



# **TRANSFORMACIÓN UML-XML-BDOR-UML**

Ing. Janmarco Rojas Nava  
Tutor: Isabel Besembel Carrera

COMO REQUISITO PARA OBTENER  
EL GRADO DE  
MAGISTER SCIENTIAE EN COMPUTACIÓN  
DE LA  
UNIVERSIDAD DE LOS ANDES  
MÉRIDA, VENEZUELA  
DICIEMBRE 2004



UNIVERSIDAD DE LOS ANDES  
FACULTAD DE INGENIERÍA  
POSTGRADO DE COMPUTACIÓN

El jurado aprueba el proyecto de grado titulado “**TRANSFORMACIÓN UML-XML-BDOR-UML**” realizado por el **Ing. Janmarco Rojas Nava** como requisito parcial para la obtención del grado de **Magister Scientiae en Computación**.

Fecha: Septiembre 2004

Tutor:

---

Isabel Besembel Carrera

Jurado:

---

Primer Jurado

---

Segundo Jurado



*A Dios Todopoderoso...*



# Índice general

<i>Índice de tablas</i>	<i>xv</i>
<i>Índice de figuras</i>	<i>xix</i>
<i>Agradecimientos</i>	<i>xxiii</i>
<i>Resumen</i>	<i>xxv</i>

## **Capítulo 1. Introducción ----- 1-1**

### **1.1 Antecedentes ----- 1-2**

1.1.1 Esfuerzos por establecer reglas que permiten migrar documentos XML a bases de datos y viceversa.	1-2
1.1.2 Implementación de herramientas que permiten convertir los datos contenidos en las bases de datos relacionales en documentos XML	1-2
1.1.3 Transformaciones basadas en tablas y transformaciones basadas en objetos	1-3
1.1.4 Uso del lenguaje de programación Java para importar documentos XML a bases de datos relacionales	1-4
1.1.5 Integración de XML a las nuevas tecnologías de bases de datos relacionales	1-4

### **1.2 Áreas Relacionadas ----- 1-4**

1.2.1 XML (eXtensible Markup Language)	1-5
1.2.2 DTD (Document Type Definition)	1-6
1.2.3 Tecnologías XML	1-8
Esquemas XML	1-8
DOM (Document Object Model)	1-8
SAX (Simple API for XML)	1-8
XSLT/XPath	1-9

XML Query -----	1-9
XLink -----	1-9
<b>1.3 UML -----</b>	<b>1-10</b>
1.3.1 Diagramas de clases-----	1-10
Las clases -----	1-10
Las asociaciones -----	1-11
Las clases-asociaciones -----	1-12
Las agregaciones-----	1-13
La composición -----	1-13
La navegación-----	1-14
La generalización -----	1-14
Las clases abstractas-----	1-15
<b>1.4 Bases de Datos Relacionales-----</b>	<b>1-15</b>
1.4.1 Diseño Lógico-----	1-16
Entidades-----	1-16
Atributos -----	1-16
Relaciones -----	1-16
1.4.2 Diseño Físico-----	1-16
Tablas -----	1-17
Claves Principales -----	1-17
Claves Foráneas-----	1-17
Índices-----	1-17
Disparadores-----	1-17
Procedimientos Almacenados -----	1-18
<b>1.5 Bases de Datos Objeto-Relacionales-----</b>	<b>1-18</b>
1.5.1 El Modelo Relacional Anidado -----	1-18
1.5.2 Los Tipos Complejos y la Herencia-----	1-19
<b>1.6 Justificación -----</b>	<b>1-19</b>
<b>1.7 Objetivo -----</b>	<b>1-21</b>
<b>1.8 Métodos de desarrollo -----</b>	<b>1-22</b>
<b>1.9 Alcance -----</b>	<b>1-24</b>
<b>1.10 Organización del manuscrito-----</b>	<b>1-24</b>



**Capítulo 2. Revisión Bibliográfica----- 2-1**

<b>2.1</b>	<b>DOM -----</b>	<b>2-1</b>
2.1.1	Arquitectura DOM -----	2-2
2.1.2	Interfaces e Implementaciones-----	2-3
2.1.3	DOM Core-----	2-3
	Tipos básicos definidos por la especificación DOM Core-----	2-4
	Interfaces fundamentales del módulo DOM Core -----	2-4
	Interfaz DOMImplementation -----	2-7
	Interfaz Node-----	2-7
	Interfaz Document -----	2-8
	Interfaz Attr -----	2-9
	Interfaz Element -----	2-10
2.1.4	DOM Level 3 Load and Save -----	2-11
	Tipos Básicos definidos en DOM Level 3 Load and Save -----	2-11
	Interfaces fundamentales de DOM Level 3 Load and Save -----	2-12
	Interfaz DOMImplementationLS -----	2-12
	Interfaz LSParser -----	2-13
	Interfaz LSInput-----	2-14
	Interfaz LSOutput-----	2-15
	Interfaz LSParserFilter -----	2-15
	Interfaz LSProgressEvent-----	2-15
	Interfaz LSLoadEvent-----	2-16
	Interfaz LSSerializer -----	2-16
	Interfaz LSSerializerFilter -----	2-17
<b>2.2</b>	<b>Esquemas XML-----</b>	<b>2-17</b>
2.2.1	Documento esquema XML-----	2-18
2.2.2	Modelo de Datos de un esquema XML-----	2-18
2.2.3	Tipos de Datos -----	2-20
	Espacio de Valores -----	2-22
	Espacio Léxico-----	2-23
2.2.4	Propiedades de los Tipos de Datos -----	2-23
	Propiedades fundamentales-----	2-23
	Propiedades no fundamentales o restricciones-----	2-24
2.2.5	Definiciones de tipos simples -----	2-25

Derivación por restricción -----	2-26
Derivación por lista-----	2-27
Derivación por unión-----	2-28
2.2.6 Definiciones de tipos complejos-----	2-29
Modelos de Contenido Simple-----	2-30
Modelos de Contenido Complejo-----	2-30
2.2.7 Declaración de Atributos -----	2-31
2.2.8 Declaración de Elementos-----	2-32
<b>2.3 Esquemas BDOR -----</b>	<b>2-34</b>
2.3.1 SQL (Structured Query Language)-----	2-34
Sentencias-----	2-35
Tipos de Datos -----	2-37
2.3.2 Lenguaje de Definición de Datos (LDD)-----	2-38
CREATE TABLE -----	2-39
DROP TABLE-----	2-40
ALTER TABLE -----	2-41
CREATE VIEW -----	2-42
DROP VIEW-----	2-42
CREATE INDEX-----	2-42
DROP INDEX -----	2-43
CREATE SCHEMA-----	2-43
 <b>Capítulo 3. Transformación UML-XML-BDOR-UML -----</b>	<b>3-1</b>
<b>3.1 Usando diagramas de clases UML para generar esquemas XML -----</b>	<b>3-2</b>
3.1.1 Representación de los atributos-----	3-4
3.1.2 Representación de las clases -----	3-5
3.1.3 Representación de las asociaciones -----	3-9
Representación de las asociaciones a través de contenido-----	3-9
Representación de las asociaciones a través de declaraciones de elementos-----	3-11
Representación de las clases-asociación -----	3-14
3.1.4 Declaración del elemento raíz del documento instancia-----	3-17
3.1.5 Representación gráfica de la vista del documento instancia -----	3-17
3.1.6 Reglas para construir un documento esquema XML	

a partir de un diagrama de clases UML .....	3-19
3.1.7 Resultados de la transformación UML → XML .....	3-26
<b>3.2 Usando esquemas XML para construir esquemas BDOR .....</b>	<b>3-27</b>
3.2.1 Representación de las declaraciones de atributos.....	3-27
3.2.2 Representación de las definiciones de tipos complejos .....	3-29
3.2.3 Representación de las declaraciones de elementos .....	3-30
3.2.4 Reglas para construir un esquema BDOR a partir de un documento esquema XML .....	3-32
3.2.5 Resultados de la transformación XML→BDOR .....	3-36
<b>3.3 Usando esquemas BDOR para construir diagramas de clases UML .....</b>	<b>3-37</b>
3.3.1 Representación de las tablas que poseen claves primarias .....	3-37
3.3.2 Representación de las tablas que poseen claves foráneas.....	3-38
3.3.3 Reglas para construir un diagrama UML a partir de un esquema BDOR .....	3-39
3.3.4 Resultados de la transformación BDOR→UML .....	3-41
<b>Capítulo 4. Diseño del prototipo.....</b>	<b>4-1</b>
<b>4.1 Análisis del dominio de la aplicación.....</b>	<b>4-1</b>
4.1.1 Definición del alcance del dominio .....	4-1
4.1.2 Modelo Conceptual.....	4-2
4.1.3 Definición del vocabulario del dominio .....	4-3
<b>4.2 Definición de requerimientos .....</b>	<b>4-4</b>
4.2.1 Funciones básicas.....	4-4
4.2.2 Atributos del sistema.....	4-6
<b>4.3 Análisis y especificación de requerimientos.....</b>	<b>4-6</b>
4.3.1 Casos de uso.....	4-6
4.3.2 Diagramas de interacción.....	4-9
Diagrama de colaboración: nuevoDiagramaUML.....	4-9
Diagrama de colaboración: abrir.....	4-10
Diagrama de colaboración: guardarDiagramaUML.....	4-11
Diagrama de colaboración: guardarDiagramaUMLcomo .....	4-12
Diagrama de colaboración: imprimirDiagramaUML .....	4-12
Diagrama de colaboración: crearClaseUML .....	4-13
Diagrama de colaboración: modificarClaseUML .....	4-14

