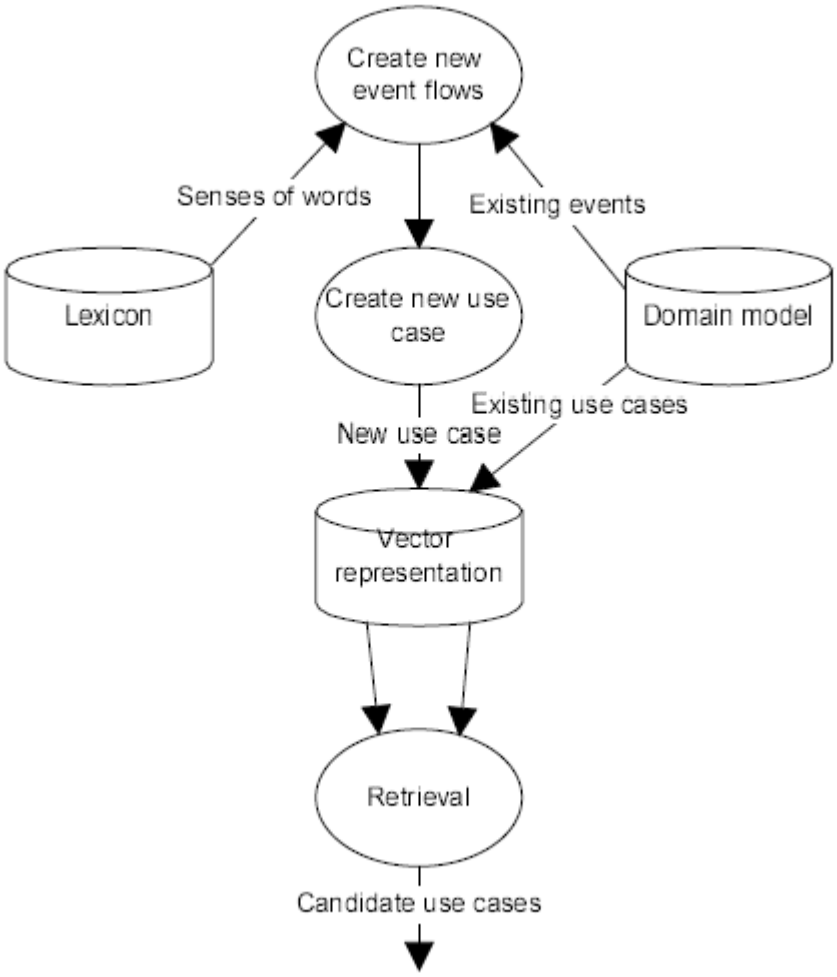


Figura 3.4: Vista del Proceso de Búsqueda

4. Alcance y Tecnologías Utilizadas

El presente capítulo brinda una introducción al trabajo propuesto, delimitando el alcance de la investigación, identificando las restricciones o limitaciones y brindado una breve descripción de las herramientas y tecnologías utilizadas en el desarrollo del proyecto.

4.1. Alcance

El alcance de este trabajo comprende la definición de la arquitectura para la Reutilización de Información de Procesos de Software en un Ambiente Multi-Agente (ARIPSAMA) y una herramienta de apoyo utilizando la tecnología Java 2 EE en conjunto con los frameworks Jade, Hibernate, JSF y la tecnología de servicios web.

La meta de la arquitectura ARIPSAMA es proveer un marco o modelo conceptual y tecnológico que permita resolver los requerimientos funcionales y no funcionales necesarios para solventar la problemática planteada en el capítulo 1, a la vez que funcione como una guía y fundamento para el proceso de desarrollo. En forma simplificada se puede decir que son los “planos” del sistema a desarrollar.

Para comprobar la validez de la arquitectura ARIPSAMA se realizará la creación de una aplicación web basada en agentes de software y servicios web con el fin de monitorear, analizar, y clasificar las diversas lógicas planteadas en artefactos UML. Para ello, se presenta una arquitectura basada en agentes de monitoreo, análisis y clasificación de modelos, que permitan la aplicación de meta-datos semánticos sobre un modelo UML dado, con el objetivo de obtener los conceptos fundamentales que determinen la naturaleza del modelo diagramado, los cuales serán catalogados y almacenados como base de conocimiento para posteriores consultas que propicien la reutilización de los modelos catalogados.

4.2. Tecnologías y Herramientas Utilizadas

Para el diseño e implementación de la arquitectura ARIPSAMA, se han empleado un conjunto de tecnologías y herramientas de código libre o libre de costo, que ofrecen

las mejore facilidades tanto para el modelado conceptual como la implementación. Las principales herramientas y tecnologías utilizadas a lo largo de eta investigación se describen a continuación.

4.2.1. Java EE 6

Java EE, formalmente conocido como J2EE, es una plataforma que define un conjunto de estándares para el desarrollo de aplicaciones empresariales. La tecnología J2EE incluye tecnologías como servlets, Java Server Pages (JSP), Java Server Faces (JSF), Enterprise Java Beans (EJBs), Java Messaging Services (JMS), Java Persistent Api (JPA), Java Api for XML Web Services (JAX-WS) y Java Api for RESTful Web Services (JAX-RS) (Heffelfinger, 2010).

Uno de los principales frameworks a utilizar en este proyecto es JSF, el cual es un framework web MVC que simplifica la creación de interfaces de usuario para aplicaciones Java web, es el framework estándar para componentes de interfaz de usuario java.

Para mejorar el diseño web de la aplicación, se utilizará la implementación JSF de Apache MyFaces, la cual reúne un conjunto de componentes enriquecidos para el desarrollo de interfaces de usuario, con características enfocadas al reúso de software. (Thomas, 2009).

4.2.2. Oracle JDeveloper 11.1.1.5

Oracle JDeveloper es un entorno de desarrollo integrado gratuito que simplifica el desarrollo de aplicaciones basadas en SOA, Java 2 EE y las interfaces de usuario con soporte para el ciclo de vida de desarrollo completo.

JDeveloper ofrece diversas funcionalidades que facilitan y agilizan el desarrollo de aplicaciones java, entre ellas: apoyo en le creación de código, estructuración de proyectos que facilita la organización del código fuente, integración con servidores de control de versiones, integración con servidores web y frameworks Open Source como Apache Ant y Maven (Mills, Koletzke, & Roy-Faderman, 2010).

4.2.3. Java Agent Development Framework (JADE)

Es un framework que facilita el desarrollo de aplicaciones multi-agente cumpliendo con las especificaciones FIPA. JADE puede ser considerado como un middleware que implementa una plataforma de agentes eficiente y apoya el desarrollo de sistemas multiagente (Bellifemine, Caire, & Greenwood, 2007).

La plataforma de agentes JADE trata de mantener un sistema de agentes distribuidos de alto rendimiento desarrollado con el lenguaje Java. En particular, su arquitectura de comunicación trata de ofrecer mensajería flexible y eficiente, de forma transparente para elegir el mejor transporte disponible, aprovechando el estado del arte de la incrustación de objetos distribuidos en el entorno de ejecución Java.

JADE utiliza un modelo de agentes, además de la aplicación Java, el cual permite la eficiencia en tiempo de ejecución, la reutilización del software, la movilidad del agente y la realización de diferentes arquitecturas de agentes.

4.2.4. Glassfish v3.1

GlassFish Server 3.1 es el sucesor de las versiones anteriores 3.0.x de GlassFish creado bajo una arquitectura flexible y modular basada en el estándar OSGi, el cual incorpora plenamente las funciones de clustering con administración centralizada de múltiples clusters y la alta disponibilidad de los componentes (Heffelfinger, 2010).

GlassFish Server 3.1 es también el más rápido servidor de aplicaciones de código abierto y ofrece funciones avanzadas de servidor, tales como versiones de las aplicaciones, recursos de ámbito de aplicación, y un gran apoyo al desarrollo con las herramientas de NetBeans 7.0, Eclipse y otros entornos de desarrollo populares.

4.2.5. MySQL 5.5

MySQL es un servidor de base de datos rápido, multi-hilo, multi-usuario y ofrece un SQL (Structured Query Language) robusto. El servidor MySQL está diseñado para misiones críticas, sistemas de producción de alta carga, así como para integrarse de forma embebida en software para ser distribuido (Widenius & Axmark , 2011).

MySQL tiene una doble licencia. Los usuarios pueden optar por utilizar el software MySQL como un producto de código abierto bajo los términos de la Licencia Pública General de GNU o puede comprar una licencia comercial estándar de Oracle.

5. ARIPSAMA: Mecanismos de Arquitectura.

En capítulos anteriores se han presentado aspectos relacionados a UML, XML, XMI, agentes de software, y servicios web. Cada uno de estos aspectos está relacionado de manera directa o indirecta con los objetivos propuestos en este trabajo. Se ha definido el alcance del proyecto, así como las diversas tecnologías y herramientas necesarias para el desarrollo del mismo.

Este capítulo define la forma en que se desarrollará la arquitectura, y presenta los principales mecanismos de arquitectura necesarios para la implementación de la misma. En capítulos posteriores se detallará el proceso de implementación de la arquitectura ARIPSAMA, la cual hace uso de cada uno de los mecanismos definidos en este capítulo.

5.1. Representación de la Arquitectura

Este capítulo se limita a la descripción formal de los mecanismos de arquitecturas utilizados para el desarrollo de la arquitectura ARIPSAMA, los aspectos concernientes al desarrollo de la propia arquitectura serán presentados en el capítulo 6, siguiendo los lineamientos de representación de la arquitectura definidos en este capítulo.

La arquitectura propuesta se presenta a través del “Modelo de 4+1 Vistas” (Krutchten, 1996) que utiliza cinco vistas concurrentes, cada una de las cuales direcciona un conjunto específico de aspectos diferentes de los sistemas.

A continuación se describe los aspectos que abarca cada vista:

- Vista lógica: describe el diseño del modelo de objetos. Esta vista de la arquitectura dirige los requerimientos funcionales de los sistemas, lo que los sistemas deben hacer para el usuario final. Da una abstracción del modelo de diseño e identifica los paquetes, subsistemas y clases.

- Vista de Implementación: Esta vista describe la organización estática de los módulos de software (código fuente, componentes, ejecutables) en el ambiente de desarrollo en términos de paquetes, capas y componentes.

- Vista de Procesos: abarca aspectos de concurrencia de los sistemas en tiempo de ejecución – muestra las tareas, hilos, o procesos así como sus interacciones. Ilustra los puntos de concurrencia y paralelismo del sistema.

- Vista de Producción: Muestra como varios ejecutables y otros componentes son distribuidos físicamente en los servidores y estaciones de trabajo. Refleja las decisiones que afectarán la puesta en producción, la instalación y el rendimiento.

- Vista de Casos de Uso: Aunque forma parte del Modelo de 4+1 Vistas, esta sección no se tomará en cuenta ya que este documento de arquitectura no se refiere a un sistema en particular.

5.2. Mecanismos de Arquitectura

A continuación se listan los mecanismos de arquitectura:

Nombre (mecanismo de análisis)	Descripción	Solución Propuesta (Mecanismo de diseño e implementación)
Estructura de Caso de Uso	Define la estructura y comportamiento a nivel de capas que tendrán las aplicaciones, y las decisiones que deberán ser respetadas para la construcción de las funcionalidades que exponen los casos de uso.	Este mecanismo se documenta en las siguientes secciones
Acceso a Datos	Define la estrategia y las decisiones técnicas para proveer el acceso a una base de datos relacional, desde una	Este mecanismo se documenta en las siguientes secciones

	aplicación orientada a objetos	
Seguridad	Define los procesos de autenticación y autorización necesarios para la seguridad de la información manejada por el sistema.	Este mecanismo se documenta en las siguientes secciones
Comunicación.	Define el proceso de comunicación entre los agentes de software y la aplicación web.	Este mecanismo se documenta en las siguientes secciones.
Excepciones	Determina como se manejarán las excepciones y como viajarán las mismas a través de las capas de la aplicación.	Este mecanismo se documenta en las siguientes secciones

Tabla 5-1: Mecanismos de Arquitectura ARIPSAMA

5.2.1. Mecanismo de Estructura de Casos de Uso

Problema

En la ingeniería de software se denomina aplicación web a aquellas aplicaciones que residen en un servidor web y que pueden accederse a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación de software que se codifica en un lenguaje soportado por los navegadores web. Esta codificación a la que se hace mención puede realizarse de distintas maneras, y es precisamente esta variedad y la falta de estándares lo que provoca que las diversas funcionalidades requeridas en una aplicación particular no sean implementadas en forma correcta, sean estructuradas de forma ad-hoc, y den libertades a los desarrolladores de escribir "Spaghetti Code", dando como resultado aplicaciones poco escalables, difíciles de mantener y de integrar con otros sistemas.

Solución

Estandarización de la estructura de las aplicaciones web, mediante el establecimiento patrones bien definidos para el manejo de las capas de la aplicación.

El presente mecanismo hace uso del patrón de diseño Modelo –Vista/Controlador, el cual define un mínimo de tres capas para la estructuración de las aplicaciones web separando la parte lógica de una aplicación de su presentación. Básicamente sirve para separar el lenguaje de programación del HTML lo máximo posible y para poder reutilizar componentes fácilmente (Mahmoud & Maamar, 2006).

El Modelo representa las estructuras de datos. Típicamente el modelo de clases contendrá funciones para consultar, insertar y actualizar información de la base de datos. La Vista es la información presentada al usuario. Una vista puede ser una página web o un fragmento de una página web. El Controlador actúa como intermediario entre el Modelo, la Vista y cualquier otro recurso necesario (Leff & Rayfield, 2001).

La presente arquitectura no se limita a la estructuración del proyecto en tres capas, más bien utiliza el esquema propuesto por el patrón MVC, para definir una serie de capas especializadas que agrupan las funcionalidades requeridas de acuerdo a su naturaleza.

De esta forma la capa de presentación (Vista), contiene las páginas (JSF), la capa controlador contiene el intermediario (Backing Bean) entre la vista y el modelo; el modelo contiene las entidades, las consultas y los servicios que proveen acceso a la data requerida por la aplicación.

Bajo este contexto se establece el uso de las siguientes tecnologías para las aplicaciones a desarrollarse: plataforma J2EE, JSF. La siguiente sección resume las clases participantes en el mecanismo de estructuración de casos de uso.

Clases Participantes

Clase	Descripción
Pag<CasoUso>	Página .JSPX mostrada por el navegador al usuario, es la página con la cual interactúa para realizar las diferentes peticiones al sistema. Esta página .JSPX se encuentra a nivel del “Sitio Web (Web Site) o Presentación”.
Pag<CasoUso>(Backing Bean)	Clase de tipo “.java” utilizada para realizar las peticiones de los eventos de la página .JSPX al servidor. Esta clase se encuentra a nivel del “Sitio Web (Web Site) y representa la capa del controlador”.
ApplicationService<Service>	Los ApplicationService exponen los servicios de negocio, y proporcionan acceso a los datos. Conforman la interfaz de comunicación entre el mecanismo de análisis y los demás componentes de la arquitectura.