Diagrama de colaboración: eliminarClaseUML -----	4-14
Diagrama de colaboración: crearAsociacionUML -----	4-15
Diagrama de colaboración: modificarAsociaciónUML -----	4-16
Diagrama de colaboración: eliminarAsociaciónUML -----	4-17
Diagrama de colaboración: crearVista -----	4-18
Diagrama de colaboración: destruirVista -----	4-18
Diagrama de colaboración: crearEsquemaXMLdesdeDiagramaUML -----	4-19
Diagrama de colaboración: crearEsquemaBDORdesdeEsquemaXML -----	4-20
Diagrama de colaboración: crearDiagramaUMLdesdeCatalogoBDOR -----	4-20
4.3.3 Diagramas de actividades -----	4-21
4.3.4 Diagrama de clases -----	4-23
<b>4.4 Diseño del sistema -----</b>	<b>4-24</b>
4.4.1 Diseño de la interfaz de usuario -----	4-25
Barra de Menú -----	4-26
Barra de herramientas estándar -----	4-26
Barra de herramientas UML -----	4-27
Estilos para el diseño de las ventanas -----	4-27
Mensajes -----	4-28
4.4.2 Diseño de la arquitectura -----	4-29
 <b>Capítulo 5. Implementación -----</b>	 <b>5-1</b>
<b>5.1 Herramientas de desarrollo -----</b>	<b>5-1</b>
<b>5.2 Implementación del prototipo -----</b>	<b>5-2</b>
<b>5.3 Arquitectura de despliegue -----</b>	<b>5-7</b>
<b>5.4 Prototipo -----</b>	<b>5-8</b>
5.4.1 Entorno de la aplicación -----	5-8
Barra de menú -----	5-9
Barra de herramientas estándar -----	5-10
Barra de herramientas UML -----	5-10
5.4.2 Construcción de diagramas de clases -----	5-11
Creación, Modificación y Eliminación de una clase -----	5-12
Creación, Modificación y Eliminación de asociaciones -----	5-13

5.4.3	Creación de esquemas XML y esquemas BDOR-----	5-15
5.4.4	Creación de un diagrama de clases a partir del catálogo de una BDOR-----	5-18
<b>Capítulo 6. Conclusiones y Recomendaciones-----</b>		<b>6-1</b>
<b>6.1</b>	<b>Conclusiones-----</b>	<b>6-1</b>
<b>6.2</b>	<b>Recomendaciones-----</b>	<b>6-4</b>
<b>Bibliografía-----</b>		<b>7-1</b>



# Índice de tablas

<i>Tabla 1-1. Valores de cardinalidad convencionales</i> -----	1-12
<i>Tabla 2-1. Módulos definidos por la especificación DOM Level 3</i> -----	2-2
<i>Tabla 2-2. Tipos básicos definidos por la especificación DOM Core</i> -----	2-4
<i>Tabla 2-3. Interfaces fundamentales definidas por DOM Core</i> -----	2-4
<i>Tabla 2-4. Tipos Básicos definidos en DOM Level 3 Load and Save</i> -----	2-11
<i>Tabla 2-5. Interfaces definidas por DOM Level 3 Load and Save</i> -----	2-12
<i>Tabla 2-6. Propiedades fundamentales</i> -----	2-24
<i>Tabla 2-7. Propiedades no fundamentales o restricciones</i> -----	2-24
<i>Tabla 2-8. Sintaxis del elemento simpleType</i> -----	2-26
<i>Tabla 2-9. Sintaxis del elemento restriction</i> -----	2-27
<i>Tabla 2-10. Sintaxis del elemento list</i> -----	2-27
<i>Tabla 2-11. Sintaxis del elemento union</i> -----	2-28
<i>Tabla 2-12. Sintaxis del elemento complexType</i> -----	2-29
<i>Tabla 2-13. Sintaxis del elemento simpleContent</i> -----	2-30
<i>Tabla 2-14. Sintaxis del elemento complexContent</i> -----	2-31
<i>Tabla 2-15. Sintaxis del elemento attribute</i> -----	2-31
<i>Tabla 2-16. Sintaxis del elemento element</i> -----	2-32
<i>Tabla 2-17. Principales Sentencias SQL [Groff y Weingberg 1999]</i> -----	2-35
<i>Tabla 2-18. Tipos de Datos ANSI/ISO SQL92 [Groff y Weingberg 1999]</i> -----	2-37
<i>Tabla 2-19. Sintaxis de la sentencia CREATE TABLE.</i> -----	2-39
<i>Tabla 2-20. Sintaxis de la sentencia DROP TABLE.</i> -----	2-40
<i>Tabla 2-21. Sintaxis de la sentencia ALTER TABLE.</i> -----	2-41
<i>Tabla 2-22. Sintaxis de la sentencia CREATE VIEW.</i> -----	2-42
<i>Tabla 2-23. Sintaxis de la sentencia DROP VIEW.</i> -----	2-42
<i>Tabla 2-24. Sintaxis de la sentencia CREATE INDEX usada por MySQL</i> -----	2-43
<i>Tabla 2-25. Sintaxis de la sentencia DROP INDEX usada por MySQL</i> -----	2-43

Tabla 2-26. Sintaxis de la sentencia <code>CREATE SCHEMA</code> .	2-44
Tabla 3-1. Ejemplo de representación de clases UML en esquemas XML.	3-8
Tabla 3-2. Tipos de datos XML que son compatibles con los tipos de datos de los SGBDOR.	3-28
Tabla 3-3. Ejemplos de transformación de declaraciones de atributos XML en cláusulas SQL para MySQL.	3-28
Tabla 4-1. Asociaciones entre conceptos.	4-3
Tabla 4-2. Diccionario de términos del dominio.	4-4
Tabla 4-3. Funciones que debe ofrecer la aplicación	4-5
Tabla 4-2. Funciones relacionadas con la creación de diagramas UML	4-5
Tabla 4-4. Atributos del sistema	4-6
Tabla 4-5. Caso de uso: Crear diagrama UML	4-7
Tabla 4-6. Caso de uso: Crear esquema XML desde diagrama UML	4-8
Tabla 4-7. Caso de uso: Crear esquema BDOR desde esquema XML	4-8
Tabla 4-8. Caso de uso: Crear diagrama UML desde esquema BDOR	4-9
Tabla 4-9. Contrato de uso: <code>nuevoDiagramaUML</code>	4-10
Tabla 4-10. Contrato de uso: <code>abrir</code>	4-10
Tabla 4-11. Contrato de uso: <code>guardarDiagramaUML</code>	4-11
Tabla 4-12. Contrato de uso: <code>guardarDiagramaUMLcomo</code>	4-12
Tabla 4-13. Contrato de uso: <code>imprimirDiagramaUML</code>	4-13
Tabla 4-14. Contrato de uso: <code>crearClaseUML</code>	4-13
Tabla 4-15. Contrato de uso: <code>modificarClaseUML</code>	4-14
Tabla 4-16. Contrato de uso: <code>eliminarClaseUML</code>	4-15
Tabla 4-17. Contrato de uso: <code>crearAsociacionUML</code>	4-16
Tabla 4-18. Contrato de uso: <code>modificarAsociacionUML</code>	4-16
Tabla 4-19. Contrato de uso: <code>eliminarAsociacionUML</code>	4-17
Tabla 4-20. Contrato de uso: <code>crearVista</code>	4-18
Tabla 4-21. Contrato de uso: <code>destruirVista</code>	4-19
Tabla 4-22. Contrato de uso: <code>crearEsquemaXMLdesdeDiagramaUML</code>	4-19
Tabla 4-23. Contrato de uso: <code>crearEsquemaBDORdesdeEsquemaXML</code>	4-20
Tabla 4-24. Contrato de uso: <code>crearDiagramaUMLdesdeCatalogoBDOR</code>	4-21
Tabla 4-25. Modelo de íconos para la barra de herramientas UML.	4-27
Tabla 5-1. Herramientas utilizadas para el desarrollo de la aplicación	5-1
Tabla 5-2. Clases que conforman el paquete <code>trilogia.gui</code>	5-3
Tabla 5-3. Clases que conforman el paquete <code>trilogia.gui.util</code>	5-4
Tabla 5-4. Clases que conforman el paquete <code>trilogia.uml</code>	5-4



<i>Tabla 5-5. Clases que conforman el paquete trilogia.xs</i> .....	5-5
<i>Tabla 5-6. Clases que conforman el paquete trilogia.mdr</i> .....	5-6
<i>Tabla 6-1. Ejemplo de transformación de definición de tipo complejo XML en una sentencia de creación de tipo usando el estándar SQL-3 de ORACLE.</i> .....	6-4