Tabla 5-2: Clases Participantes Mecanismo C.U

Diagrama de Clases Participantes

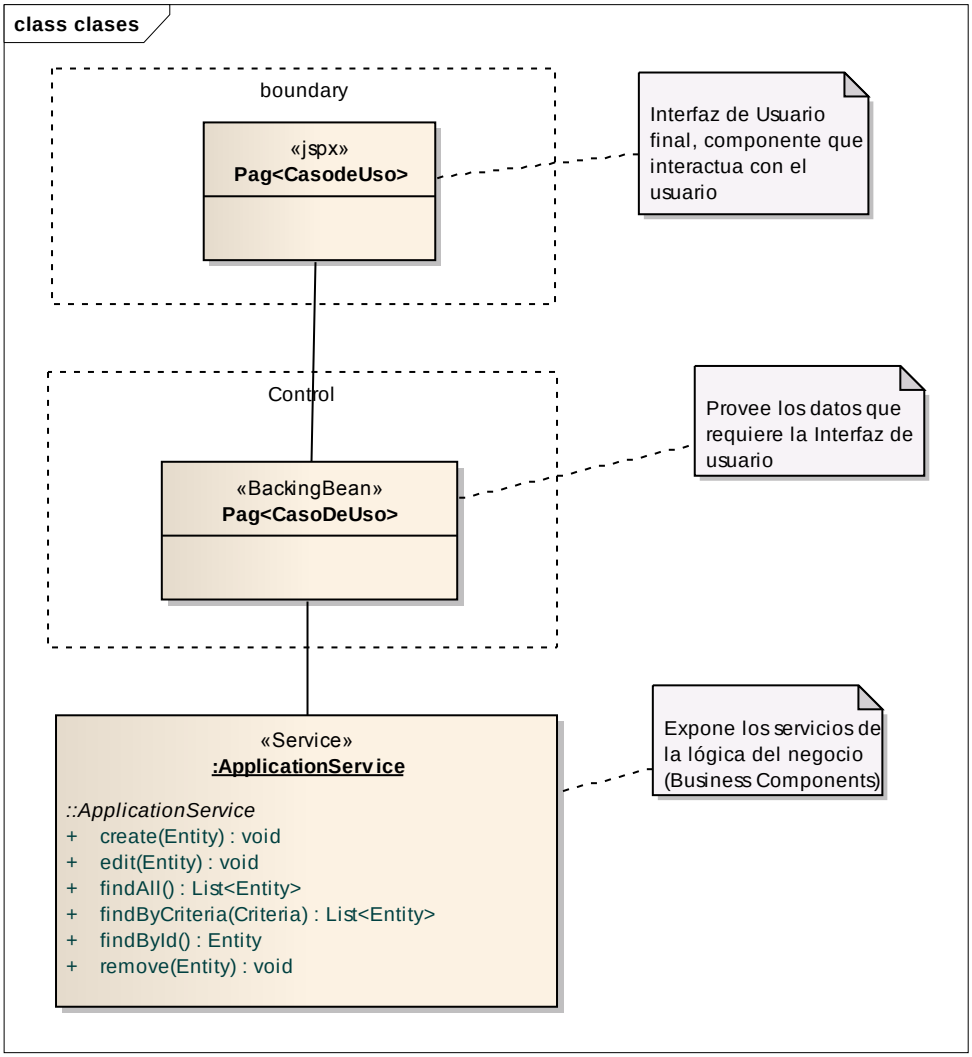


Figura 5.1: Vista de Clases Participantes Mecanismo C.U

La Figura 5.1 y Figura 5.2 muestran claramente las separaciones entre capas que maneja la aplicación, la capa de presentación es la única que interactúa de forma directa con el usuario final, las peticiones de usuario en la capa de presentación son manejadas por el controlador y de requerir acceso a datos son enviadas al application services que es la interfaz de comunicación entre el modelo y el controlador.

Diagrama de Secuencia (Procesar Petición de Usuario)

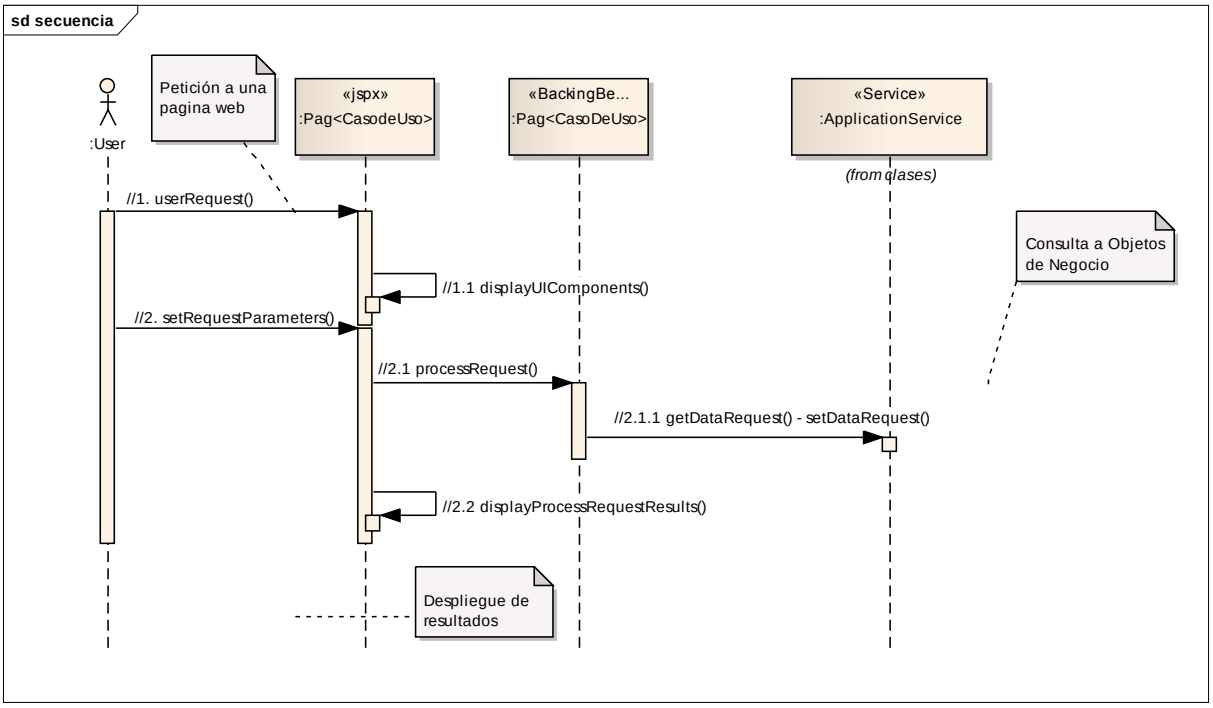


Figura 5.2: Secuencia Procesar Petición de Usuario

5.2.2. Mecanismo de Acceso a Datos

Problema

Las bases de datos relacionales no soportan la orientación a objetos, por lo que, al desarrollar una aplicación orientada a objetos, que necesita almacenar información, se requiere de algún mecanismo para conectar los objetos de negocio con la base de datos.

Solución

Este mecanismo define las decisiones técnicas para proveer el acceso a una base de datos relacional, desde una aplicación orientada a objetos.

El acceso a datos está diseñado bajo la arquitectura provista por el framework Hibernate, mediante el cual las tablas de base datos tienen una clase equivalente a nivel de aplicación denominada entidad. La entidad solamente define el mapeo clase-tabla, más no define la forma de persistir los datos, para ello se utiliza el patrón

Fachada (Facade), el cual es un patrón que permite crear una interfaz homogénea para un conjunto de clases que deben realizar tareas similares. La siguiente sección resume las clases participantes en el mecanismo, en el cual solo se representan los objetos relevantes a la aplicación, los demás componentes manejados por el framework Hibernate son omitidos para reducir la complejidad del modelo (Para mayor información, ver referencias bibliográficas. Sec. 12).

Clases Participantes

Clase	Descripción
Service	Facade especializado por entidad, que define las operaciones especializadas de una entidad.
AbstractHibernateFacade<Entity>	Clase que aplica el patrón facade y define las operaciones comunes a todas las entidades.
hibernate.cfg.xml	Archivo de configuración que relaciona las entidades con los archivos de mapeo y define las credenciales de acceso a datos.
Entity	Los objetos de entidad corresponden directamente con una tabla de la base de datos, y contendrán un atributo para cada uno de los campos de ésta. Adicionalmente, pueden definirse atributos no persistentes cuando sea necesario.
EntityMap	Archivo XML de mapeo de tablas a clases java.

Tabla 5-3: Clases Participantes-Mecanismo Acceso a Datos

Diagrama de Clases Participantes

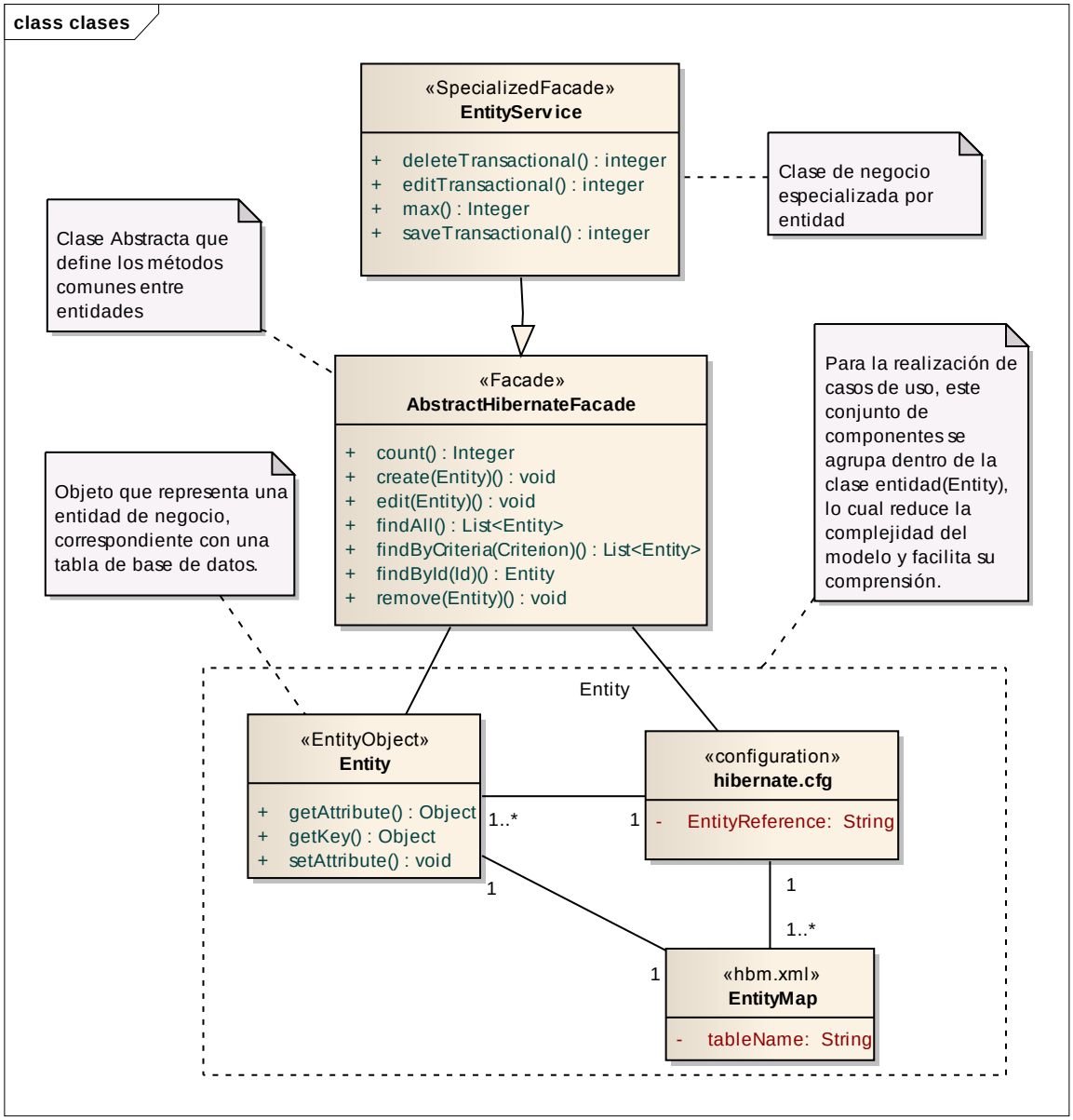


Figura 5.3: Vista Clases Participantes Mecanismo de Acceso a Datos

Diagrama de Secuencia (Flujo guardar entidad)

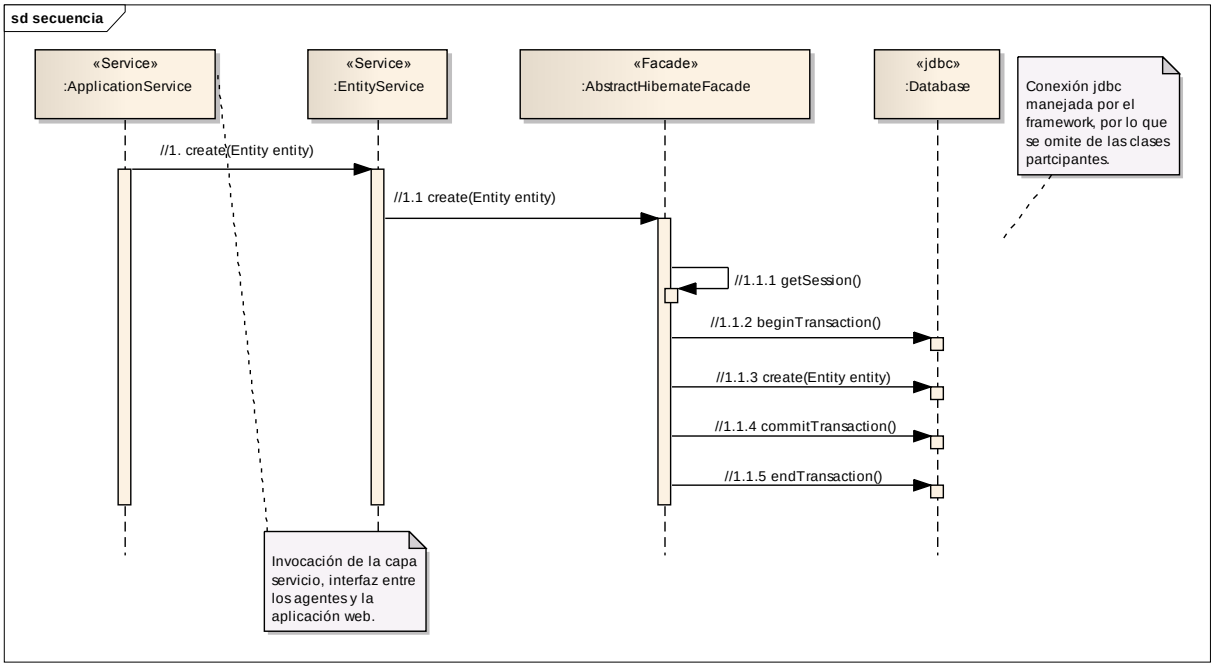


Figura 5.4: Secuencia persistir entidad.

5.2.3. Mecanismo de Seguridad

Problema

La seguridad web se ha convertido en uno de los principales retos que afronta la comunidad informática actual. La información y los datos que se manejan en una organización deben ser confidenciales y deben estar restringidos a todas aquellas personas ajenas a la institución, dueña de estos recursos; es por ello que se hace evidente la necesidad de implementar tecnologías y aplicaciones que nos permitan facilitar el acceso seguro a los recursos y servicios ofrecidos.

Solución

La aplicación de dos componentes básicos de la seguridad de la información: autenticación y autorización. La autenticación web es el proceso que verifica la identidad de un usuario que intenta acceder a un servicio o recurso determinado, la autorización brinda el acceso a determinados recursos o servicios a los cuales tiene derecho el usuario previamente autenticado.

En este mecanismo de arquitectura maneja la seguridad de la aplicación mediante el control de acceso (autenticación) y personalizando las funcionalidades a las cuales tiene derecho determinado rol (autorización). Este mecanismo utiliza los servicios proporcionados por los mecanismos de acceso a datos y mecanismo de estructura de caso de uso.

La siguiente sección resume las clases participantes en el mecanismo, en el cual solo se representan los objetos relevantes a la aplicación web específicamente a la autenticación y autorización, la seguridad de otros elementos como la base de datos, el sistema operativo o el servidor web dependen de los mecanismos de seguridad definidos por el fabricante de dichos productos.

Clases Participantes

Las clases participantes específicas del mecanismo de estructura de caso de uso y mecanismo de acceso a datos han sido omitidas del mecanismo de seguridad para simplificar su modelado y hacer más comprensible el objetivo del mismo.

Clase	Descripción
Login<jspx>	Página .jspx que presenta el formulario.
Login<BackingBean>	Clase de tipo ".java" utilizada para realizar las peticiones de los eventos de la página .JSPX al servidor. Esta clase se encuentra a nivel del "Sitio Web (Web Site) o Presentación".
MainMenu<jspx>	Representa el conjunto de opciones a las cuales tiene acceso un usuario de determinado Rol dentro de la aplicación.
MainMenu<BackingBean>	Clase de tipo ".java" que ejecuta las operaciones necesarias para el despliegue y visualización del menú.
User<Entity>	Clase de tipo entidad que almacena la información de los usuarios de la aplicación.

Rol<Entity>	Clase de tipo entidad que almacena la información de los roles de la aplicación.
Menu<Entity>	Clase de tipo entidad que almacena la información de las opciones del menú de la aplicación.
MenuRol<Entity>	Entidad que representa la implementación de una relación muchos a muchos entre el menú y los roles.

Tabla 5-4: Clases Participantes Mecanismo de Seguridad

Diagrama de Clases Participantes

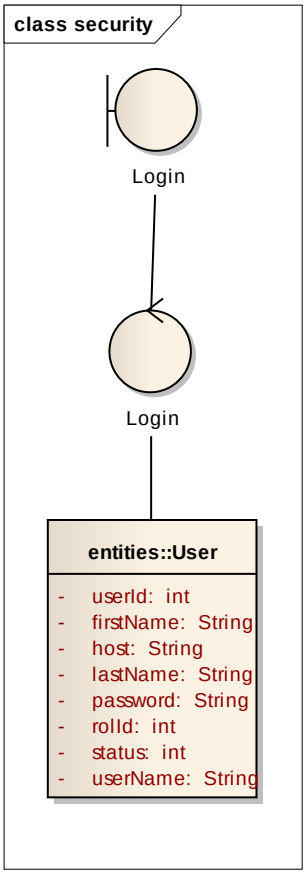


Figura 5.5: Clases participantes flujo Login

Diagrama de Secuencia (Flujo Autorización de Menus)

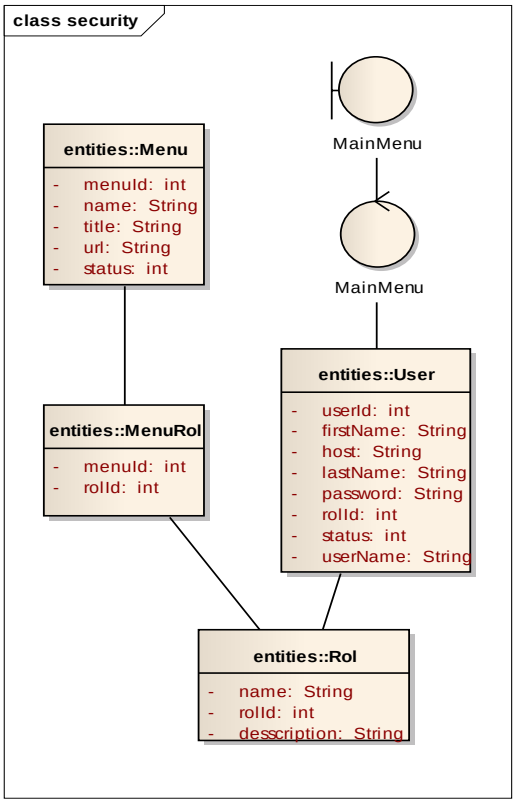


Figura 5.6: Diagrama de Secuencia - Flujo Autorizar Menús

5.2.4. Mecanismo de Comunicación

Este mecanismo define la manera de comunicarse entre los diferentes componentes de la arquitectura ARIPSAMA, específicamente la comunicación entre el componente de análisis (Agentes de Software) y el componente web. Este mecanismo hace uso del proyecto WSIG (Web Service Integration Gateway) (JADE Board, 2011) como intermediario entre la plataforma de agentes y la aplicación web. En esta sección no se presenta en detalle el proyecto WSIG, en contraste se presenta el mecanismo de comunicación de ARIPSAMA que utiliza WISIG como base para enlazar el componente de agentes con el componente web.

Problema

Los agentes de software fueron concebidos inicialmente como procesos computacionales derivados de la computación distribuida y la inteligencia artificial

(Bellifemine, Caire, & Greenwood, 2007). En la última década la evolución de las tecnologías web ha permitido que la mayoría de las empresas y organizaciones en general, planteen sus negocios como aplicaciones distribuidas a través de internet, en lo que se conoce como aplicaciones web. Por ende surge la necesidad de comunicación entre tecnologías emergentes como la tecnología de agentes de software y la tecnología web.

Solución

Los servicios web, conocidos como "Web Services", se han convertido en uno de los temas más importantes en el panorama del desarrollo de software, y se han convertido en el estándar para la interconexión de diferentes aplicaciones (JADE Board, 2011).

El objetivo de este mecanismo es exponer los servicios de agentes publicados en las páginas amarillas (DF) como servicios web, sin o con un mínimo esfuerzo adicional, dando la flexibilidad de comunicación a los agentes de software con casi cualquier aplicación existente, siempre y cuando esta aplicación soporte los servicios web.

Clases y Componentes Participantes

Clase	Descripción
AripsamaAgent<Agent>	Clase que representa un agente.
WSIG	Componente utilizado para publicar los servicios de agentes como servicios web.
Jade DF	Componente del core Jade – Páginas amarillas.
UDDI	Universal Description Discovery and Integration
ApplicationService	Interfaz que expone los servicios ofrecidos tanto por el modelo como los servicios de agente

Tabla 5-5: Mecanismo de Comunicación - Clases y Componentes Participantes

Vista de Componentes

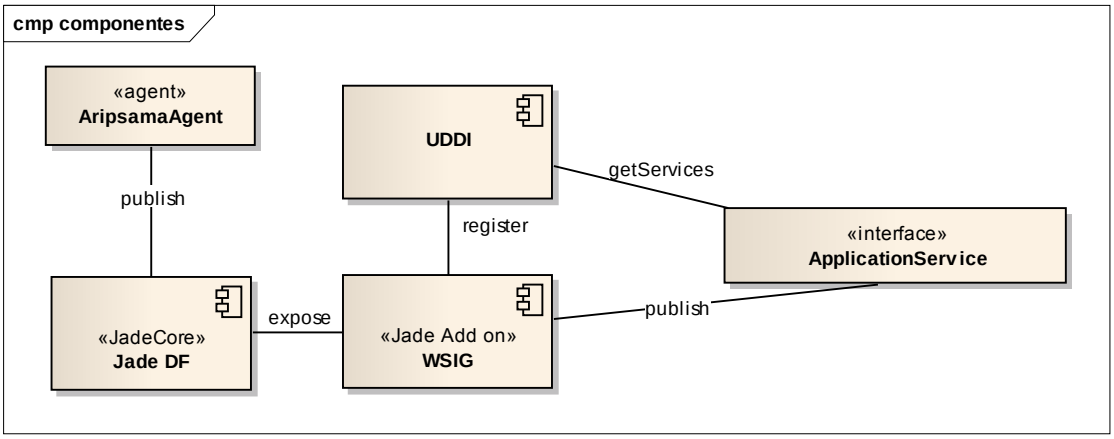


Figura 5.7: Vista de Componentes - Mecanismo de Comunicación

La Figura 5.7 muestra la comunicación entre los diversos componentes que conforman la arquitectura ARIPSAMA. Los agentes de software exponen sus funcionalidades por medio de servicios de agente, los cuales son publicados en el directorio de servicios de agentes DF (Páginas amarillas), el componente WSIG busca dentro del directorio de servicios de agentes las funcionalidades a ser expuestas como servicios web, para ello registra el servicio de agente en el UDDI, exponiéndolo como un servicio web que puede ser consumido por clientes o aplicaciones finales por medio de la interfaz ApplicationService

5.2.5. Mecanismo de Excepciones

Problema

El desarrollo de aplicaciones debe considerar la correcta identificación y tratamiento de excepciones dentro del contexto de la aplicación. Cuando no se maneja un estándar para tratar los diferentes errores que ocurren durante la ejecución del programa resulta ser complicado mostrar al usuario un mensaje acorde para que este pueda entender un suceso y al administrador notificarle cual es el problema real. Durante el desarrollo todas las personas involucradas tienden a tratar las excepciones de diferente forma lo cual puede llegar a ser no tan agradable cuando se cuenta con una cantidad diversa de aplicaciones.

Solución

La estandarización de cómo se deben tratar las excepciones, sin importar su tipo, incluye diferentes formas de registro y auditoría de las mismas, las cuales son: Base de Datos, Logs Servidor, Archivo Plano ".txt" y Notificación por correo con el error ocurrido en la aplicación, el cual puede ser enviado a una o varias personas.

Los niveles desarrollados incluyen cada funcionalidad individual, combinaciones de dos en dos y todas lo cual procura que siempre los administradores de los sistemas puedan estar al tanto de lo que ocurre para dar respuesta rápida ante alguna situación en particular.

Clases Participantes

Clase	Descripción
Exception	Clase base utilizada para la implementación de la funcionalidad de manejo de errores en los aplicativos. Esta clase está desarrollada forma parte del sdk de java solo se debe extender.
Java Exceptions	Clases utilizadas para extender la clase base Exception y agregar nuevos comportamientos dependiendo de la naturaleza de la excepción ocurrida. Los tipos de excepciones del SDK pueden ser RuntimeException, NullPointerException, entre otras.
AppException	Clase encargada de la implementación de los registros o pistas de las excepciones ocurridas en las aplicaciones. Debe utilizarse para el manejo de excepciones ya que es la que permite el registro de los errores en los diferentes

	repositorios. Esta clase debe ser desarrollada en cada solución.
ALog	Clase abstracta cuya funcionalidad es registrar las excepciones ocurridas en las aplicaciones, define los métodos y atributos a implementarse para el registro de excepciones en el log.
TextLog	Clase cuya funcionalidad es registrar en un archivo plano las excepciones ocurridas en las aplicaciones. Esta clase debe heredar de la clase base ALog e implementar sus métodos. Para esta funcionalidad se debe realizar una configuración previa.
DBLog	Clase cuya funcionalidad es registrar en la base de datos las excepciones ocurridas en las aplicaciones, para esta funcionalidad se debe realizar una configuración previa.
Sistema	Esta clase representa el sistema en el cual, se crea la instancia y se ejecuta el llamado para el registro de la excepción generada.

Tabla 5-6: Clases Participantes - Mecanismo de Excepciones

Diagrama de Clases Participantes

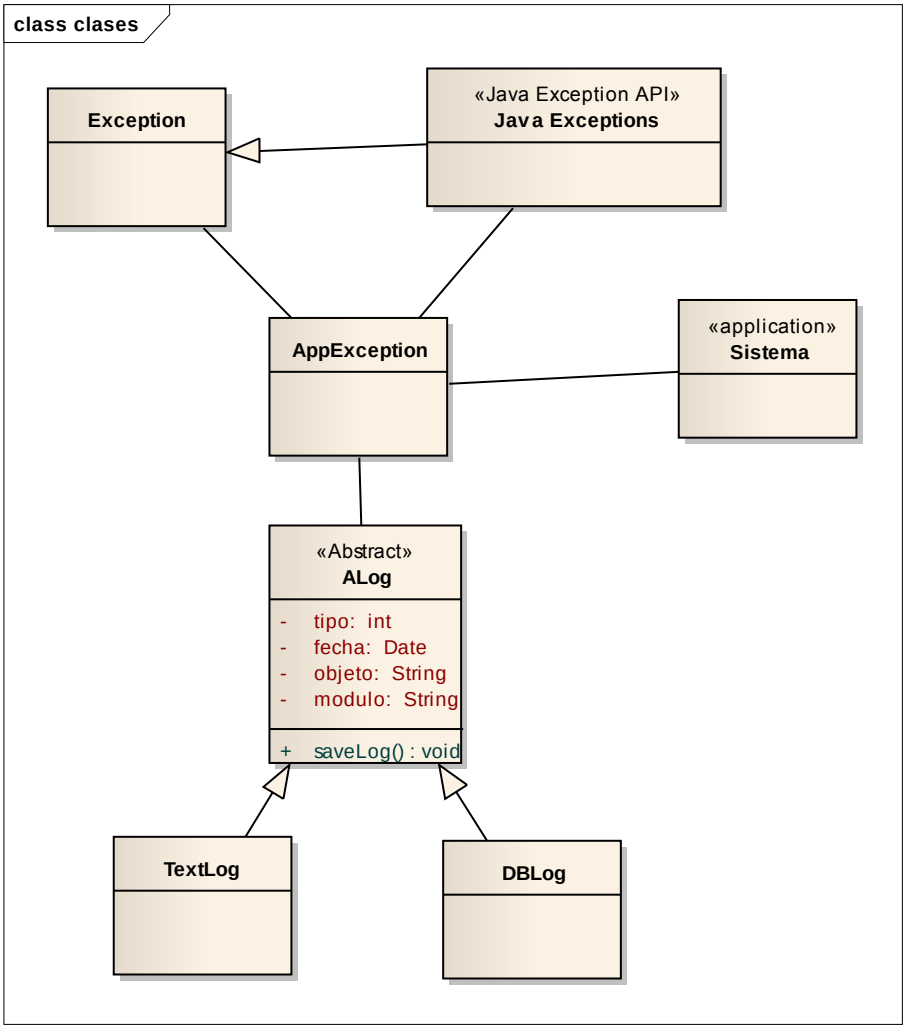


Figura 5.8: Clases Participantes - Excepciones

6. ARIPSAMA: Implementación

El presente capítulo tiene el propósito de proveer una descripción formal del trabajo propuesto, en el cual se pueda encontrar una definición y explicación del desarrollo de cada uno de los componentes que conforman la arquitectura ARIPSAMA.

6.1. Descripción del Ambiente

Para cubrir con los objetivos planteados en este proyecto, se ha diseñado un ambiente o entorno de realización basado en cuatro componentes principales como lo son: el repositorio, el componente de análisis, el catálogo y la aplicación web tal como se observa en la Figura 6.1.

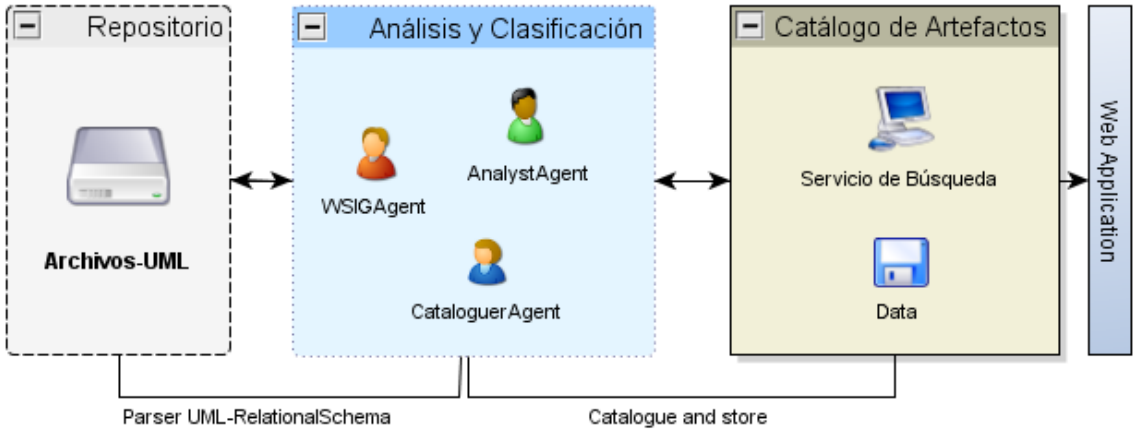


Figura 6.1: Ambiente ARIPSAMA vista Informal

6.1.1. Repositorio

Este componente está compuesto de los artefactos desarrollados para un sistema en particular, cuya lógica será analizada y clasificada para dar origen al catálogo de artefactos de funcionalidades reutilizables. El mismo consta de los modelos y diagramas UML que constituyen la arquitectura o representan un flujo o proceso de un sistema. El repositorio consiste de diagramas de procesos/actividades, en formato UML, XMI e imágenes descriptivas del modelo diagramado.

El repositorio consta de un servidor donde se almacenan los artefactos UML correspondientes a las arquitecturas de los sistemas previamente desarrollados.

6.1.2. Componente de Análisis

Esta componente está constituida por los agentes de software que conforman el sistema, y son los encargados del monitoreo, análisis y clasificación de los artefactos (diagramas) UML contenidos en el repositorio y que conforman la base del catálogo de modelos.

El sistema está compuesto de tres agentes principales:

- Un monitoreador de peticiones de usuario denominado WSIGAgent (alerta a otros agentes cuando se carga un nuevo archivo al repositorio). Este agente forma parte del componente WSIG y es el encargado de la comunicación entre el ambiente web y el ambiente de agentes. Las peticiones de aplicaciones clientes son realizadas utilizando como intermediarios los servicios web expuestos por medio de la interfaz ApplicationService. Estas peticiones son recibidas por el WSIGAgent, el cual las envía al analista quien se encarga de procesarlas y redireccionarlas al catalogador para su posterior publicación en el catálogo de artefactos.