# Índice de figuras

<i>Figura 1-1. Representación gráfica de una clase.</i>	1-10
<i>Figura 1-2. Representación gráfica de una asociación entre dos clases.</i>	1-11
<i>Figura 1-3. Representación de los roles y valores de cardinalidad.</i>	1-12
<i>Figura 1-4. Ejemplo de clase-asociación.</i>	1-12
<i>Figura 1-5. Representación de las agregaciones.</i>	1-13
<i>Figura 1-6. Representación gráfica de la composición.</i>	1-14
<i>Figura 1-7. Representación de una asociación navegable.</i>	1-14
<i>Figura 1-8. Representación de la generalización.</i>	1-15
<i>Figura 1-9. Representación de una clase abstracta.</i>	1-15
<i>Figura 1-10. El Modelo de Procesos Watch.</i>	1-23
<i>Figura 2-1. Arquitectura DOM Level 3</i>	2-3
<i>Figura 2-2. Interfaces del módulo DOM Core</i>	2-6
<i>Figura 2-3. Interfaz DOMImplementation</i>	2-7
<i>Figura 2-4. Interfaz Node</i>	2-7
<i>Figura 2-5. Interfaz Document</i>	2-8
<i>Figura 2-6. Interfaz Attr</i>	2-9
<i>Figura 2-7. Interfaz Element</i>	2-10
<i>Figura 2-8. Interfaz DOMImplementationLS</i>	2-12
<i>Figura 2-9. Interfaz LSParser</i>	2-13
<i>Figura 2-10. Interfaz LSInput</i>	2-14
<i>Figura 2-11. Interfaz LSOutput</i>	2-15
<i>Figura 2-12. Interfaz LSParserFilter</i>	2-15
<i>Figura 2-13. Interfaz LSProgressEvent</i>	2-16
<i>Figura 2-14. Interfaz LSLoadEvent</i>	2-16
<i>Figura 2-15. Interfaz LSSerializer</i>	2-17
<i>Figura 2-16. Interfaz LSSerializerFilter</i>	2-17

<i>Figura 2-17. Modelo de Datos de un esquema XML</i>	2-19
<i>Figura 2-18. Tipos de datos definidos por la especificación Esquema XML</i>	2-22
<i>Figura 3-1. Diagrama de Clases</i>	3-3
<i>Figura 3-3. Asociación simple entre las clases Departamento y Proyecto.</i>	3-9
<i>Figura 3-4. Asociaciones múltiples entre las clases Departamento y Empleado.</i>	3-12
<i>Figura 3-5. Clase-Asociación Tiene.</i>	3-14
<i>Figura 3-6. Representación gráfica de una vista XML.</i>	3-18
<i>Figura 3-7. Clases generadas a partir de las tablas Departamento y Proyecto del esquema BDOR.</i>	3-38
<i>Figura 3-8. Asociación generada a partir de la tabla desarrolla del esquema BDOR.</i>	3-39
<i>Figura 3-9. Clases creadas a partir del esquema BDOR.</i>	3-40
<i>Figura 3-10. Asociaciones creadas a partir del esquema BDOR</i>	3-41
<i>Figura 4-1. Modelo de entidades de negocio.</i>	4-2
<i>Figura 4-2. Diagrama de casos de uso.</i>	4-7
<i>Figura 4-3. Diagrama de Colaboración: nuevoDiagramaUML</i>	4-9
<i>Figura 4-4. Diagrama de Colaboración: abrir</i>	4-10
<i>Figura 4-5. Diagrama de Colaboración: guardarDiagramaUML</i>	4-11
<i>Figura 4-6. Diagrama de Colaboración: guardarDiagramaUMLcomo</i>	4-12
<i>Figura 4-7. Diagrama de Colaboración: imprimirDiagramaUML</i>	4-13
<i>Figura 4-8. Diagrama de Colaboración: crearClaseUML</i>	4-13
<i>Figura 4-9. Diagrama de Colaboración: modificarClaseUML</i>	4-14
<i>Figura 4-10. Diagrama de Colaboración: eliminarClaseUML</i>	4-15
<i>Figura 4-11. Diagrama de Colaboración: crearAsociacionUML</i>	4-15
<i>Figura 4-12. Diagrama de Colaboración: modificarAsociacionUML.</i>	4-16
<i>Figura 4-13. Diagrama de Colaboración: eliminarAsociacionUML</i>	4-17
<i>Figura 4-14. Diagrama de Colaboración: crearVista</i>	4-18
<i>Figura 4-15. Diagrama de Colaboración: destruirVista</i>	4-18
<i>Figura 4-16. Diagrama de Colaboración: crearEsquemaXMLdesdeDiagramaUML</i>	4-19
<i>Figura 4-17. Diagrama de Colaboración: crearEsquemaBDORdesdeEsquemaXML</i>	4-20
<i>Figura 4-18. Diagrama de Colaboración: crearDiagramaUMLdesdeCatalogoBDOR</i>	4-21
<i>Figura 4-19. Diagrama de Actividades: Crear esquema XML desde diagrama UML</i>	4-22
<i>Figura 4-19. Diagrama de Actividades: Crear esquema BDOR desde esquema XML.</i>	4-22
<i>Figura 4-20. Diagrama de Actividades: Crear diagrama UML desde catalogo BDOR.</i>	4-23
<i>Figura 4-21. Diagrama de clases de negocio con atributos y operaciones.</i>	4-24
<i>Figura 4-22. Características gráficas de la interfaz gráfica de usuario.</i>	4-25
<i>Figura 4-23. Modelo de menú desplegable.</i>	4-26

<i>Figura 4-24. Modelo de barra de herramientas.</i>	4-26
<i>Figura 4-25. Modelo de cuadro de diálogo.</i>	4-28
<i>Figura 4-26. Modelo de mensaje de información.</i>	4-28
<i>Figura 4-27. Modelo de mensaje de advertencia.</i>	4-28
<i>Figura 4-28. Modelo de mensaje de interrogación.</i>	4-29
<i>Figura 4-29. Modelo de mensaje de error.</i>	4-29
<i>Figura 4-30. Capas de la arquitectura representadas en paquetes UML</i>	4-30
<i>Figura 5-1. Paquetes que conforman la aplicación</i>	5-2
<i>Figura 5-2. Paquete trilogia.gui</i>	5-2
<i>Figura 5-3. Paquete trilogia.gui.util</i>	5-3
<i>Figura 5-4. Paquete trilogia.uml</i>	5-4
<i>Figura 5-5. Paquete trilogia.xs</i>	5-5
<i>Figura 5-6. Paquete trilogia.mdr</i>	5-6
<i>Figura 5-7 Arquitectura de despliegue.</i>	5-7
<i>Figura 5-8 Interfaz gráfica de la aplicación.</i>	5-8
<i>Figura 5-9. Barra de herramientas estándar.</i>	5-10
<i>Figura 5-10. Barra de herramientas UML.</i>	5-11
<i>Figura 5-11. Diagrama de clases vacío.</i>	5-11
<i>Figura 5-12. Nueva clase.</i>	5-12
<i>Figura 5-13. Cuadro de diálogo: Propiedades de la clase</i>	5-12
<i>Figura 5-14. Representación gráfica de una clase.</i>	5-13
<i>Figura 5-15. Representación gráfica de una asociación.</i>	5-14
<i>Figura 5-16. Cuadro de diálogo: Propiedades de asociación</i>	5-14
<i>Figura 5-17. Representación de una vista sobre el diagrama de clases.</i>	5-15
<i>Figura 5-18. Documento esquema XML creado por la aplicación.</i>	5-16
<i>Figura 5-19. Cuadro de diálogo: Crear esquema BDOR desde esquema XML.</i>	5-17
<i>Figura 5-20. Documento esquema BDOR creado por la aplicación.</i>	5-17
<i>Figura 5-21. Cuadro de diálogo: Crear diagrama UML desde catálogo BDOR.</i>	5-18
<i>Figura 5-22. Diagrama UML generado a partir del catálogo de la BDOR.</i>	5-18



# Agradecimientos

Al Postgrado de Computación de la Universidad de Los Andes, por abrirme sus puertas y a todas las personas que forman parte de este gran equipo, entre ellas: Luisa Díaz y Alexander Barrios, por ser amigos incondicionales y en especial al coordinador del postgrado Wladimir Rodríguez, a su hermana Taniana y a todos los que fueron mis profesores: Eitan Altman, Isabel Besembel, Jacinto Dávila, Leandro León, Jonás Montilva, Ramón Pino, Rafael Rivas y Wladimir Rodríguez.

A mi tutora, la profesora Isabel Besembel Carrera. Ella siempre fue el pilar fundamental para llevar a cabo la realización de esta tesis y más que una guía fue una mano amiga que siempre estuvo a mi lado. Trabajar a su lado es toda una experiencia. ¡Mil gracias profe!

A mis amigos: Damian<sup>\*</sup>, Dayana<sup>\*</sup>, Glenda, Lisdrelys, Adriana, Marisela, María Elena, Ramón, Chadi, Solazver y Víctor, por poner su grano de arena en el momento preciso.

Al FONACIT por financiar mis estudios del postgrado. Espero que este tipo de instituciones le siga brindando la oportunidad a muchas otros profesionales que quieren realizar sus estudios de postgrado en esta gran casa de estudios, la “Universidad de Los Andes”.





# Resumen

El gran auge que ha tenido la tecnología XML en los últimos años, han hecho que XML se haya convertido en el estándar universal para transmitir, intercambiar y manipular datos. Siendo las bases de datos objeto-relacionales (BDOR) un medio ideal para almacenar los datos contenidos en los documentos XML, y considerando además, que pueden ser utilizadas para compartir la gran cantidad de datos almacenados en éstas, produjo la necesidad de integrar estas dos grandes tecnologías. Este concepto condujo al desarrollo de un número significativo de trabajos enfocados en resolver el problema del intercambio de datos entre XML y las BDOR (XML↔BDOR), sin embargo, los primeros esfuerzos utilizaban definiciones de tipo de documento (DTDs) que no solventaron el problema.

Es por ello que este trabajo tiene como objetivo fundamental, hacer uso de la nueva recomendación Esquema XML, la cual promete resolver los problemas que se presentan al usar las antiguas DTDs y establecer un conjunto de reglas bien definidas que permitan obtener un conjunto de esquemas XML y esquemas BDOR, modelados a partir de diagramas de clases UML. El conjunto de esquemas resultante permite establecer las bases para realizar un intercambio bidireccional de datos entre documentos XML y las bases de datos objeto-relacionales (BDOR).

El resultado final de este trabajo es la herramienta de software llamada Trilogía, la cual es independiente de la plataforma y permite realizar las transformaciones UML-XML-BDOR-UML, basadas en el conjunto de reglas establecido.





## Capítulo 1. Introducción

En la actualidad, XML se ha convertido en el estándar para intercambio de datos entre aplicaciones y organizaciones. XML ofrece una solución para que sistemas heterogéneos puedan compartir sus datos a través de las redes. Además, está siendo adoptado rápidamente por las compañías de software como una norma para enviar y recibir datos. Sin embargo, XML resulta ineficiente cuando se maneja una gran cantidad de datos que son consultados frecuentemente.

Por otro lado, las bases de datos relacionales son los tipos de bases de datos más comúnmente usadas por las organizaciones y permiten almacenar una gran cantidad de datos. Ellas almacenan datos eficientemente y sin redundancia, ya que la información está normalizada. Proporcionan seguridad, veracidad y escalabilidad inigualables y pueden ser accedidas por un gran número de usuarios concurrentes.

Con la alta persistencia que tienen las bases de datos relacionales para almacenar formatos de datos y con la flexibilidad ofrecida por XML como mecanismo de intercambio de datos, surge la necesidad de unir las potencialidades que ofrecen ambas tecnologías.

Los Sistemas de Gestión de Bases de Datos (SGBD) como Oracle, SQL Server y DB2 ya han comenzado a ofrecer soporte XML; sin embargo, cada uno lo hace de forma propietaria y diferente. Oracle usa Java para transformar XML en un modelo de datos objeto relacional. IBM DB2 Extender utiliza un archivo DAD (Data Access Definition) codificado en XML para definir la transformación y Microsoft SQL Server extiende el lenguaje SQL introduciendo la función OPENXML.

A pesar que se han realizado muchos esfuerzos por parte de los programadores de software para desarrollar una herramienta que permita la transformación XML ↔ BDOR,



que sea independiente de la plataforma, las herramientas obtenidas no siguen los mismos estándares de diseño y resultan incompatibles unas con otras.

### **1.1 Antecedentes**

Existe una numerosa cantidad de trabajos que abarcan el tema de la transformación XML ↔ BDOR. En las siguientes secciones se presenta un breve resumen de los artículos más interesantes.

#### **1.1.1 Esfuerzos por establecer reglas que permiten migrar documentos XML a bases de datos y viceversa.**

En el artículo llamado “XML Structures for Existing Databases”, publicado por Kevin Williams y otros nueve desarrolladores de bases de datos, se exponen once reglas para migrar bases de datos a XML. La explicación detallada de estas reglas se encuentra en [Williams et al. 2000], así como también se incluyen otras dieciocho reglas que resuelven el problema de migrar documentos XML a bases de datos y presenta el uso de tecnologías de acceso a datos, tales como ADO (ADO+) y SQLServer, las cuales brindan soporte XML.

Cabe destacar que esta publicación solo define reglas basadas en la definición de tipo de documento (DTD).

#### **1.1.2 Implementación de herramientas que permiten convertir los datos contenidos en las bases de datos relacionales en documentos XML**

En [Turau 1999] se presenta el diseño e implementación de DB2XML. Una herramienta para transformar datos de base de datos relacionales en documentos XML. DB2XML proporciona tres funciones principales:

- Transforma el resultado de las consultas o del contenido completo de la base de datos en documentos XML.



- Genera los metadatos que describen las características de los datos en forma de definición de tipo de documento (DTD).
- Transforma los documentos XML generados, haciendo uso de hojas de estilo (XSLT) haciendo uso del lenguaje de estilos XML (XSL).

DB2XML está implementado en Java. El acceso a las bases de datos está basado en JDBC versión 1.2. DB2XML ha sido probado en diferentes plataformas (Unix y Win32), usando diferentes bases de datos (Oracle, SQL Server, MySQL, Access) y diferentes controladores (JDBC-ODBC, tipo 3 y 4).

Actualmente esta herramienta es limitada y no permite la importación de documentos XML en bases de datos. Tampoco permite la validación de documentos mediante el uso de esquemas XML y las consultas solamente se realizan mediante sentencias SQL.

### **1.1.3 Transformaciones basadas en tablas y transformaciones basadas en objetos**

El artículo presentado por [Bourret 2001], discute dos tipos de transformaciones: una transformación basada en tablas y una transformación objeto-relacional (basada en objetos). Estas transformaciones son bidireccionales y pueden ser usadas para transferir datos XML a base de datos y de base de datos a documentos XML.

La transformación objeto-relacional se realiza en dos pasos: Primero, una DTD es transformada en un esquema de objetos y luego el esquema de objetos es transformado en el esquema de base de datos.

El artículo solamente explica cómo se realizan las transformaciones haciendo uso de las DTDs y no explica cómo se realizan cuando se usan los esquemas (XML Schemas).



### **1.1.4 Uso del lenguaje de programación Java para importar documentos XML a bases de datos relacionales**

El artículo propuesto por [Gicqueau] introduce las bases para importar datos XML a bases de datos relacionales y explica (a través de ejemplos escritos en Java), los mecanismos para ir de un esquema XML a un esquema de base de datos (unmarshalling).

Dicho artículo solamente pretende mostrar el uso del lenguaje Java como una herramienta para llevar documentos XML a bases de datos y no intenta establecer normas o reglas a seguir para realizar esta transformación.

### **1.1.5 Integración de XML a las nuevas tecnologías de bases de datos relacionales**

En el artículo presentado por [Guardalben 2002], se discuten muchas de las ventajas que se obtienen al integrar XML con tecnologías de bases de datos relacionales. Además, se describen las características de los diferentes productos que ofrecen los vendedores de software de bases de datos (IBM, Microsoft y Oracle), comenzando por el más reciente, ADO.NET de Microsoft, el cual proporciona una opción para acceder datos relacionales vía XML. Finalmente presenta a Hit Software's Allora, que es una familia middleware para la integración de XML y Sistemas de Gestión de Bases de Datos Relacionales.

El artículo es solamente una referencia que muestra las características más resaltantes de los SGBD en el mercado que están comenzando a ofrecer soporte XML.

## **1.2 Áreas Relacionadas**

Antes de abarcar el tema principal de este documento, es importante conocer los conceptos y la terminología que utilizan estas dos grandes tecnologías: XML y las Bases de Datos Objeto Relacionales (BDOR). Aunque existan términos comunes que se utilizan indistintamente en ambas, su significado cambia dependiendo del contexto.



### 1.2.1 XML (eXtensible Markup Language)

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.

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. Por ejemplo:

```
<autor>
  Janmarco Rojas Nava
</autor>
```

Este elemento está delimitado por la etiqueta de inicio <autor>, la etiqueta de cierre </autor> y tiene como contenido la cadena “Janmarco Rojas Nava”.

Cada documento XML debe tener por lo menos un elemento en el cual van anidados todos los demás elementos. A continuación se muestra un documento que contiene un único elemento llamado raíz.

```
<?xml version="1.0"?>
<raiz>
  Dentro de este elemento van anidados todos los demás elementos del documento.
</raiz>
```

A la primera línea del documento XML mostrado arriba, se le conoce como Prólogo o declaración XML, la cual establece la versión de XML que se está utilizando. Al elemento del nivel más alto se le denomina elemento documento o elemento raíz.

Los elementos también pueden llevar atributos, los cuales 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. Por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<tesis titulo="Transformación UML-XML-BDOR-UML">
  <autor nombre="Janmarco" apellido="Rojas"/>
  <tutor nombre="Isabel" apellido="Besembel"/>
```



</tesis>

Los documentos XML son procesados por un analizador sintáctico (*parser*), que valida cada instancia de un documento XML comparándola con una DTD.

### 1.2.2 DTD (Document Type Definition)

La definición de tipo de documento (DTD) es básicamente un conjunto de reglas que un documento XML debe cumplir. En una DTD se declaran los tipos de elementos, atributos y entidades que se permiten en un documento XML.

El tipo de declaración más importante en una DTD es la declaración de elementos. Una declaración de elemento especifica el nombre del elemento y su modelo de contenido, es decir, el tipo de contenido válido para el elemento. La sintaxis usada es la siguiente:

<!ELEMENT nombreElemento (modeloContenido)>

Existen cinco modelos de contenidos que pueden ser declarados:

- **Solo elementos:** Es usado cuando un elemento solo puede contener otros elementos.
- **Contenido mixto:** Se usa cuando los elementos pueden contener cero o muchas instancias de una lista de elementos.
- **Solo texto:** Cuando los elementos solamente pueden contener cadenas de texto.
- **EMPTY:** En este caso, se declara un elemento vacío, que no permite contenido.
- **ANY:** Se permite que el elemento tenga cualquier elemento o información de caracteres como contenido.

Otra declaración muy común que aparece en una DTD es la declaración de atributos, que establece los atributos de un elemento determinado, sus tipos y si son necesarios o no. La sintaxis general utilizada es la siguiente:

<!ATTLIST nombreElemento definicion-atributo\*>





Donde la definición del atributo viene dada por:

definición-atributo::=nombre-atributo tipo-atributo declaracion-predeterminada

El nombre del atributo puede ser cualquier nombre XML válido. Los tipos de atributos pueden clasificarse en tres categorías:

- **Tipo cadena:** Es el tipo de atributo más común en una DTD. Un atributo CDATA puede tener como valor cualquier cadena de caracteres válida.
- **Tipos enumerados:** Cuando se necesita representar información que solo puede tomar valores de un conjunto pequeño de posibilidades.
- **Tipo token:** Son los tipos de atributos más importantes. Los tipos tokens pueden ser: ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS.

La declaración predeterminada de atributos se emplea para definir si un atributo es necesario o no, un valor fijo o si tiene un valor predeterminado. Las formas de declaración son: #REQUIRED, #IMPLIED, #FIXED o por omisión, respectivamente.

El siguiente ejemplo muestra el uso de las DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tesis SYSTEM "C:\tesis.dtd">
<tesis titulo="Transformación UML-XML-BDOR-UML">
  <autor nombre="Janmarco" apellido="Rojas"/>
  <tutor nombre="Isabel" apellido="Besembel"/>
</tesis>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT autor EMPTY>
<!ATTLIST autor
  nombre CDATA #REQUIRED
  apellido CDATA #REQUIRED
>
<!ELEMENT tesis (autor, tutor)>
<!ATTLIST tesis
  titulo CDATA #REQUIRED
>
<!ELEMENT tutor EMPTY>
```



```
<!ATTLIST tutor
  nombre CDATA #REQUIRED
  apellido CDATA #REQUIRED
>
```