- Un analista, que recibe la información del monitoreador y realiza la conversión y análisis semántico del modelo UML. El analista recibe la información del modelo cargado por parte del monitoriador, el modelo en formato xmi es procesado por el analista y almacenado en un esquema relacional, el cual será utilizado por el catalogador como fuente de información primaria para responder las consultas realizadas por los usuarios del sistema.

- Un catalogador, encargado de clasificar la información almacenada y registrarla en el catálogo, también es responsable de procesar las consultas del usuario para devolver el modelo mejor adaptado a los criterios de búsqueda.

6.1.3. Catalogo

Esta componente está constituida por aquella información clasificada y analizada por el catalogador y el analista; la cual contiene información relevante de los modelos UML analizados. Esta información será la base para el motor de búsqueda el cual brinda un acceso rápido y sencillo a los modelos catalogados.

Constituye la sección de información procesada, almacena la conversión relacional correspondiente a los artefactos UML. Contiene una Base de Datos con la descripción de funcionalidades comunes y un servicio que funciona como motor de búsqueda, base para la extracción de modelos existentes útiles para los nuevos sistemas.

6.1.4. Aplicación Web

Constituye el componente más cercano al usuario, consta de una interfaz para el usuario, permite realizar búsqueda de modelos y ofrece vistas previas de las funcionalidades almacenadas que coinciden con diversos criterios de búsqueda especificados por el usuario. Brinda como resultado el listado de artefactos con mayor grado de similitud los cuales son la base para los nuevos sistemas.

Cada uno de los componentes del ambiente, interactúan con el usuario a través de la componente web, la cual será encargada de proveer las interfaces de usuario para la carga y descarga de modelos, la búsqueda y despliegue de resultados de las consultas y el manejo de los componentes de seguridad.

En la Figura 6.2 se observa el flujo de actividades necesarias para buscar un modelo dado por parte del usuario o bien, para cargar un nuevo modelo el cual es analizado, clasificado y almacenado por parte de los agentes de software existentes en el sistema.

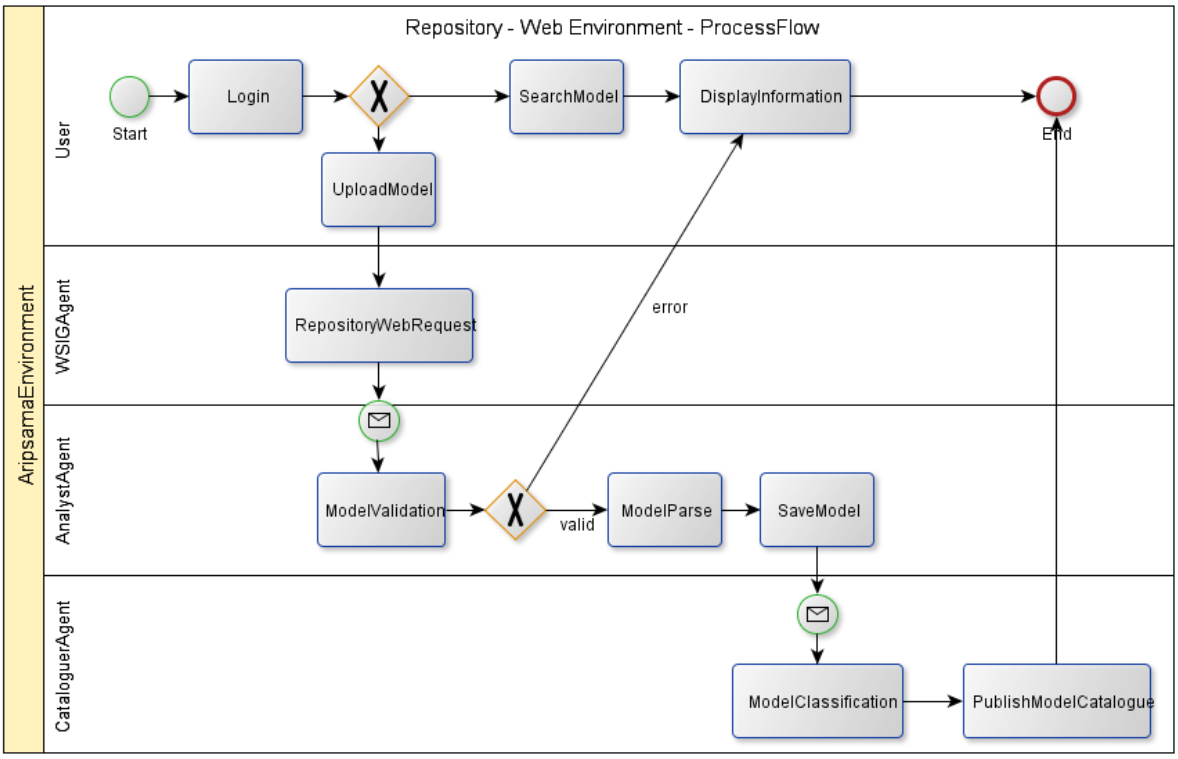


Figura 6.2: Flujo de eventos ARIPSAMA

6.2. Conversión de Modelos UML-Esquema Relacional

Para que el motor de búsqueda funcione adecuadamente, es necesario representar los modelos en un formato que facilite la extracción de los conceptos y funcionalidades almacenados en él. La representación del modelo depende de la estructura que se desea manejar y el tipo de consultas a realizar, por ejemplo una representación en grafos puede facilitar la validación de relaciones y semántica de un modelo, pero el tiempo de consulta puede no ser óptimo para un gran volumen de información (Robinson & Woo, 2004). Representar los modelos como flujos de eventos (Blok & Cybulski, 1998), es una alternativa viable, pero requiere la creación de flujos por cada nuevo modelo.

En este proyecto se desea una representación que permita extraer los modelos de una manera rápida y con un porcentaje de precisión aceptable, para ello a diferencia de los trabajos relacionados, el análisis no se realiza al momento de la búsqueda del modelo, sino al momento de almacenar el mismo.

Este esquema brinda la facilidad de resumir la búsqueda a una simple consulta SQL, con un mínimo de análisis en la realización de la misma. Para la conversión de modelos UML a esquemas relacionales, se ha decido plantear un modelo de conversión el cual consta de dos pasos ver Figura 6.3, los cuales se describen a continuación:

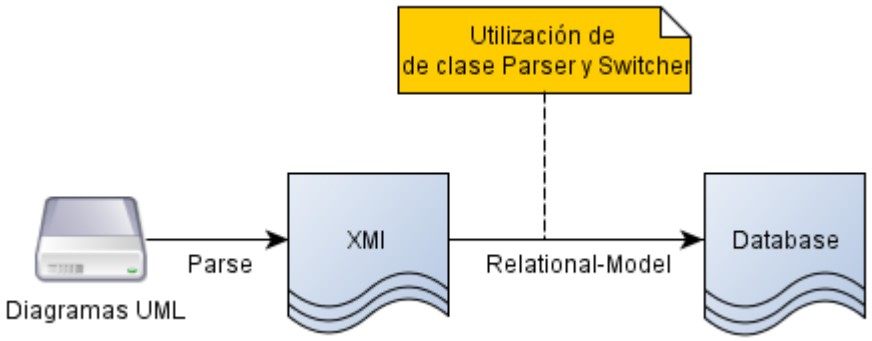


Figura 6.3: Proceso de Conversión UML - Esquema Relacional

Como primer paso se debe contar con los modelos representados en UML (Diagramas de Actividades/Procesos) representados en su estructura XML/XMI, esta etapa debe realizarse de forma manual, ya que es tarea del analista almacenar el modelo en formato XMI, es una tarea sencilla ya que la mayoría de las herramientas de modelación actuales soportan el formato XMI. Este flujo puede observarse en la Figura 6.4.

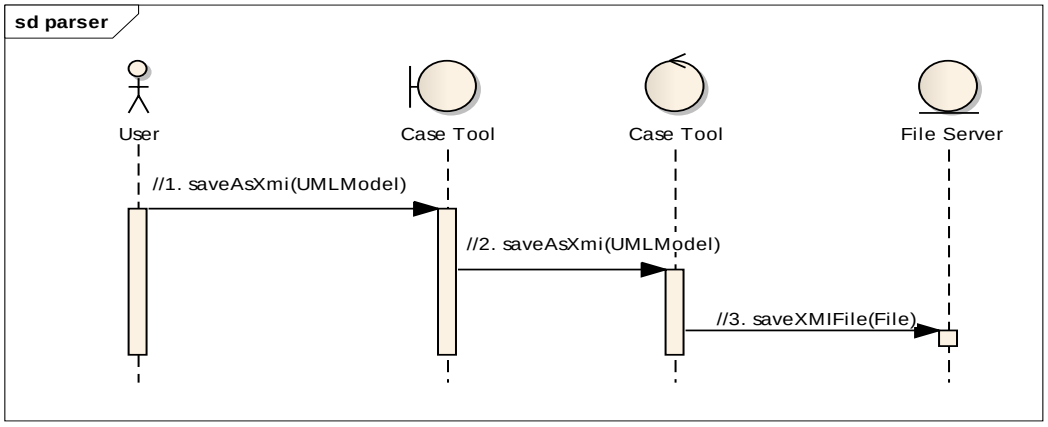


Figura 6.4: Flujo de exportación de modelo UML a XMI

El segundo paso corresponde al proceso de conversión del modelo en formato XMI al esquema Relacional. Para ello se ha tomado como referencia los procesos de conversión UML a XMLSchema (Mun-Young, Lim, & Kyung-Soo, 2005; Narayanan & Ramaswamy, 2005; Routledge, Bird, & Goodchild, 2002), debido a la similitud de los esquemas XML con una tabla relacional. En este punto es necesario definir un mapeo entre los elementos UML, específicamente del diagrama de actividad UML, y el esquema relacional propuesto tal como se muestra en la Tabla 6-1.

Elemento o Atributo UML	Correspondencia Relacional
xmi:type="uml:Activity"	ACTION
xmi:type="uml:Model"	MODEL
xmi:type="uml:Diagram"	MODEL
xmi:type="uml:Action"	ACTION
xmi:type="uml:ControlFlow"	CONTROL_FLOW
xmi:type="uml:Transition"	CONTROL_FLOW
xmi:type="uml:MergeNode"	NODE con tipo MERGE
xmi:type="uml:ForkNode"	NODE con tipo FORK o BRANCH
xmi:type="uml:JoinNode"	NODE con tipo JOIN
xmi:type="uml:InitialNode"	NODE con tipo INITIAL
xmi:type="uml:ActivityFinalNode"	NODE con tipo FINAL
xmi:id	ID

Tabla 6-1: Mapeo de Diagrama de Actividad UML a esquema Relacional

El esquema relacional planteado presenta una estructura definida por seis tablas de base de datos, tal como se muestra en la Figura 6.5:

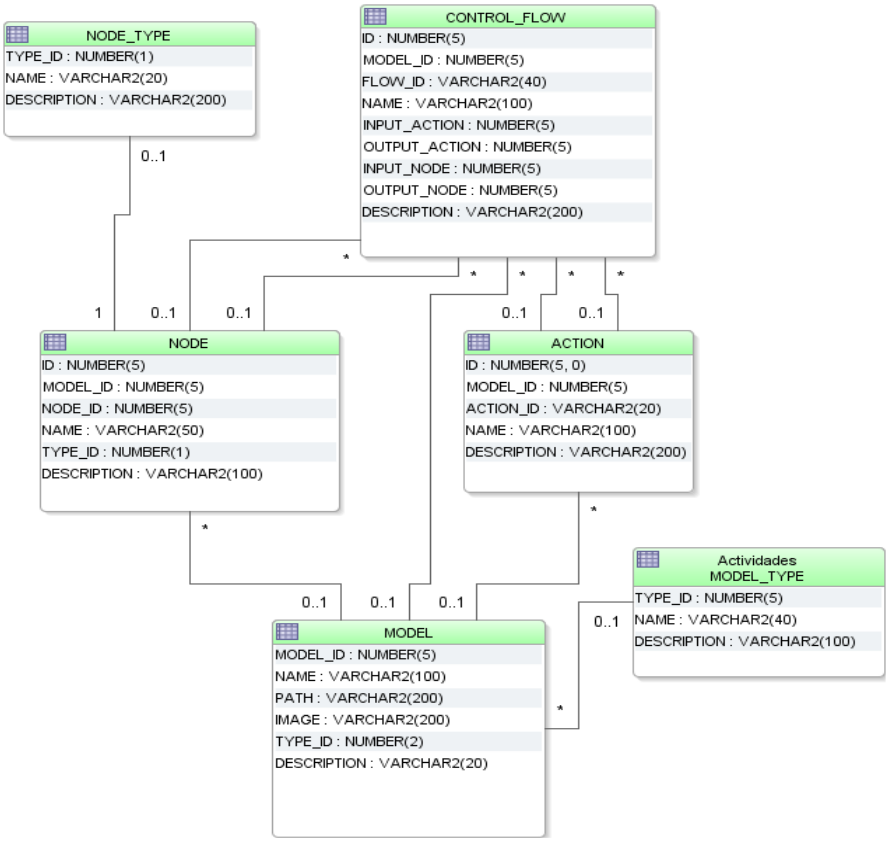


Figura 6.5: Esquema relacional propuesto

La tabla MODEL, almacena la información general del modelo, la tabla CONTROL_FLOW, la información de las relaciones y flujos, la tabla ACTION, almacena la información de las actividades y tareas, y la tabla NODE, almacena la información de los nodos clasificados según los criterios definidos en la tabla NODE_TYPE, entre los cuales tenemos el tipo MERGE, JOIN, BRANCH, FORK, INITIAL y FINAL. Por definición todo modelo o proceso diagramado con diagramas de actividades/procesos debe contar mínimo con el nodo inicial y final, los demás son opcionales y dependen de la complejidad del modelo.

La tabla MODEL_TYPE, es una tabla cargada por los usuarios para categorizar el modelo en un dominio específico.

6.3. Modelo de Análisis y Catalogación – Agentes

La componente de Análisis y catalogación de modelos, está manejada totalmente por agentes de software. La arquitectura plantea la existencia de tres agentes distintos, el WSIGAgent, el AnalystAgent y el CataloguerAgent.

- WSIGAgent: Es el agente que recibe las peticiones del usuario y las comunica al agente con los servicios adecuados para procesarlas. Este agente es externo y pertenece al subsistema WSIG (Web Service Integration Gateway) descrito en el mecanismo de comunicación entre el ambiente Web y el ambiente de Agentes ver sección 5.2.4.

- AnalystAgent: Es el agente que procesa las solicitudes de carga de un nuevo modelo al repositorio. Este agente hace uso del proceso de conversión UML al esquema relacional descrito en la sección 6.2. Una vez convertido el modelo, el agente se encarga de almacenarlo en la entidad correspondiente de la base de datos. La Figura 6.6 muestra el flujo de trabajo del analista al momento de procesar una petición de conversión y almacenado de un nuevo modelo.

- CataloguerAgent: Una vez el analista ha almacenado el modelo, el catalogador revisa las tablas y crea las relaciones y secuencias entre cada una de las actividades del modelo. Tiene la tarea de recrear un pseudo grafo que defina el flujo original de las tareas del modelo, además clasifica el modelo de acuerdo con el dominio definido por el usuario. Este agente maneja el proceso de ponderación de los modelos, llevando un conteo de las descargas y la valoración dada por el usuario.