### 1.2.3 Tecnologías XML

Las tecnologías XML son mecanismos que son usados para facilitar el acceso y la manipulación de los datos que están almacenados en los documentos XML.

#### Esquemas XML

Es un mecanismo para definir vocabularios XML y se utiliza como reemplazo de las DTD. Los esquemas XML son documentos XML bien formados y son una solución a los problemas de modelado que están fuera del alcance de las DTD.

Los esquemas XML soportan una variedad de tipos de datos atómicos, tales como cadena, decimal y fecha. Se han definido tipos de datos para mantener la compatibilidad con las DTDs, tales como ID y IDREF, que podrían ser usados por los atributos [Bergholz 2000]. También soporta la herencia como parte del concepto más general de derivación.

#### DOM (Document Object Model)

Es el mecanismo especificado por el W3C que permite acceder y manipular documentos XML. DOM carga el documento completo en memoria y construye un árbol a partir de su contenido, lo que hace a DOM excelente para la manipulación de documentos.

#### SAX (Simple API for XML)

SAX es un API orientada a eventos usada para analizar sintácticamente documentos XML, pero a diferencia de DOM, SAX requiere menos memoria, pero si el documento XML es muy complejo, es decir, si usa relaciones ID-IDREF, puede ser que se necesite analizar múltiples veces al documento para devolver la información requerida.



### **XSLT/XPath**

XSLT es el lenguaje de estilo para XML y permite transformar documentos XML en otros formatos, como por ejemplo, HTML o texto. XPath permite especificar nodos o valores para ser seleccionados desde el documento origen XML para su manipulación y presentación.

### **XML Query**

Es el lenguaje de consultas para XML que permite acceder al contenido de un documento XML. XQuery es un lenguaje funcional en el cual una consulta se representa con una expresión. Soporta varios tipos de expresiones: Expresiones de camino, constructores de elementos, expresiones que involucran funciones y operadores, expresiones condicionales, expresiones cuantificadas y expresiones que prueban y modifican tipos de datos.

### **XLink**

XLink (XML Linking Language), es una especificación que permite insertar elementos en documentos XML para describir y crear enlaces entre recursos. Un recurso es cualquier cosa que se pueda direccionar en la red, incluyendo por supuesto, información XML interna para el propio recurso.

XLink proporciona seis tipos de elementos que pueden ser usados para crear y describir características de enlaces. Solamente los tipos de elemento `simple` y `extended` crean enlaces XLink. Los elementos `simple` reproducen la funcionalidad de los enlaces HTML y los elementos `extended` son enlaces más poderosos y tienen la capacidad de establecer enlaces entre dos o más recursos.

Otras tecnologías XML que no serán discutidas son: XPointer, XBase, XForms, XML Fragment Interchange y XInclude.



## 1.3 UML

El Lenguaje de Modelado Unificado (UML), fue desarrollado con el objetivo de definir una notación estándar para el modelado de las aplicaciones construidas mediante objetos [Muller 1997].

### 1.3.1 Diagramas de clases

Los diagramas de clases expresan de manera general la estructura estática de un sistema, en términos de clases y de asociaciones entre estas clases. Una clase describe un conjunto de objetos, en cambio, una asociación describe un conjunto de enlaces; los objetos son instancias de clases y los enlaces son instancias de relaciones.

#### Las clases

Una clase es una descripción abstracta de un conjunto de objetos. Cada clase se representa bajo la forma de un rectángulo dividido en tres compartimentos (ver figura 1-1). El primero, contiene el nombre de la clase; el segundo, los atributos y el último, las operaciones. Los compartimentos pueden suprimirse para simplificar los diagramas.

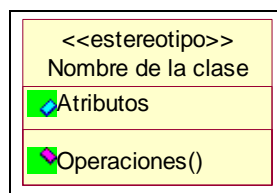


Figura 1-1. Representación gráfica de una clase.

El rectángulo que simboliza la clase también puede contener un estereotipo y propiedades, los cuales son definidos libremente por el usuario. Las propiedades designan los valores vinculados a un elemento de modelado, como los atributos, las asociaciones y las etiquetas.

La sintaxis para la descripción de atributos es la siguiente:



Nombre\_Atributo: Tipo\_Atributo = Valor inicial

La sintaxis seguida para la descripción de las operaciones es la siguiente:

Nombre\_Operacion(Nombre\_Argumento:Tipo\_Argumento=Valor\_Inicial,...) : Tipo\_Devuelto

UML define tres niveles de visibilidad para los atributos y las operaciones. El nivel de visibilidad se simboliza por los caracteres +, # y -, que corresponden con los niveles público, protegido y privado, respectivamente.

### Las asociaciones

Una asociación expresa una conexión semántica bidireccional entre clases. Una asociación es una abstracción de los enlaces que existen entre los objetos instancias de las clases asociadas.

Las asociaciones se representan a través de una línea entre las clases asociadas (ver figura 1-2). Las asociaciones generalmente poseen un nombre, el cual aparece en cursiva, en medio de la línea que simboliza la asociación.

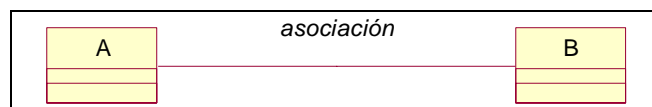


Figura 1-2. Representación gráfica de una asociación entre dos clases.

Una asociación binaria posee dos roles, uno en cada extremo. El rol describe el papel que juega una clase respecto a otra a través de una asociación. Cada rol tiene asociado un valor de multiplicidad (cardinalidad) que indica cuántos objetos de la clase considerada pueden ser relacionados con un objeto de otra clase.

El diagrama de la figura 1-3 muestra un ejemplo los roles y valores de multiplicidad existentes entre las clases Empleado y Departamento.

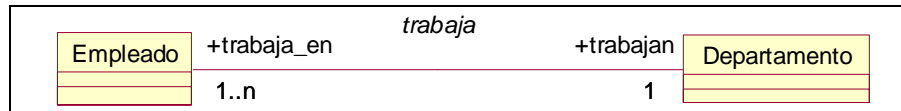


Figura 1-3. Representación de los roles y valores de cardinalidad.

La tabla 1-1 muestra los valores de cardinalidad más comunes:

Tabla 1-1. Valores de cardinalidad convencionales

Cardinalidad	Instancias que participan en la relación
1	Uno y solo uno
0..1	Cero o uno
M..N	De M a N (enteros naturales)
*	De cero a varios
0..*	De cero a varios
1..*	De uno a varios

### Las clases-asociaciones

Una asociación puede representarse por una clase cuando se necesita añadir atributos y operaciones a la asociación. Una clase de este tipo es llamada clase asociativa o clase-asociación y al igual que las otras clases, puede participar en otras relaciones en el modelo. La notación utiliza una línea punteada para vincular una clase a una asociación. La figura 1-4 muestra un ejemplo de clase-asociación.

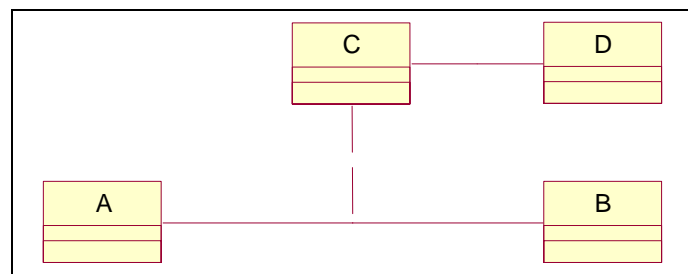


Figura 1-4. Ejemplo de clase-asociación.



## Las agregaciones

Una agregación representa una asociación no simétrica en la que uno de los extremos cumple un papel predominante respecto al otro extremo. Cualquiera que sea la multiplicidad, la agregación solo afecta a un rol de una asociación. La agregación se representa añadiendo un rombo al lado del agregado (ver figura 1-5).

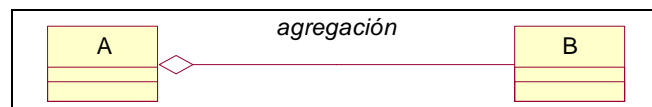


Figura 1-5. Representación de las agregaciones.

Los siguientes criterios implican una agregación:

- Una clase forma parte de otra clase.
- Los valores de atributos de una clase se propagan en los valores de atributos de la otra clase.
- Una acción sobre una clase implica una acción sobre otra clase.
- Los objetos de una clase son subordinados a los objetos de otra clase.

Los roles de una agregación no poseen ninguna restricción particular sobre los valores de cardinalidad. Esto significa que la cardinalidad del lado del agregado puede ser superior a uno (1).

## La composición

La composición es un caso particular de la agregación realizada por valor, donde los atributos son físicamente contenidos por el agregado. La composición implica una restricción sobre el valor de la cardinalidad en el lado del agregado donde solamente puede tomar los valores cero o uno (0 . . 1). El valor cero en el lado del componente corresponde a un atributo no explicitado.



La composición se representa en los diagramas por un rombo de color negro al lado del agregado (ver figura 1-6).

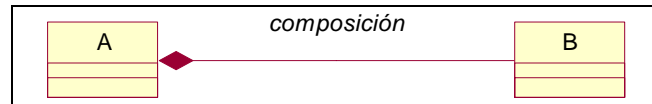


Figura 1-6. Representación gráfica de la composición.

Las clases realizadas por composición se llaman también clases compuestas. Estas clases proporcionan una abstracción de sus componentes.

### La navegación

Las asociaciones son navegables en ambas direcciones. En ciertos casos sólo es útil una dirección de navegación, la cual se representa por una flecha orientada (ver figura 1-7). La ausencia de flecha significa que la asociación es navegable en los dos sentidos.



Figura 1-7. Representación de una asociación navegable.

Una asociación navegable únicamente en un sentido puede ser vista como una semi-asociación.

### La generalización

UML emplea el término generalización para designar la relación de clasificación entre un elemento más general y un elemento más específico.

La relación de generalización se representa por medio de una flecha de forma triangular que apunta desde la clase más especializada hacia la clase más general (ver figura 1-8).

Los atributos, las operaciones, las relaciones y las restricciones definidas en la superclase se heredan íntegramente en las subclases.



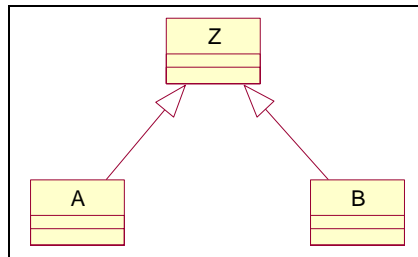


Figura 1-8. Representación de la generalización.

### Las clases abstractas

Las clases abstractas no se pueden instanciar directamente, sino que sirven de especificación más general (de tipo), para manipular los objetos instancias de una o varias de sus subclases.

Una clase se designa como abstracta por medio de la propiedad booleana abstracta definida para todos los elementos generalizables: los tipos, los paquetes y los estereotipos. Por convención, el nombre de una clase abstracta va en letras cursivas (ver figura 1-9).

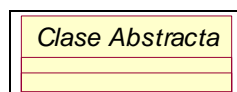


Figura 1-9. Representación de una clase abstracta.

## 1.4 Bases de Datos Relacionales

Las bases de datos relacionales son el tipo de bases de datos más comúnmente usado en la actualidad. Una base de datos relacional está representada por un conjunto de relaciones o tablas interrelacionadas. Las interrelaciones pueden ser de uno a uno, de uno a muchos o de muchos a muchos. Una tabla puede almacenar a un número indefinido de tuplas (filas). Cada tupla está compuesta de un número fijo de atributos (columnas), que corresponden a los atributos del modelo lógico de datos. Una base de datos relacional soporta muchos tipos de datos. Cada tabla está caracterizada por una única clave principal, la cual puede estar referenciada por una clave foránea en otra tabla que almacena su mismo valor.



Existen dos conceptos fundamentales en el diseño de bases de datos: Diseño Lógico de Datos y Diseño Físico de Datos.

### **1.4.1 Diseño Lógico**

El diseño lógico tiene que ver con el modelado teórico de los datos que contendrá la base de datos. La construcción de un sistema comienza por la identificación de sus entidades y sus relaciones. Una entidad es cualquier entidad física o concepto, donde cada una está caracterizada por sus atributos.

#### **Entidades**

El primer paso en un diseño lógico, es identificar las entidades que serán modeladas. Las entidades pueden definir cualquier persona, lugar, cosa o concepto (sustantivos). Una entidad es análoga a un elemento en XML.

#### **Atributos**

Los atributos son las características que describen a las entidades (adjetivos). Se deben definir atributos para cada una de las entidades creadas. Los atributos tienen un tipo de dato básico (atómico).

#### **Relaciones**

El paso final en el diseño lógico es identificar las relaciones entre las entidades. Las relaciones representan acciones (verbos) que relacionan a las entidades.

### **1.4.2 Diseño Físico**

El diseño físico de datos involucra los aspectos reales del diseño de bases de datos, como lo son: Seleccionar la plataforma, escribir el código para crear las tablas, definir las columnas de las tablas, etc.



### **Tablas**

Las tablas son análogas a las entidades. Debe crearse una tabla en el diseño físico para cada entidad creada en el diseño lógico. Una vez que se han definido las tablas se necesitan crear las columnas que tendrán y asignar un tipo de dato para cada una de las mismas.

### **Claves Principales**

Las claves principales (primary keys) son usadas para identificar de forma única las tuplas de cada tabla.

### **Claves Foráneas**

Las claves foráneas (foreign keys) son usadas para expresar asociaciones entre tablas. Para cada relación que se ha identificado en el diseño lógico, se debe añadir una asociación (clave foránea) en la base de datos. Una clave foránea referencia a una clave principal e indica como se relacionan las tablas.

### **Índices**

Los índices son usados por las bases de datos relacionales para acceder rápidamente a la información. Como regla general, debe crearse un índice para cada clave primaria y foránea y para cada campo no clave que forme parte de las consultas.

### **Disparadores**

Los disparadores o gatillos (triggers), pueden ser usados en una base de datos relacional para tomar alguna acción cuando se inserte, actualice o elimine un registro de una tabla.



### **Procedimientos Almacenados**

Los procedimientos almacenados son usados para encapsular funciones de negocio en una función precompilada. Son parecidos a los disparadores, a diferencia que éstos pueden ser ejecutados a solicitud del programador.

## **1.5 Bases de Datos Objeto-Relacionales**

El término base de datos objeto-relacional se usa para describir una base de datos que ha evolucionado desde el modelo relacional hasta una base de datos híbrida, que contiene ambas tecnologías: relacional y de objetos.

Los modelos de datos relacionales orientados a objetos extienden el modelo relacional proporcionando un sistema de tipos más rico que incluye la programación orientada a objetos y añade constructores a los lenguajes de consulta relacionales como SQL para trabajar con los tipos de datos añadidos. Los sistemas de tipos extendidos permiten que los atributos de las tuplas tengan tipos complejos. Estas extensiones intentan conservar las bases relacionales al tiempo que extienden la capacidad de modelado. Los sistemas de bases de datos basados en el modelo objeto-relación proporcionan un modo de cambio adecuado para los usuarios de las bases de datos relacionales que deseen utilizar características orientadas a objetos [Silberschatz et al. 2002].

El modelo relacional anidado permite la representación directa de las estructuras jerárquicas y el LDD de SQL es extendido para trabajar con tipos complejos, incluyendo las relaciones anidadas y los objetos.

### **1.5.1 El Modelo Relacional Anidado**

El modelo relacional anidado es una extensión del modelo relacional en la que los dominios pueden ser atómicos o de relación. Por lo tanto, el valor de las tuplas de los atributos puede ser una relación, y las asociaciones pueden guardarse en otras relaciones. Los objetos complejos pueden entonces representarse mediante una única tupla de las relaciones anidadas.



### 1.5.2 Los Tipos Complejos y la Herencia

El modelo de datos orientado a objetos ha creado la necesidad de características como la herencia y las referencias a los objetos. Los sistemas de tipos complejos permiten crear tipos estructurados y colecciones. Los tipos creados se registran en el esquema guardado en la base de datos. Por tanto, las instrucciones que tengan acceso a la base de datos podrán hacer uso de las definiciones de los tipos.

Los sistemas de tipos complejos suelen permitir otros tipos de colecciones, como los arrays y los multiconjuntos (es decir, colecciones no ordenadas en las que un elemento puede figurar varias veces).

La herencia puede hallarse en el nivel de los tipos o en el nivel de las tablas. La herencia hace más natural la definición de los esquemas. Sin la herencia de tablas, el diseñador de los esquemas tiene que vincular de modo explícito las tablas correspondientes a las subtablas con tablas correspondientes a las supertablas mediante las claves primarias y definir los enlaces entre las tablas para asegurar los enlaces referenciales y de cardinalidad.

## 1.6 Justificación

Existen muy buenas razones para usar la tecnología XML con la tecnología de las Bases de Datos Objeto-Relacionales (BDOR).

Las bases de datos relacionales son una tecnología madura que ha evolucionado y que ha permitido a sus usuarios, modelar complejas relaciones entre datos que necesitan ser almacenados. Además, son los tipos de bases de datos más comúnmente usadas y nos permiten gestionar una gran cantidad de datos de una forma segura. Por otro lado, algunas herramientas middleware, en su mayoría bases de datos relacionales y servidores de objetos que permiten XML, usan un tipo de transformación más sofisticada llamada objeto-relacional, la cual modela el documento XML como un árbol de objetos.