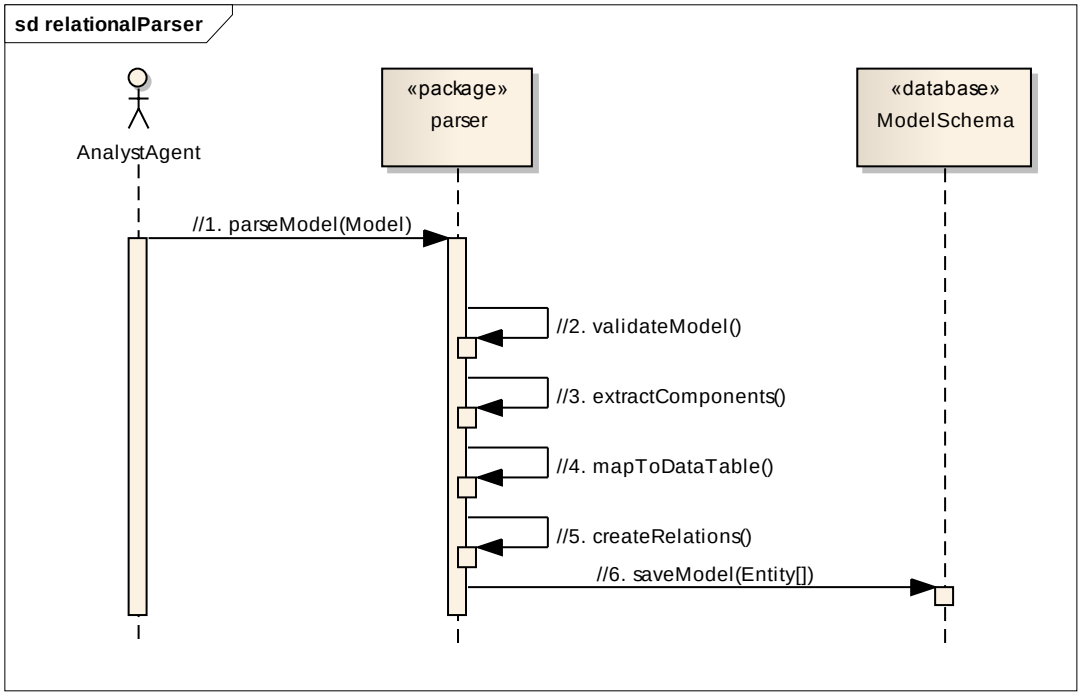


Figura 6.6: Proceso de trabajo del AnalystAgent para cargar un modelo

La Figura 6.6 muestra la secuencia de acciones del agente analista al recibir la información de carga de un nuevo modelo al sistema por parte del monitoreador (WSIGAgent). Estas tareas consisten en validar la estructura xml del modelo, realizar la extracción de los componentes significativos para la conversión, mapear cada elemento con la entidad correspondiente, crear las relaciones básicas y almacenar el modelo.

6.3.1. Proceso de Ponderación de Modelos

La diferencia principal de la arquitectura ARIPSAMA en comparativa a los trabajos relacionados descritos en al capítulo 3 consiste en realizar la mayor parte del análisis al momento de almacenar y catalogar el modelo. Además de ofrecer un método de retroalimentación basada en ponderaciones del modelo que permita tener mayor impacto en el resultado obtenido por medio de una consulta de usuario.

El proceso de ponderación de modelos se basa en tres criterios distintos:

- **Criterio de valoración de usuario:** este criterio se retroalimenta de las ponderaciones o calificaciones suministradas por el usuario o los usuarios que

consultan determinado modelo. La puntuación corresponde a un promedio de las calificaciones dadas por todos los usuarios que han evaluado el modelo tal como se muestra en la siguiente ecuación.

$$c1 = \frac{\sum tPuntos}{tEvaluaciones} \quad (6.1)$$

Donde,

tPuntos *= total de puntos que ha recibido el modelo. La puntuación varía en escala de 1 a 5, siendo 1 la puntuación más baja y 5 la puntuación más alta.*

tEvaluaciones *= cantidad de evaluaciones de usuario que ha recibido un modelo.*

c1 *= variable que almacena el resultado correspondiente al criterio de evaluación de usuario.*

- **Criterio por descargas:** este criterio se retroalimenta de la cantidad de veces que ha sido descargado un modelo, la variante significativa de este método con un contador de descargas común, es la posibilidad de brindar una vista previa del modelo antes de descargarlo, por lo cual se infiere que el usuario ya le ha dado su aprobación inicial al modelo al momento de descargarlo, tal como se muestra en la siguiente ecuación.

$$c2 = \sum descarga \quad (6.2)$$

Donde,

c2 *= variable que almacena el resultado correspondiente a la sumatoria total de las descargas que ha recibido un modelo.*

descarga *= corresponde a una descarga del modelo.*

- **Criterio de similitud léxico semántica:** Este método se basa en las ponderaciones obtenidas de la comparación léxico semántica entre la información del modelo, sus acciones y sus relaciones.

  Cada elemento tiene la siguiente ponderación:

Modelo – 5pts: Un criterio buscado que coincide con la descripción del modelo tiene la ponderación mayor debido a que la funcionalidad representada en un modelo por lo general se indica en el nombre o la descripción del modelo.

Acción 3pts: Un criterio buscado que coincide con la descripción de una acción o tarea recibe una ponderación de intermedia, debido a que representa una de las tareas que componen el proceso diagramado más, no definen el proceso completo.

Relación 1pts: Un criterio de búsqueda puede coincidir con una relación ya sea por la descripción o el nombre de la relación o por la unión de una o dos tareas coincidentes con el criterio de búsqueda.

La suma de la coincidencia de los tres factores corresponde a la ponderación total del modelo tal como se muestra en la siguiente ecuación.

$$c3 = \sum_{i=0}^{n} \{e = c | p = p + pe\} \qquad (6.3)$$

Donde,

i *= contador que recorre los elementos de un modelo*

n *= corresponde al número de elementos del modelo*

e *= corresponde al elemento evaluado*

c *= corresponde al criterio de búsqueda*

p *= corresponde a la ponderación acumulada del modelo*

pe *= corresponde a la ponderación del elemento*

c3 *= corresponde a la ponderación en base al criterio de evaluación léxico semántica.*

Aunque un modelo sea almacenado con un nombre equivocado, la coincidencia de tareas y relaciones permitirá que el modelo sea tomado en cuenta y devuelto como resultado de la consulta.

La puntuación final de un modelo dado está definido por la suma de las tres ponderaciones anteriores, tal y como se observa en la siguiente ecuación:

$$pfinal = c1 + c2 + c3 \quad (6.4)$$

Los resultados de una consulta devuelven los modelos en orden descendente de ponderación, es decir, los primeros modelos listados tienen la mayor similitud con el criterio buscado.

6.4. Vistas de Arquitectura

Esta sección muestra la arquitectura propuesta en diferentes vistas complementarias que ayudan a una mejor comprensión de la misma.

6.4.1. Vista de Paquetes

La arquitectura propuesta se desarrolló por medio de una agrupación lógica de componentes, en los cuales se identifica el Paquete Web, el paquete del Modelo y el Paquete de Agentes, intercomunicados por los servicios web publicados. La relación y contenido de cada uno de estos paquetes a nivel de solución, se ilustra en la Figura 6.7.

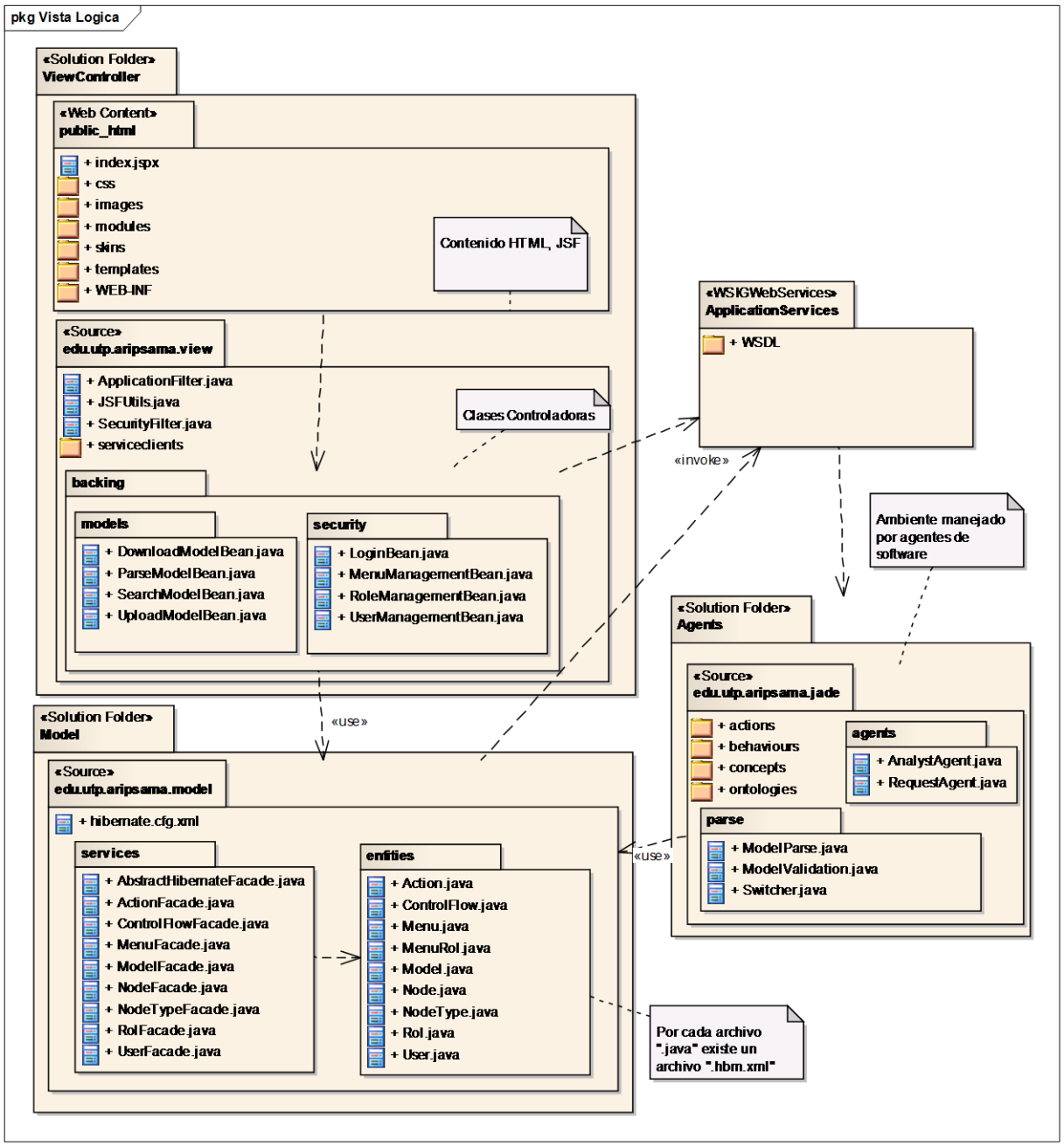


Figura 6.7: Vista de Paquetes – ARIPSAMA

Paquete ViewController

Este paquete contiene las clases del sistema dividido en dos paquetes principales:

- **WebSite(public_html)**: contiene las páginas, templates, estilos e imágenes que se utilizan en la capa de presentación web del aplicativo.

- **Source**: Este paquete contiene clases de utilidades globales a la aplicación, así clases manejadoras de filtros, sevlets entre otros. Además incluye los siguientes sub-paquetes.

    - **Backing:** contiene cada uno de los backing beans utilizados por las páginas jspx, agrupados en paquetes de acuerdo a su funcionalidad.

Una vista mas detallada del contenido del paquete ViewController se muestra en la Figura 6.8.

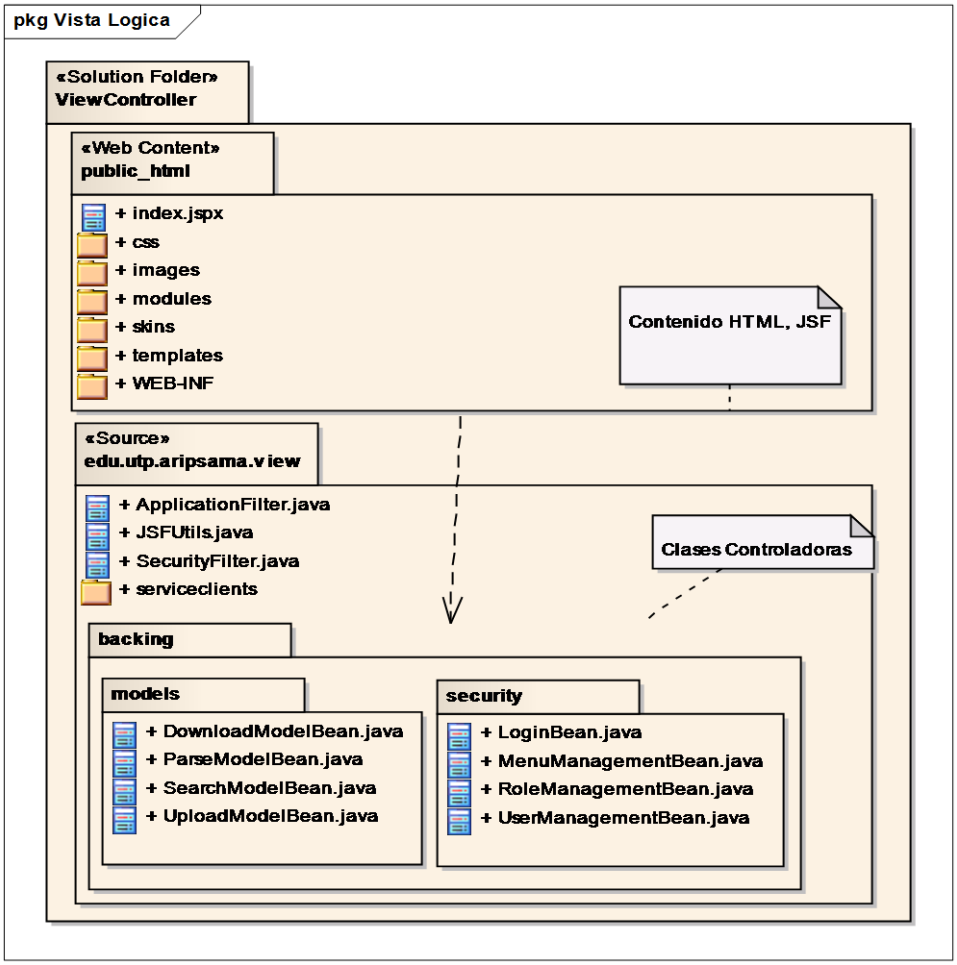


Figura 6.8: Paquete ViewController

Paquete Model

Este paquete contiene las clases y componentes del modelo, encargados del acceso a datos y la lógica del negocio. Se encuentra estructurado en dos sub-paquetes descritos a continuación:

- **Entities:** Contiene las entidades del sistema y las respectivas relaciones (asociaciones) entre ellas. Son implementadas por medio de entidades hibernate.
- **Services:** Componente que encapsula las consultas y las entidades, a la vez que abstrae la complejidad del acceso a datos. Este componente expone los servicios de negocio a través de interfaces que son consumidas por el paquete ViewController. Se implementa por medio del patrón Facade.

Una vista mas detallada de este paquete se observa en la Figura 6.9

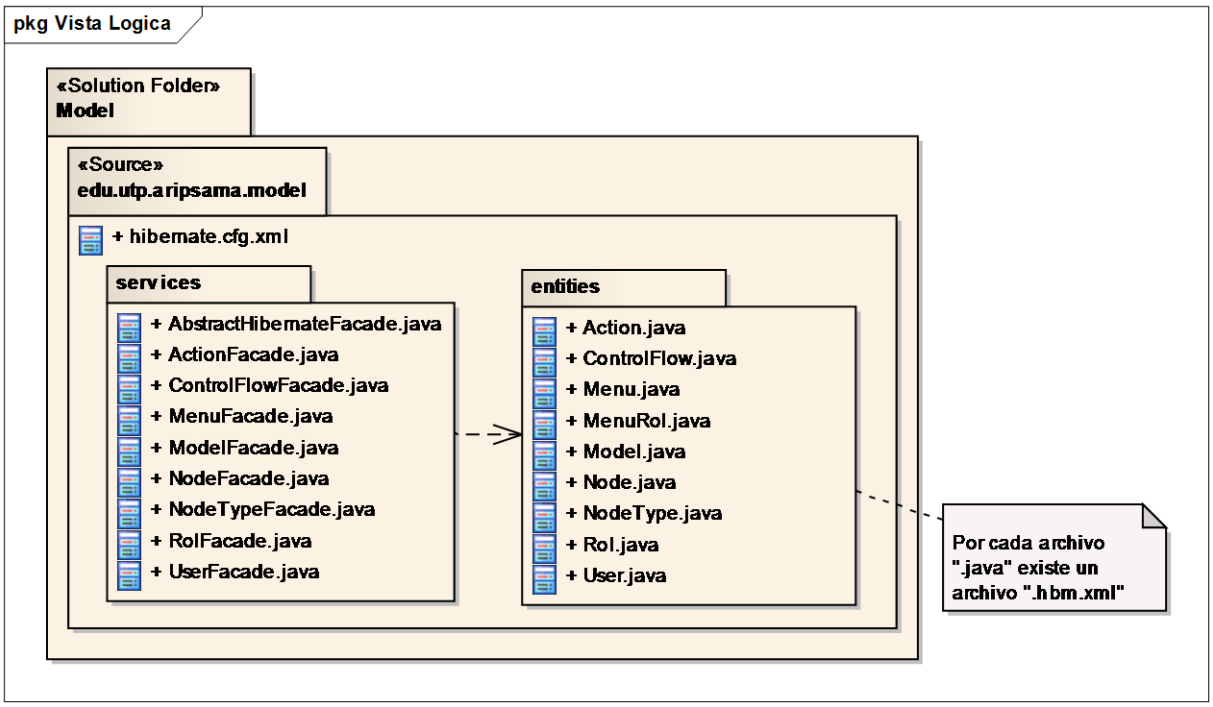


Figura 6.9: Paquete Model

Paquete Agents

Este paquete contiene la implementación de los agentes de software que residen en la plataforma Jade. También contiene las clases que manejan la conversión del modelo. Los sub-paquetes más significativos son: paquete agents, que contiene las clases que implementan los agentes de software y el sub-paquete parse, que contiene las clases que efectúan la conversión de modelos.