XML en cambio, se está convirtiendo rápidamente en el protocolo universal para transmitir, intercambiar y manipular datos. XML es una solución muy simple a un



problema complejo. XML ofrece un formato estándar en forma de documento autodefinido independiente de la plataforma.

Algunas de las razones del por qué llevar los datos de una base de datos objeto-relacional a XML son:

- Para compartir los datos de negocio entre sistemas que pueden estar en diferentes niveles de la empresa.
- Para lograr la interoperabilidad con sistemas heterogéneos (incompatibles).
- Para exponer los datos legados en otras aplicaciones que utilizan XML.
- Para realizar transacciones de negocios (B2B).

De igual forma existen razones para llevar el contenido de un documento XML a bases de datos.

- Cuando se tiene una gran cantidad de datos en documentos XML que se necesitan consultar frecuentemente.
- Para aprovecharse de las características transaccionales y de seguridad que proporcionan las bases de datos.
- Para aprovechar las diversas ventajas que ofrecen los recientes sistemas de gestión de bases de datos objeto-relacionales.

La estructura jerárquica de XML puede ser usada para crear modelos que no se ajustan fácilmente al paradigma de tablas y relaciones de las bases de datos. También existen estructuras anidadas complejas que no pueden representarse en estructuras de creación de tablas y existen restricciones en las DTDs que no pueden ser representadas con tablas y claves. Sin embargo, el modelado objeto-relacional puede ser una solución a estos problemas.



## 1.7 Objetivo

Este estudio tiene como objetivo fundamental la definición de un conjunto de reglas para la transformación de los diagramas de clases en UML en múltiples documentos esquemas XML y en un esquema de bases de datos objeto-relacional para cada uno de los posibles esquemas XML. Dicho conjunto de reglas será probado mediante la implementación de un prototipo de software, desarrollado utilizando un lenguaje de programación que soporte las tecnologías XML existentes y que permita:

- Generar la estructura de un documento esquema XML a partir de un diagrama de clases UML.
- Generar el esquema de una base de datos objeto relacional (BDOR) a partir del documento esquema XML y generar la base de datos que contendrá los datos de los documentos instancia.
- Generar el diagrama de clases UML a partir del esquema BDOR o bien del diccionario de datos de la BDOR.

Además, el prototipo a desarrollar debe ser independiente del sistema operativo y debe funcionar preferiblemente para bases de datos Oracle, SQL Server, MySQL y PostgreSQL.

Dados estos objetivos, se plantea inicialmente que, si la nueva recomendación Esquema XML permite solventar los problemas de modelado que están fuera del alcance de las DTDs, entonces se podrá establecer un conjunto de reglas bien definidas para la construcción esquemas XML y esquemas BDOR, modelados a partir de diagramas de clases UML, que constituirán las bases para lograr un intercambio de datos XML↔BDOR que sea factible».



## 1.8 Métodos de desarrollo

Para el desarrollo de este trabajo, se llevaron a cabo las tres primeras fases que se siguen en el método científico. En la primera fase, correspondiente a la observación, se hizo una recopilación acerca del material informativo concerniente al problema que abarca la transformación XML $\leftrightarrow$ BDOR. La información recopilada fue clasificada, ordenada y posteriormente analizada. En el análisis se observó que la mayoría de los trabajos realizados, que intentaban resolver el problema de la transformación XML $\leftrightarrow$ BDOR, presentaban dificultades al utilizar las definiciones de tipo de documento (DTDs) y que además, en ninguno de éstos se hacía uso de una herramienta de modelado que permitiera definir la estructura de los documentos XML. Esto tuvo como consecuencia, un cambio radical en la perspectiva inicial y el problema de la transformación se enfocó en hacer uso de los diagramas de clases UML como una herramienta de modelado, que permitiría modelar la estructura de los documentos esquema XML, que luego se transformarían en esquemas BDOR equivalentes.

En la segunda fase se propuso como hipótesis inicial que, si los esquemas XML lograban resolver los problemas de modelado que se presentaban al usar las DTD, entonces se podrían definir un conjunto de reglas para llevar a cabo la transformación UML-XML-BDOR-UML, que permitirían establecer las bases del intercambio de datos XML $\leftrightarrow$ BDOR.

En la tercera fase, correspondiente a la experimentación, se desarrolló un prototipo de software llamado Trilogía que permitió verificar, comprobar y dar por aceptada la hipótesis. Para el desarrollo del prototipo se utilizó el Modelo de Procesos Watch descrito por [Montilva et al. 2000].

El modelo de procesos Watch es una integración de los modelos de desarrollo de software Cascada, Modelo V y Modelo de Prototipos. Su objetivo es servir de guía para la construcción de aplicaciones de pequeña y de mediana complejidad. Este modelo sigue el paradigma de la Orientación por Objetos propuesto por Bruegge y Dutoit.

El modelo de procesos Watch usa una metáfora de reloj para representar la forma y la dinámica del modelo en término de sus fases (ver figura 1-10).



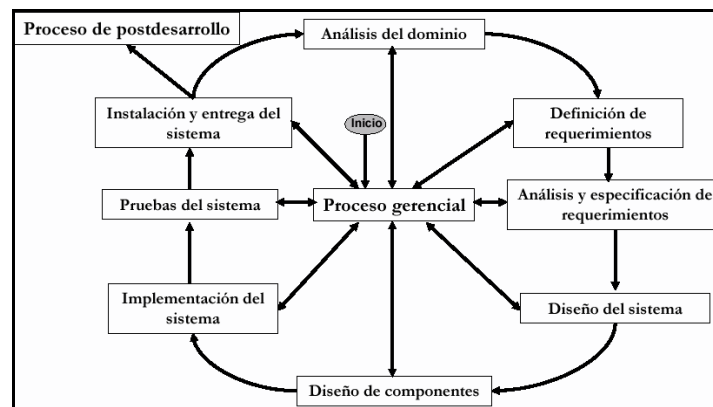


Figura 1-10. El Modelo de Procesos Watch.

En el modelo Watch, el proceso gerencial dirige y controla la ejecución de cada una de las fases del proceso de desarrollo. El proceso de desarrollo comprende ocho fases:

1. **Análisis del dominio:** Es la primera fase de desarrollo a ser ejecutada, en la cual se identifica y se describe el dominio de la aplicación.
2. **Definición de requerimientos:** Es la siguiente fase de desarrollo y su propósito es definir informalmente los requerimientos funcionales y no funcionales de la aplicación.
3. **Especificación y análisis de requerimientos:** En esta fase se hacen formales los requerimientos de los usuarios, para que puedan ser entendidos por los diseñadores o desarrolladores del sistema.
4. **Diseño del sistema:** En la fase del diseño del sistema se traduce la especificación de requerimientos en la especificación del diseño de la aplicación. La estructura de la aplicación se diseña en términos de un conjunto de componentes, conexiones y restricciones, denominada arquitectura del sistema. También se especifican en esta fase, los detalles de la interfaz gráfica del usuario.
5. **Diseño de componentes:** En esta fase se diseñan con detalle, cada uno de los componentes y conexiones identificados en la arquitectura de la aplicación.
6. **Implementación del sistema:** La fase de implementación del sistema traduce las especificaciones de diseño en un producto definitivo de software. Los componentes de software son codificados y probados usando diferentes técnicas.



7. **Pruebas del sistema:** En esta fase se llevan a cabo las pruebas funcionales, de desempeño y de aceptación de la aplicación. El resultado principal de esta fase es tener un sistema probado que este listo para ser instalado.
8. **Instalación y entrega del sistema:** Es la última fase del proceso de desarrollo, en la cual el sistema es instalado y probado en el ambiente operacional. Una vez que el sistema está completamente funcional, el sistema es entregado al cliente.

## 1.9 Alcance

La importancia de este trabajo es, en esencia, establecer nuevas reglas que permitan dar solución al problema de la transformación XML $\leftrightarrow$ BDOR, donde UML juega un papel primordial como herramienta de modelado. Estas reglas fundarán las bases para llevar a cabo un intercambio bidireccional de datos entre los documentos XML y las BDOR.

El resultado a obtener será una herramienta de software innovadora que utiliza estas reglas para generar esquemas XML y esquemas BDOR a partir de diagramas de clase UML, con la expectativa de que influya fuertemente las áreas relacionadas con UML, XML y BDOR.

A futuro, el modelo de componentes de J2EE puede ser utilizado para desarrollar un componente de software reutilizable que ofrezca el servicio de transformación XML $\leftrightarrow$ BDOR utilizando las reglas que serán establecidas en este trabajo.

## 1.10 Organización del manuscrito

A continuación se presenta un resumen de los siguientes capítulos y secciones en los que está dividido este trabajo:

- **Capítulo 2, Revisión Bibliográfica:** Este capítulo está dividido en tres secciones.
  - La primera sección está orientada a presentar en profundidad el Modelo de Objetos del Documento (DOM) para la manipulación sintáctica de documentos XML. Se



hace una introducción a las recomendaciones *DOM Level 3 Core* y *DOM Level 3 Load and Save*, las cuales constituyen un conjunto de interfaces que permiten cargar el contenido de un documento XML, manipular su estructura y posteriormente guardar esta estructura como un archivo XML.