La Figura 6.10 muestra el contenido del paquete Agents.

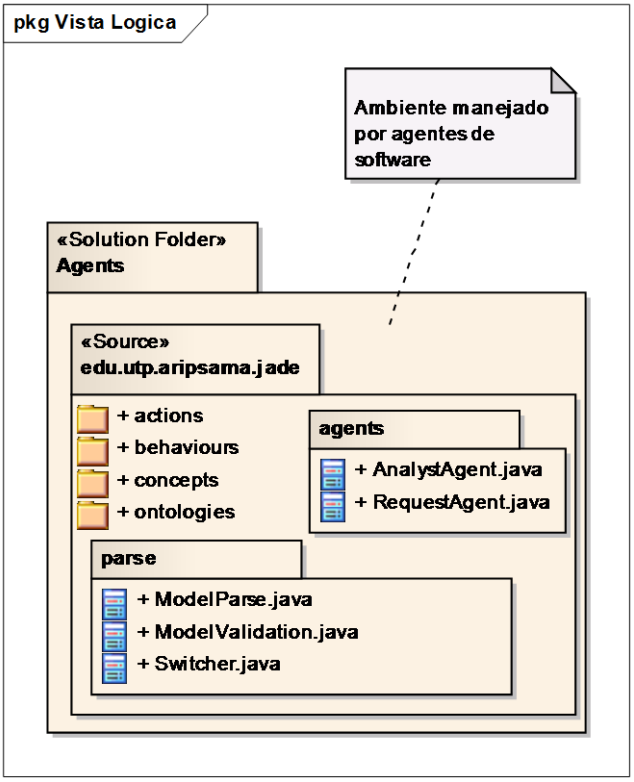


Figura 6.10: Paquete Agents

Paquete ApplicationServices

Este paquete contiene la descripción y referencia a los servicios de agentes que han sido publicados como servicios web por medio del WSIG.

6.4.2. Vista de Componentes

Esta vista presenta como están estructurados los paquetes a nivel de componentes ejecutables. Las relaciones entre ellos, y los protocolos de comunicación utilizados. La Figura 6.11 y Figura 6.12 muestran en más detalle la estructura de esta vista.

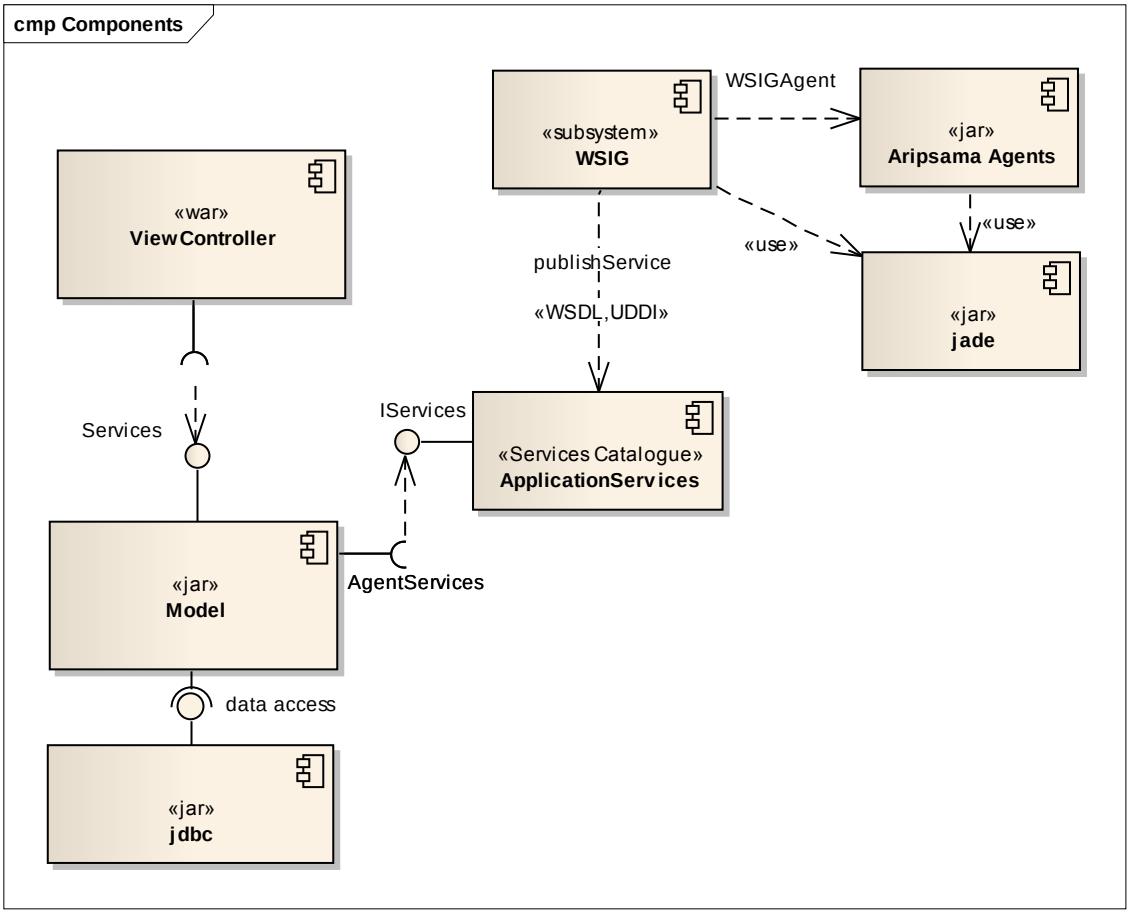


Figura 6.11: ARIPSAMA Vista de Componentes

El componente Model se comunica con la base de datos a través de la interfaz ofrecida por el driver jdbc. También ofrece la interfaz Services, la cual es utilizada por el componente ViewController y consume los servicios de agentes.

El componente Agents utiliza los paquetes ofrecidos por el framework Jade para definir loas agentes y sus respectivos comportamientos y servicios.

El subsistema WSIG a extrae los servicios de agentes y los publica en el servidor, mientras que define al agente WSIGAgent para que funcione como intermediario entre la componente web y la componente de agentes.

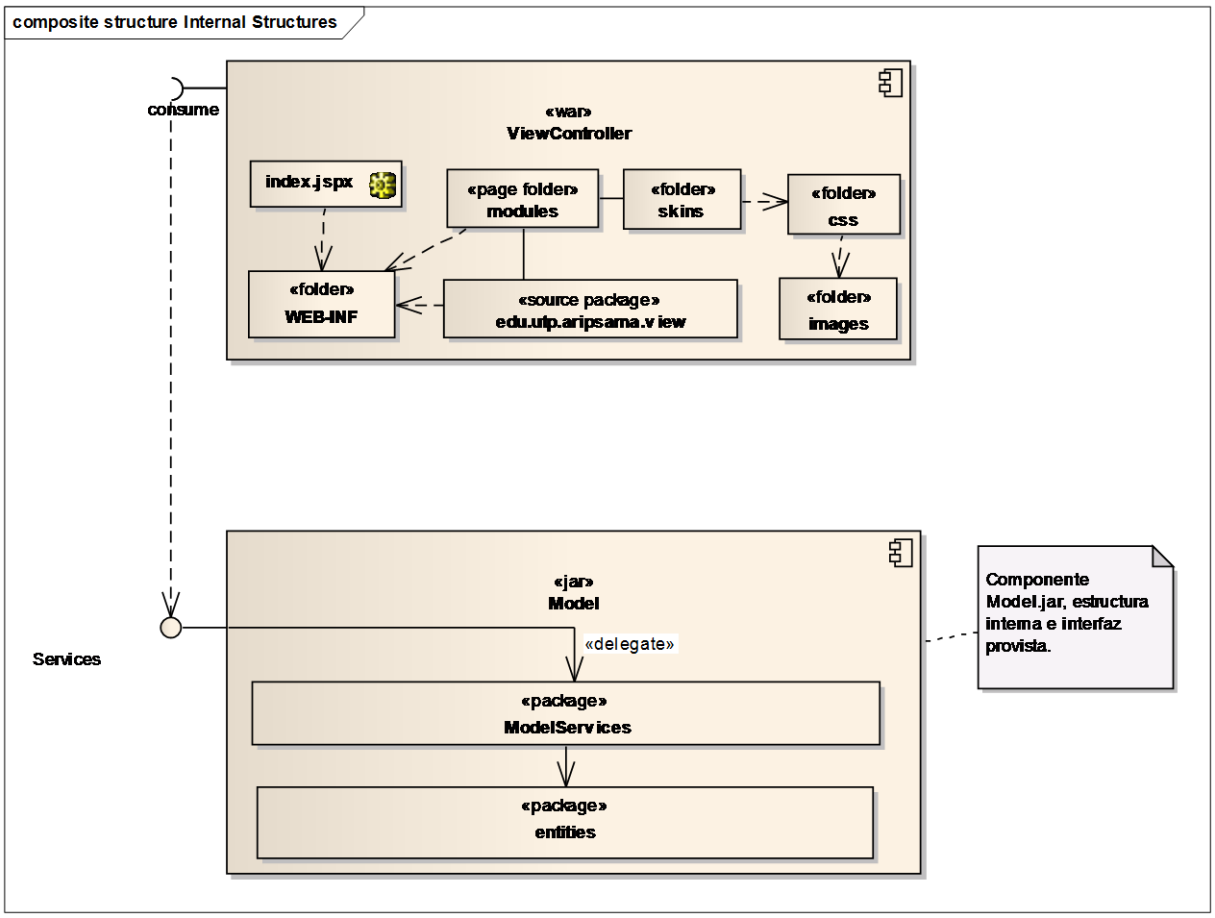


Figura 6.12: Detalle ViewController y Model

El componente ViewController se comprime en un Web Application Resource (War) que se comunica con el componente Model, comprimido como una librería java con extensión jar (Java Application Resource).

6.4.3. Vista de Despliegue

Esta vista muestra la forma en que son agrupados los componentes en los diversos dispositivos físicos. Tal como se observa en la Figura 6.13.

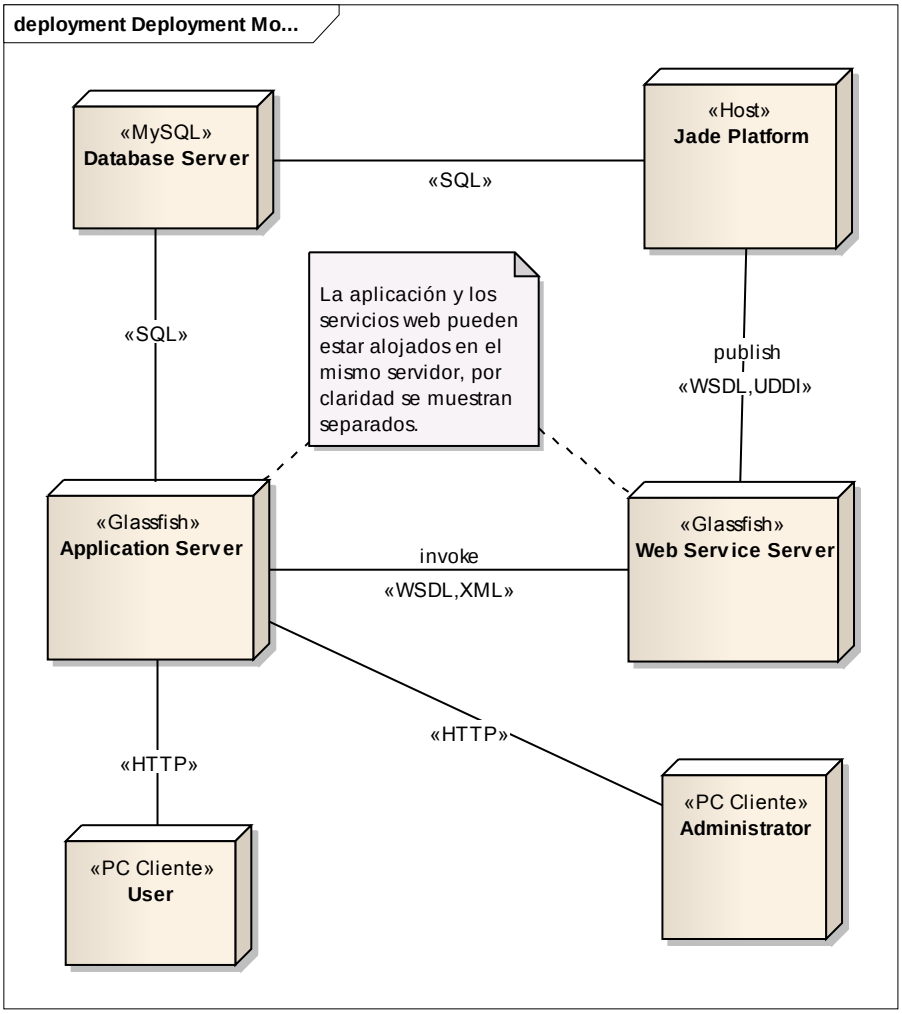


Figura 6.13: Vista de Implementación

El Database server contiene la instalación del MySQL que almacena el esquema relacional propuesto y la información de los modelos. El ApplicationServer aloja la aplicación web, el Web Service Server aloja los servicios web. La plataforma Jade aloja los agentes de software.

Las pc cliente corresponden al usuario que utiliza la aplicación.

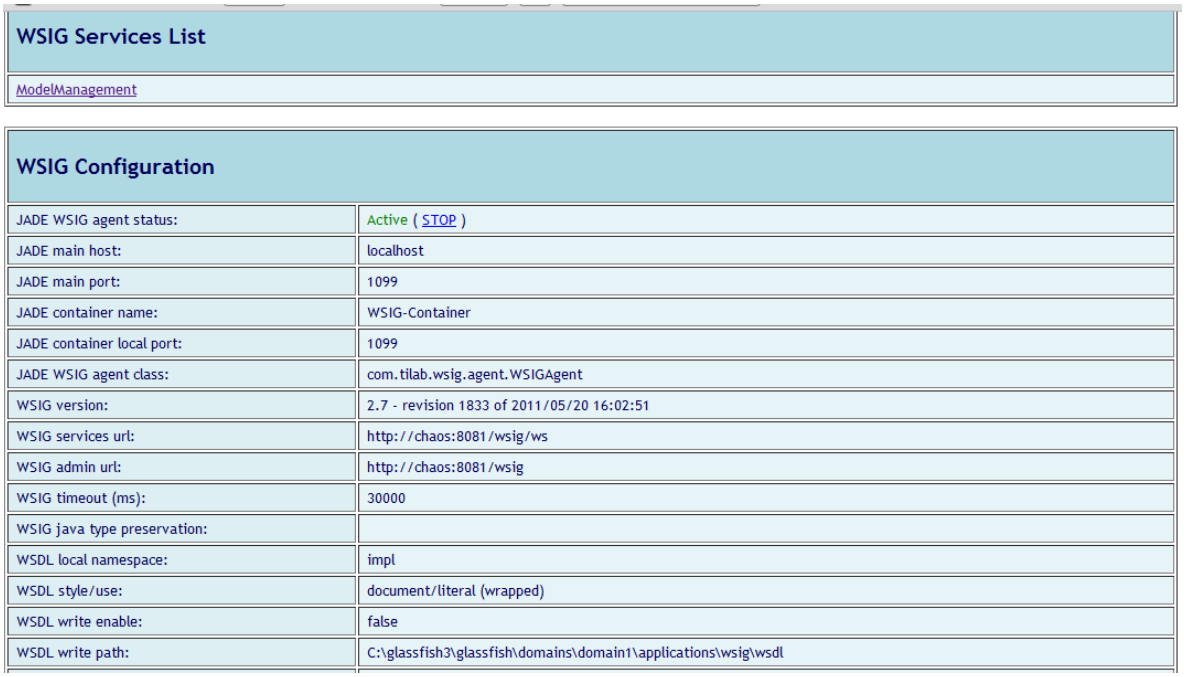
6.5. Caso de Estudio

Este capítulo se enfoca en la descripción del prototipo que utiliza la arquitectura propuesta.

Se describen las diversas interfaces que permiten a un usuario interactuar de manera transparente con los componentes de la arquitectura.

6.5.1. Interfaz WSIG

La interfaz WSIG consiste de una aplicación Web, un agente de software (WSIGAgent) y la definición y acceso a los servicios de agentes publicados como servicios web.



WSIG Services List

ModelManagement

WSIG Configuration

JADE WSIG agent status:	Active (STOP)
JADE main host:	localhost
JADE main port:	1099
JADE container name:	WSIG-Container
JADE container local port:	1099
JADE WSIG agent class:	com.tilab.wsig.agent.WSIGAgent
WSIG version:	2.7 - revision 1833 of 2011/05/20 16:02:51
WSIG services url:	http://chaos:8081/wsig/ws
WSIG admin url:	http://chaos:8081/wsig
WSIG timeout (ms):	30000
WSIG java type preservation:	
WSDL local namespace:	impl
WSDL style/use:	document/literal (wrapped)
WSDL write enable:	false
WSDL write path:	C:\glassfish3\glassfish\domains\domain1\applications\wsig\wsdl

Figura 6.14: Información de Configuración WSIG

La Figura 6.14 muestra las propiedades de configuración de la interfaz WSIG, en la cual se observan las propiedades del host donde reside el WSIGAgent, el contenedor y puerto; así como la ruta donde se publica el WSDL de los servicios publicados.

En el listado de servicios se observa el servicio ModelManagement, que corresponde con el servicio que contiene las operaciones ofrecidas por el AnalystAgent y CataloguerAgent.

ModelManagement	
Name:	ModelManagement
Prefix:	-
Mapper class:	-
Jade ontology:	model-ontology
Jade agent:	AnalystAgent1@ARIPSAMAPlatform
UDDI service key:	-
WSDL url:	http://chaos:8081/wsig/ws/ModelManagement?WSDL
Operations:	modelRequest getAgentInfo parseModel getModelInfo

Figura 6.15: Detalle de Servicio de Agentes

La Figura 6.15 muestra el detalle del servicio ModelManagement, en donde se observan las operaciones ofrecidas por el agente AnalystAgent y la ontología utilizada para el intercambio de mensajes entre todos los agentes que residen en la plataforma de agentes ARIPSAMAPlatform.

La Figura 6.16 muestra un extracto del archivo WSDL que contiene la descripción del servicio ModelManagement. Esta descripción es utilizada por los clientes que consumen el servicio.

```
▼<wsdl:definitions xmlns:impl="urn:ModelManagement" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="ModelManagement"
 targetNamespace="urn:ModelManagement">
 ▼<wsdl:types>
  ▼<xsd:schema xmlns:impl="urn:ModelManagement" xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ModelManagement">
     <xsd:annotation/>
   ▼<xsd:element name="modelRequest">
    ▼<xsd:complexType>
     ▼<xsd:sequence>
        <xsd:element name="modelId" type="xsd:int"/>
      </xsd:sequence>
     </xsd:complexType>
   </xsd:element>
   ▼<xsd:element name="modelRequestResponse">
    ▼<xsd:complexType>
     ▼<xsd:sequence>
        <xsd:element name="modelRequestReturn" type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
   </xsd:element>
   ▼<xsd:element name="getAgentInfo">
    ▼<xsd:complexType>
       <xsd:sequence/>
     </xsd:complexType>
   </xsd:element>
   ▼<xsd:element name="getAgentInfoResponse">
```

Figura 6.16: Extracto de WSDL generado

La descripción WSDL del servicio y su publicación en el UDDI permite a los clientes realizar invocaciones con el protocolo SOAP, a través de XML.

Una petición SOAP que consume una operación ofrecida por el servicio ModelManagement se presenta en la Figura 6.17.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:urn="urn:ModelManagement">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:getAgentInfo soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      </urn:getAgentInfo>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 6.17: Petición SOAP de ejemplo

La respuesta brindada por el servicio es mostrada en la Figura 6.18.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getAgentInfoResponse xmlns="urn:ModelManagement">
      <getAgentInfoReturn xmlns="">
        <agentAid xmlns="">
          <name xmlns="">AnalystAgent1@ARIPSAMAPlatform</name>
          <addresses xmlns="">
            <string xmlns="">http://CHAOS:7778/acc</string>
            <string xmlns="">http://CHAOS:49275/acc</string>
          </addresses>
        </agentAid>
        <startDate xmlns="">2012-02-26T09:00:04.785</startDate>
      </getAgentInfoReturn>
    </getAgentInfoResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 6.18: Resultado devuelto por el servicio

Al realizar la invocación del servicio ModelManagement se realiza la comunicación entre el agente WSIGAgent y los agentes AnalystAgent y CataloguerAgent. Esta comunicación se realiza por medio de mensajes FIPA ACL con un tipo de acción específico.

Esta comunicación puede observarse en la Figura 6.19 donde el agente WSIGAgent envía un mensaje ACL tipo Request al Agente AnalystAgent, el cual responde con un mensaje ACL tipo INFORM. Esta comunicación se origina al invocar el servicio con la petición SOAP mostrada en la Figura 6.17.

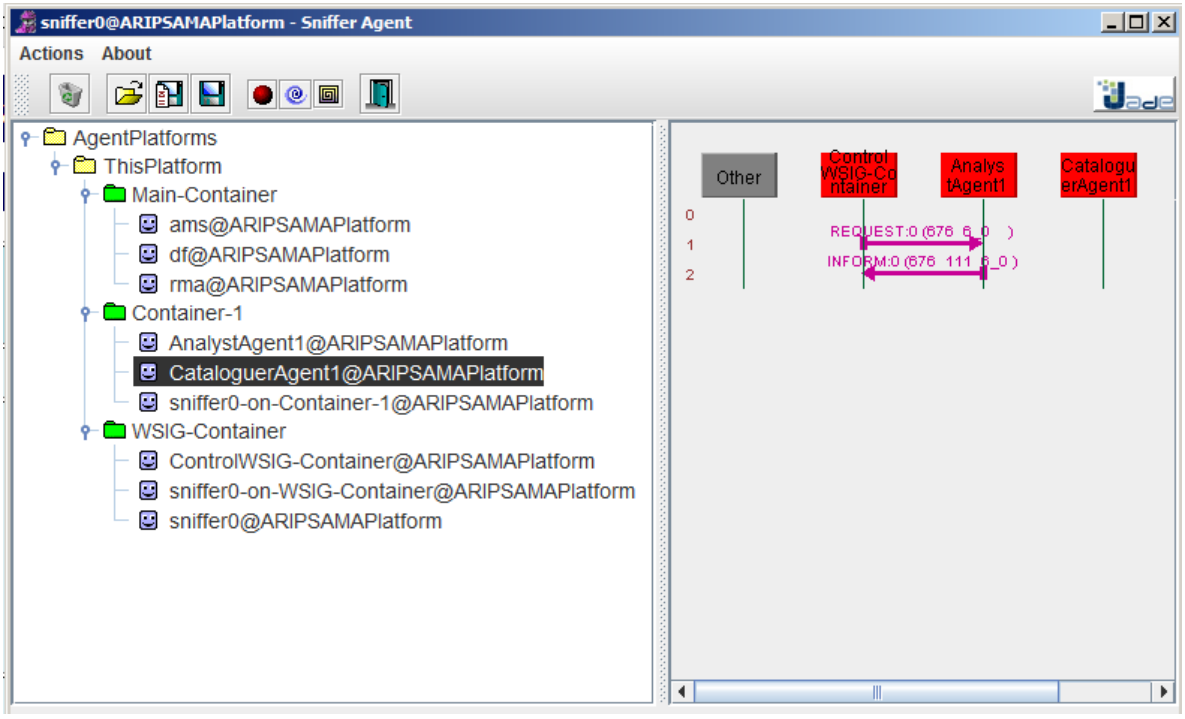


Figura 6.19: Plataforma Jade Sniffer - Comunicación entre Agentes

La plataforma Jade Sniffer mostrada en la Figura 6.19 forma parte de las herramientas ofrecidas por Jade, para el trabajo con agentes de software. Esta plataforma fue utilizada para pruebas de comunicación entre agentes de manera independiente.

6.5.2. Interfaz Web

La interfaz web consiste de una aplicación Web desarrollada con el framework JSF que permite la interacción del usuario con cada uno de los componentes de la arquitectura ARIPSAMA.

La Figura 6.2 define los flujos de carga y consulta de modelos a través de la interfaz web, atendiendo a ese flujo se presenta a continuación las diversas interfaces de usuario que satisfacen dicho flujo.

La Figura 6.20 muestra la pantalla inicial de la aplicación web ARIPSAMA.

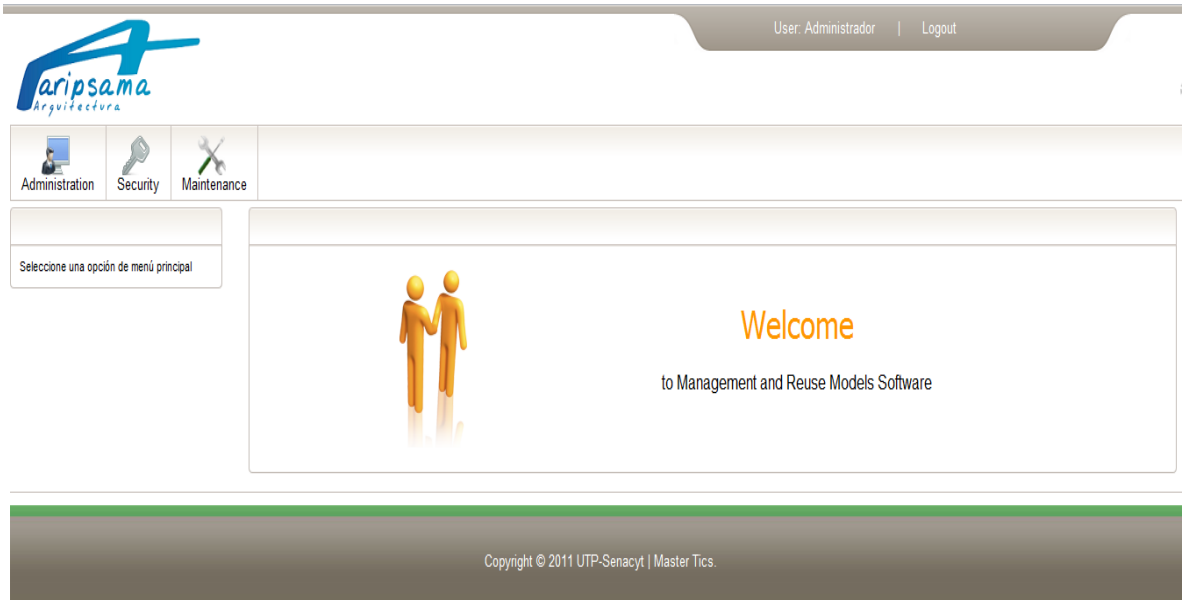


Figura 6.20: Pantalla de bienvenida aplicación web ARIPSAMA

La aplicación web permite el acceso a tres módulos distintos, el módulo de administración, permite la carga, descarga y ponderación de modelos. El módulo de seguridad permite el mantenimiento de roles, usuarios y permisos. El módulo de mantenimiento permite la carga y edición de datos primarios utilizados por la aplicación, así como la actualización de los modelos existentes.

Para la conversión UML-RelationalSchema se utiliza el modelo propuesto en la sección 6.2, basado en los elementos de la Tabla 6-1.

Esta conversión permite crear relaciones de equivalencia entre elementos (Actividades), transiciones, nodos de decisión y atributos de un diagrama de Actividades UML y elementos de un esquema Relacional. En la Figura 6.21 se muestra un diagrama de actividades de ejemplo que consiste en la conversión de un modelo XMI a un esquema XML..

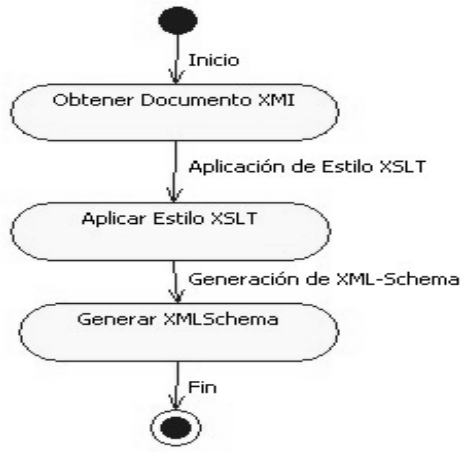


Figura 6.21: Diagrama de Actividad Ejemplo

Para la conversión de un modelo UML representado en formato XMI, como el que se muestra en la Figura 6.21 el agente analista hace uso del proceso de conversión UML-RelationalSchema descrito en la sección 6.2. Este proceso cuenta de dos clases principales, la clase parser y la clase switcher.

Un extracto del procedimiento encargado de realizar la conversión UML-XMLSchema específicamente la clase parser, la cual hace uso de la clase switcher se muestra en la Figura 6.22.

```
switch (switcher.elementSwitch(qName))
{
case XMI:
    model.setXmiVersion(attributes.getValue("xmi.version"));
    model.setNamespace(attributes.getValue("xmlns:UML"));
      break;
case EXPORTER:
    model.setExporter(attributes.getValue(0));
    break;
case EXPORTER_VERSION:
    model.setExporterVersion(attributes.getValue(0));
    break;
case MODEL:
    model.setName(attributes.getValue("name"));
    break;
case CONTROL_FLOW:
    tmpFlow=new ControlFlow();
    tmpFlow.setId(attributes.getValue("xmi.id"));
    tmpFlow.setName(attributes.getValue("name"));
    tmpFlow.setInputAction(attributes.getValue("source"));
    tmpFlow.setOutputAction(attributes.getValue("target"));
    break;
case ACTION_STATE:
    tmpAction = new Action();
    tmpAction.setId(attributes.getValue("xmi.id"));
    tmpAction.setName(attributes.getValue("name"));
    break;
case NODE:
    tmpNode=new Node();
    tmpNode.setId(attributes.getValue("xmi.id"));
    tmpNode.setName(attributes.getValue("name"));
```

Figura 6.22: Extracto del Procedimiento de Mapeo UML-RelationalSchema

El archivo XML (XMI) que representa el modelo presentado en la Figura 6.21 es cargado al repositorio por medio de la pantalla de carga de modelos mostrada en la Figura 6.23. El modelo procesado es almacenado en el repositorio y está disponible para ser consultado.

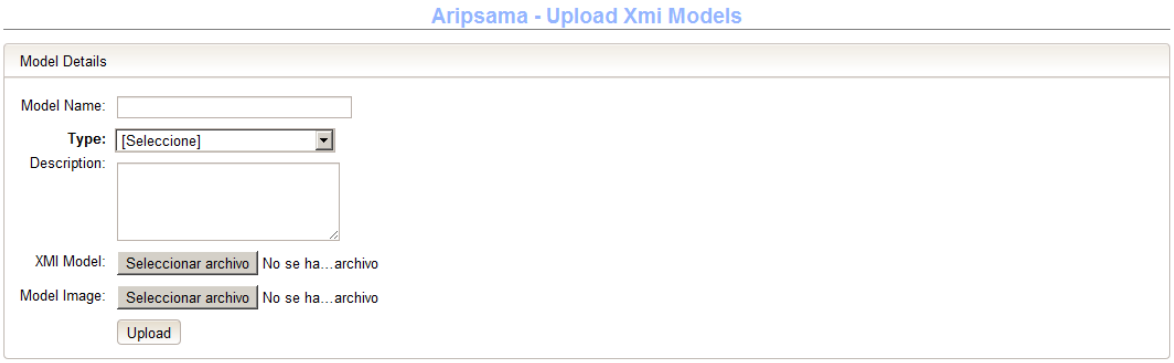


Figura 6.23: Pantalla de carga de modelos

Una vez catalogado el modelo el mismo puede ser consultado por medio de la pantalla de búsqueda. Para ello se definió un proceso de búsqueda que permite buscar por nombre del modelo o por una narración descriptiva. También permite restringir los resultados por medio de la selección del tipo de modelo que define el dominio del negocio al que pertenece el mismo.

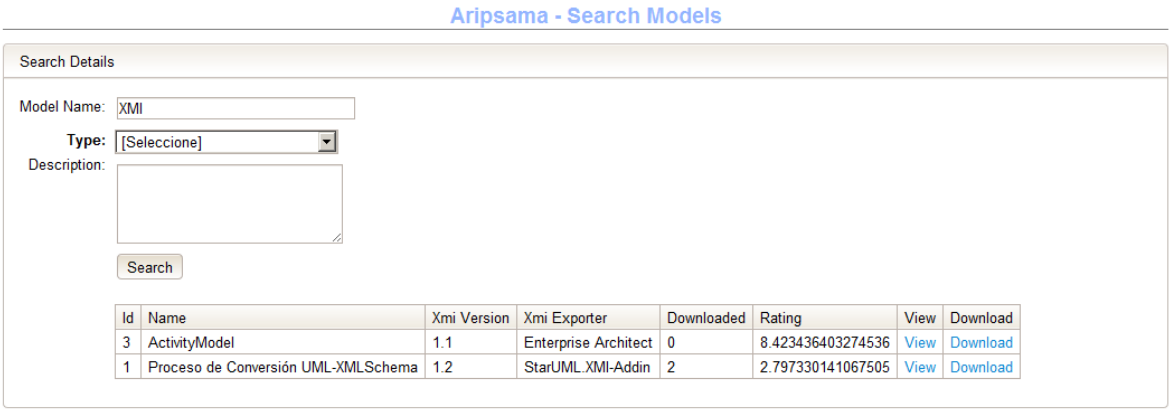


Figura 6.24: Resultados de Búsqueda del modelo UML mostrado en la Figura 6.21.

La búsqueda del modelo se realiza en lenguaje natural, para ello se hace uso de método de búsqueda FULL_TEXT definido en bases de datos como Oracle y MySQL, entre otras. También se aplica el método de ponderación descrito en la sección 6.3.1.

La Figura 6.24 muestra la pantalla de búsqueda de modelos de la aplicación Web ARIPSAMA.

Para todos los modelos listados como resultados de la consulta es posible ver atributos de ponderación y cantidad de descargas, a la vez que se muestra los detalles del modelo, los cuales incluyen una vista previa del mismo.

La figura Figura 6.24 también muestra la herramienta utilizada para modelar y exportar el diagrama cargado al repositorio. La vista previa y los detalles del modelo consultado se observa en la Figura 6.25.

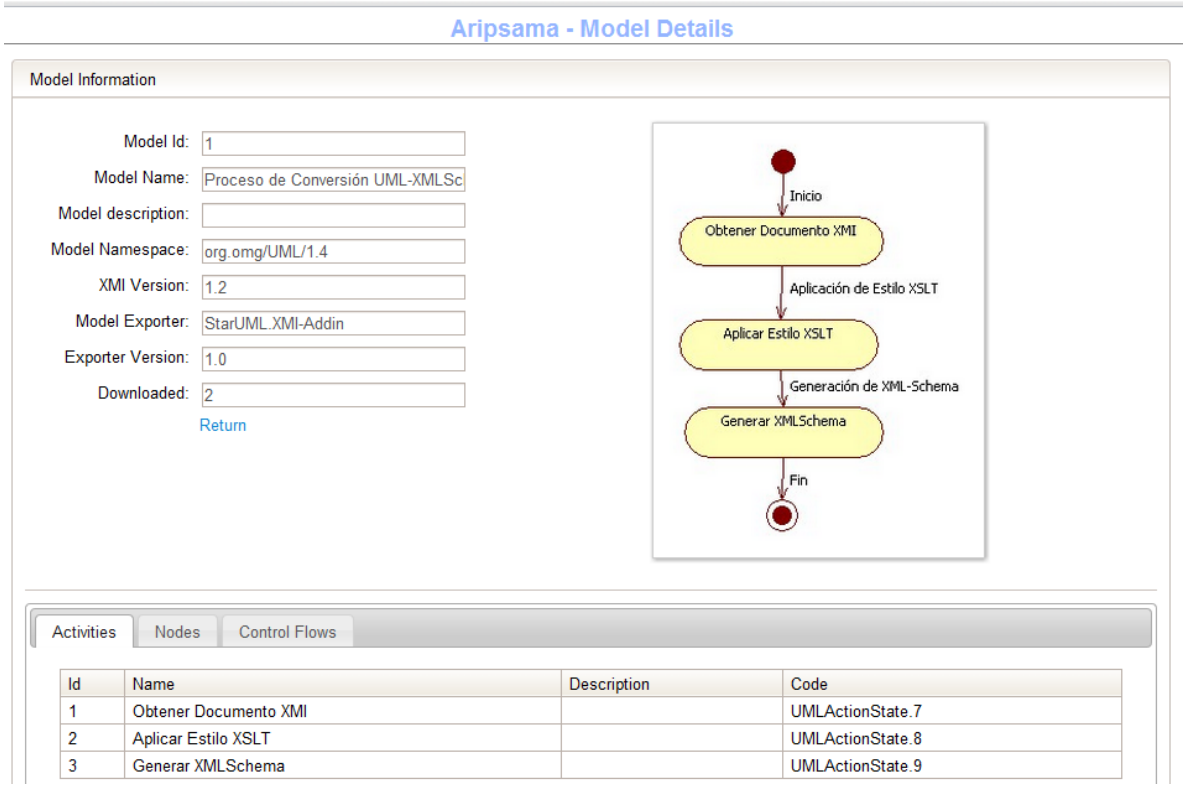


Figura 6.25: Detalles de un modelo buscado

Los detalles listados en la Figura 6.25 permiten observar los detalles del modelo, las actividades o tareas, nodos, relaciones, herramientas de modelado, cantidad de descargas, vista previa del modelo, entre otras.

Los modelos UML fueron probados con las herramientas UML: StartUML 5.0, EnterpriseArchitech 7.5, ArgoUML3.0, JDeveloper 11.1.1.5 y 11.1.2.1.

7. Conclusiones

Al haber culminado el presente estudio, se pudo validar la factibilidad de la utilización de una herramienta que facilite el almacenado, búsqueda y obtención de modelos de procesos de información de software de manera automática y con un tiempo de respuesta mucho menor en comparativa a otras soluciones ofrecidas por algunos autores mostrados en los trabajos relacionados.

Esta diferencia consiste en el método de análisis y almacenamiento de los modelos, ya que la manera en que ARIPSAMA almacena los modelos le permite hacer uso de la potencia de consulta ofrecida por las bases de datos relacionales, además la mayor parte del análisis de un modelo se realiza en el momento de la carga del modelo al repositorio, lo cual lleva el procesamiento del modelo de forma unitaria es decir, solo se analiza un modelo a la vez. Estas diferencias tienen un gran impacto en el rendimiento de la aplicación, ya que los trabajos propuestos realizan la mayor parte del análisis al momento de la consulta, y almacenan los modelos como ontologías o documentos xml, si bien es cierto esta técnica puede brindar un grado ligeramente superior de precisión, el impacto en tiempo de respuesta puede ser desastroso incluso al tratar de manejar un número relativamente bajo de modelos.

La desventaja en tiempo de respuesta fue uno de los principales motivos por el cual ARIPSAMA definió una estrategia de análisis y almacenamiento distinto, ya que se cuenta con una interfaz web que puede ser consumida de forma distribuida a través de internet y cuyo tiempo de respuesta debía ser relativamente bajo. En contraste, para minimizar la perdida de precisión en el análisis y la búsqueda del modelo, se plantearon tres métodos de ponderación distintos, dos de los cuales utilizan como retro-alimentador a los usuarios del sistema.

El trabajo realizado tuvo un enfoque no intrusivo para su funcionamiento, es decir los modelos a reutilizar no requieren de ningún atributo adicional como estereotipos o anotaciones para que puedan ser procesados.

Las búsquedas sobre un esquema relacional son rápidas, lo que permite el manejo de gran cantidad de información, en un tiempo casi instantáneo, algo que no se consigue con otros enfoques como el uso de Ontologías, o la representación por grafos (Robinson & Woo, 2004).

La principal debilidad del enfoque planteado es basar las búsquedas en conceptos almacenados en el modelo y la no realización de inferencias sobre la estructura del mismo. Para solventar esta debilidad se utiliza la ponderación dada por el usuario y la referencia por cantidad de descargas del modelo. Además que la aplicación de usuario está desarrollada enteramente sobre un ambiente web, por lo que es posible su utilización en cualquier localización geográfica en la que se encuentre el usuario.

La importancia de los modelos propuestos radica en que la mayoría de los trabajos desarrollados con propósitos similares, se enfocan en la obtención de modelos, y la reutilización de lógicas a un nivel detallado y puntual, como lo son los diagramas de clases, casos de uso y funcionalidades específicas de un sistema, entre otros. Sin embargo, el trabajo propuesto se enfoca en niveles más genéricos, a nivel de la lógica del negocio modelada a través de diagramas de Actividades/Procesos, propiciando la reutilización del conocimiento a niveles más altos dentro de un proyecto de desarrollo de software.

Otro punto relevante es la exposición de los servicios de agentes como servicios web, ya que esto permite la interoperabilidad de la arquitectura, ya que los servicios pueden ser consumidos por cualquier otra aplicación sin configuraciones adicionales, salvo la implementación del cliente del servicio.

7.1. Trabajos Futuros

- El presente trabajo se espera ampliar para permitir el reúso de los siguientes artefactos UML: Casos de Uso, Diagramas de Clase, Diagramas de Secuencia.

- Se espera realizar una evaluación de la herramienta con artefactos desarrollados para uso real, para ello se espera realizar un acuerdo con la Universidad Tecnológica de Panamá o alguna empresa de la localidad, que provea los artefactos necesarios para las pruebas.

Solo queda decir, que la reutilización del conocimiento es una parte fundamental de la ingeniería de software, ya que ayuda a reducir el tiempo de desarrollo y los costos asociados al mismo. Es por ello que con nuestro estudio esperamos realizar un pequeño aporte al campo de la Ingeniería de Software y la reutilización del software enfocándose en aspectos más estables como lo son la lógica y los procesos de software, y no en elementos dependientes de plataforma o lenguajes de programación tal como lo son, las librerías y módulos programáticos.

Referencias Bibliográficas

Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Developing Multi-agent systems With Jade.* Telecom Italia: John Wiley & Sons, Ltd.

Blok, M., & Cybulski, J. (1998). Reusing UML Especifications in Constrained Application Domain. *Software Engineering Conference*.

Booch, G., Jacobson, I., & Rumbaugh, J. (1997). *Unified Modeling Language.* Rational Software Corporation.

Cai, S., Zou, Y., Wang, L., Xie, B., & Shao, W. (2011). A Semi-supervised Approach for Component Recommendation Based on Citations. *12th International Conference on Software Reuse* (págs. 78-86). Pohang, South Korea: Springer Heidelberg Dordrecht London NewYork.

Cheesman, J., & Daniels, J. (2001). *UML Components.* Boston-San Francisco: Addison-Wesley.

Dall'oglio, P. (2010). *Um Sistema Multiagente Colaborativo para Gestão da Mudança de Requisitos de Software.* Brasil: Ciências Exatas e Tecnológicas da Unisinos.

Díaz, R. (2002). *Reutilización De Requisitos Funcionales De Sistemas Distribuidos Utilizando Técnicas De Descripción Formal.* Vigo, España.

Gomaa, H. (2011). *Software Modeling and Design.* Virginia: Uambridge University Press.

Graig, L. (2003). *UML y Patrones.* Pearson, Prentice Hall.

Heffelfinger, D. (2010). *Java EE 6 with GlassFish 3 Application Server.* Packt Publishing.

Henninger, S. (1998). An Environment for Reusing Software Processes. *International Conference on Software Reuse*, 103-112.

IBM. (24 de Agosto de 2011). *Working XML: UML, XMI, and code generation.* Obtenido de www.ibm.com

JADE Board. (2011). *Jade Web Services Integration Gateway Guide.* Italia: Telecom.

James, T. (2010). *Drupal Web Services.* Birmingham, B27 6PA, UK.: Packt Publishing Ltd.

Jørgensen, H. (2000). *Software Process Model Reuse and Learning.* Norway: Norwegian University of Science and Technology.

Kohlbacher, M., Gruenwald, S., & Kreuzer, E. (2010). Corporate Culture in Line with Business Process Orientation and Its Impact on Organizational Performance. *BPM 2010 InternationalWorkshops and Education Track* (págs. 16-24). Hoboken, NJ, USA: Springer Heidelberg Dordrecht London NewYork.

Krutchten, P. (1996). *4+1 Views for Software Architecture.* Rational Software Corporation.

Larentis, A. (2007). *Uma Arquitetura para Geração de Serviços a partir de Sistemas Legados de Forma Não Intrusiva.* São Leopoldo, Brasil: Eliete Mari Doncato Brasil - CRB 10/1184.

Leff, A., & Rayfield, J. (2001). Web-Application Development Using the ModelNiewlController Design Pattern. *IEEE 0-7695-1345-X*, 118-127.

Lorenz, D., & Rosenan, B. (2011). Code Reuse with Language Oriented Programming. *12th International Conference on Software Reuse* (págs. 167-181). Pohang, South Korea: Springer Heidelberg Dordrecht London NewYork.

Macek, O., & Richta, K. (2009). *The BPM to UML activity diagram transformation using XSLT.*

Mahmoud, Q., & Maamar, Z. (2006). Applying the MVC Design Pattern to Multi-Agent Systems. *IEEE*, 2420-2423.

Martins, R. (2007). *Composição dinâmica de Web Services.* São Leopoldo, Brasil: Bibliotecária Eliete Mari Doncato Brasil - CRB 10/1184.

Mills, D., Koletzke, P., & Roy-Faderman, A. (2010). *Oracle JDeveloper 11g Handbook.* McGraw-Hill.

Mun-Young, C., Lim, J., & Kyung-Soo, J. (2005). Developing An XML Schema Generator Based On UML Class. *IEEE*, 2336-2340.

Narayanan, K., & Ramaswamy, S. (2005). Specifications for Mapping UML Models to XML Schemas. *Proceedings of the 4th Workshop in Software Model Engineering.* Jamaica.

OMG. (2011). *Object Management Group*. Recuperado el 2011, de OMG Unified Modeling Language Specification: www.omg.org

Rahim, B. (2003). *A Software Reuse Paradigm for the Next Generation Network (NGN).* Johannesburg.

Robinson, W., & Woo, H. (2004). Finding Reusable UML Sequence Diagrams Automatically. *IEEE SOFTWARE*, 60-67.

Routledge, N., Bird, L., & Goodchild, A. (2002). UML and XML Schema.

Silva, J. (2010). *Um Sistema Multiagente Baseado Em Ontologias Para Apoio Às Inspeções De Garantia Da Qualidade De Software.* São Leopoldo, Brasil: Unisinos.

Su, M., Shyang Wong, C., & Fen Soo, C. (2007). Service-Oriented E-Learning System. *IEEE*, 6-11.

Talib, R., Volz, B., & Jablonski, S. (2010). Agent Assignment for Process Management: Goal Modeling for Continuous Resource Management. *BPM 2010 InternationalWorkshops and Education Track* (págs. 35-36). Hoboken, NJ, USA: Springer Heidelberg Dordrecht London NewYork.

The Reuse Company. (2006). Presente y Futuro de la Reutilización de Software.

Thomas, D. (2009). *Apache MyFaces Trinidad 1.2 a Practical Guide.* Birmingham: Packt Publishing.

Tonconi, L. (2009). *Comparativa de Metodologías Common-Kads e INGENIAS en el Desarrollo de Sistemas Multi-Agentes.* Bolivia.

W3C. (2011). *Web Services architecture: W3C working group note.* Recuperado el 2011 de Noviembre de 2011, de World Wide Web Consortium: http://www.w3.org/TR/ws-arch

W3C. (3 de Octubre de 2011). *XML Core.* Obtenido de http://www.w3.org/standards/xml/core

Widenius, M., & Axmark , D. (9 de 8 de 2011). *MySQL 5.5 Reference Manual.* Obtenido de http://dev.mysql.com/doc/refman/5.5/en/introduction.html