- En la segunda sección se explican los componentes que conforman la estructura de un documento esquema XML, tomando en cuenta la sintaxis recomendada por la especificación Esquema XML. La sintaxis es utilizada para la construcción de esquemas que están acordes con dicha especificación.
- En la última parte se presentan los fundamentos para la creación de esquemas BDOR, tomando como estándar el lenguaje SQL-92. La creación de un esquema BDOR implica tener conocimientos claros acerca del lenguaje de definición de datos (LDD) de SQL.
- **Capítulo 3, Transformación UML-XML-BDOR-UML:** En este capítulo se presentan tres grandes secciones, las cuales exponen un conjunto de reglas para llevar a cabo la transformación UML-XML-BDOR-UML, que es la principal contribución de este trabajo:
  - En la primera parte se demuestra el uso de los diagramas UML para la construcción de esquemas XML. Los diagramas de clase UML son utilizados para definir la estructura de los documentos XML. Posteriormente, cada clase y asociación del diagrama se representa como un componente dentro de un documento esquema XML haciendo uso reglas específicas. Todo el modelado que se considera en éste capítulo está referido al área de bases de datos, donde los diagramas de clases UML se realizan solo para aquellas clases que ameritan persistencia en BDOR.
  - En la segunda parte se explica la forma de convertir los documentos esquema XML obtenidos en esquemas BDOR. Los componentes que conforman la estructura del esquema XML son derivados en un conjunto de sentencias SQL que constituyen un esquema BDOR, haciendo uso de reglas específicas.
  - La última sección presenta la forma de obtener diagramas de clases UML a partir de los esquemas XML. El conjunto de sentencias que conforman un esquema BDOR



puede ser descrito como un conjunto de clases y asociaciones en un diagrama de clases UML.

- **Capítulo 4, Diseño del prototipo:** En este capítulo se presentan los resultados obtenidos en las cuatro primeras fases del proceso de desarrollo del modelo Watch, correspondientes al análisis y dominio de la aplicación, definición de requerimientos, análisis y especificación de requerimientos, y diseño del sistema. Se hace uso de los diagramas de casos de uso, diagramas de colaboración y diagramas de clase UML para la especificación de los requerimientos, identificación de responsabilidades y obtener la estructura estática de las clases de diseño, respectivamente.
- **Capítulo 5, Implementación:** Se presentan los resultados finales obtenidos en las fases de diseño e implementación de la aplicación. Cada paquete que conforma la arquitectura es descrito en término de sus clases y se presenta la arquitectura de despliegue de la aplicación. El resultado de la fase de implementación es la herramienta de software llamada Trilogía, que es una solución al problema de la transformación UML-XML-BDOR-UML.
- **Capítulo 6, Conclusiones y Recomendaciones:** Este último capítulo concierne a las conclusiones obtenidas en cada una de las partes del manuscrito y las posibles recomendaciones que se tuvieron al respecto.





## Capítulo 2. Revisión Bibliográfica

La manipulación un documento esquema XML involucra el uso de APIs que permitan analizar sintácticamente la estructura del documento. Los analizadores sintácticos como DOM (Document Object Model) y SAX (Simple API for XML) permiten manipular contenido XML como un árbol de nodos o mediante procesamiento de eventos, respectivamente.

Las recomendaciones DOM Level 3 Core y DOM Level 3 Load and Save ofrecen interfaces independientes de la plataforma y del lenguaje que permiten cargar el contenido de un documento esquema XML, manipular completamente su estructura y posteriormente guardar esta estructura como un documento XML. La creación de un documento esquema XML usando estas interfaces debe cumplir con las definiciones de un lenguaje de esquemas, especificadas en la recomendación W3C Esquema XML. El documento esquema XML generado puede ser transformado entonces en un esquema BDOR equivalente siguiendo un conjunto de reglas bien definidas, las cuales se mencionarán en el próximo capítulo.

### 2.1 DOM

El Modelo de Objetos del Documento (DOM) es un API para validar documentos XML y HTML, que define la estructura lógica de los documentos y la forma en que son accedidos y manipulados.



### 2.1.1 Arquitectura DOM

La arquitectura DOM está dividida en módulos. Cada módulo está dirigido a un dominio particular. Los dominios cubiertos por el API DOM actual son XML, HTML, CSS y árbol de eventos. La especificación DOM Level 3 define los módulos descritos en la tabla 2-1.

Tabla 2-1. Módulos definidos por la especificación DOM Level 3

Módulo	Descripción
DOM Core	Define una representación en estructura de árbol del documento, también denominado como árbol DOM.
DOM XML	Extiende la plataforma Core para necesidades XML 1.0 específicas, tales como procesamiento de instrucciones, CDATA y entidades.
DOM HTML	Define un conjunto de métodos para manipular documentos HTML.
DOM Events	Define eventos orientados a la manipulación de árboles XML.
DOM Cascading Style Sheets (CSS)	Define un conjunto de métodos para manipular hojas de estilo CSS fácilmente.
DOM Load and Save	Este módulo incluye una variedad de opciones para cargar un documento XML como un árbol DOM o para guardar un árbol DOM como un documento XML.
DOM Validation	Define un conjunto de métodos para modificar el árbol DOM y hacerlo válido.
DOM XPath	Define un conjunto de funciones para consultar un árbol DOM usando expresiones XPath 1.0.

La figura 2-1 muestra la arquitectura de DOM, la cual contiene todos los módulos definidos a lo largo de las especificaciones.

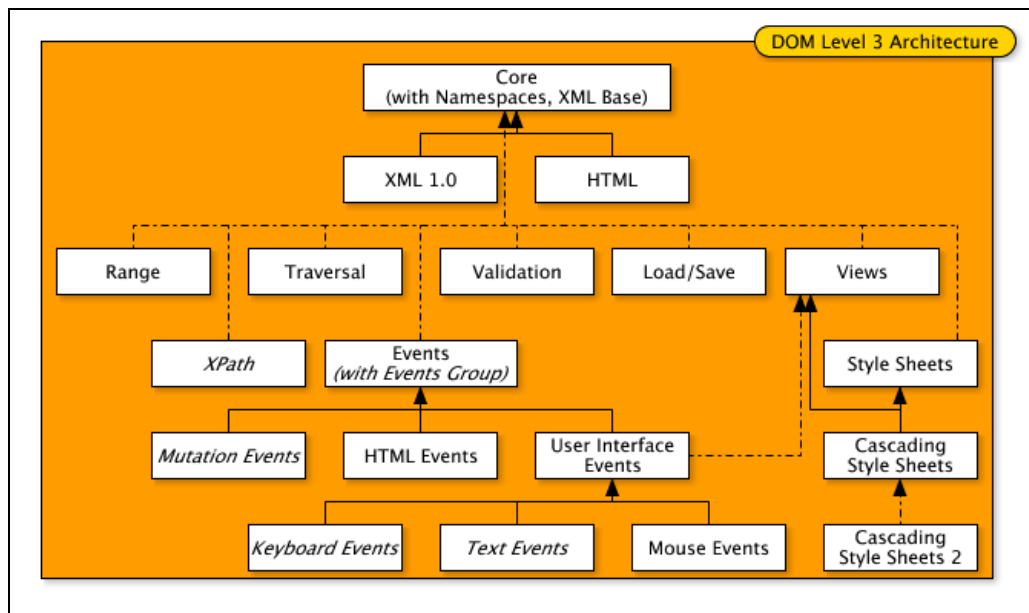


Figura 2-1. Arquitectura DOM Level 3 [12]

### 2.1.2 Interfaces e Implementaciones

DOM especifica varias interfaces, las cuales pueden ser usadas para gestionar documentos XML y HTML. Las interfaces definen la manera de acceder y manipular la representación interna de un documento pero no implican una implementación determinada, por lo tanto, DOM está diseñado para evitar dependencias con alguna implementación en particular. Esto quiere decir que una aplicación DOM puede proporcionar objetos e interfaces adicionales que no se encuentran en la especificación y todavía ser considerada una aplicación compatible con DOM.

Dado que pueden existir diferentes tipos de implementaciones, DOM proporciona un conjunto central de interfaces y un número de módulos opcionales que permiten trabajar con documentos XML, HTML y CSS.

### 2.1.3 DOM Core

La especificación define un conjunto de objetos e interfaces para acceder y manipular los objetos del documento. La API DOM Core también permite la creación de un objeto





documento (`Document`) usando solamente llamados de la API DOM. Una solución para cargar un documento y guardarlo persistentemente se presenta en la especificación DOM Level 3 Load and Save.

### Tipos básicos definidos por la especificación DOM Core

Para asegurar la interoperabilidad, la especificación DOM Core define los siguientes tipos básicos que son usados en diversos módulos DOM (ver tabla 2-2).

Tabla 2-2. Tipos básicos definidos por la especificación DOM Core

Tipo Básico	Descripción
<code>DOMString</code>	Este tipo es usado para almacenar caracteres Unicode como una secuencia de unidades de 16bits (UTF-16). Para Java el tipo básico <code>DOMString</code> está acotado por el tipo <code>String</code> dado que ambos lenguajes también usan codificación UTF-16.
<code>DOMTimeStamp</code>	Se usa para almacenar un tiempo relativo o absoluto. Un <code>DOMTimeStamp</code> representa un número de milisegundos. Para Java, el tipo <code>DOMTimeStamp</code> está acotado por el tipo <code>long</code> .
<code>DOMUserData</code>	Es usado para almacenar datos de aplicaciones. Un <code>DOMUserData</code> representa una referencia a un dato de aplicación. Para Java, el tipo básico <code>DOMUserData</code> está acotado por el tipo <code>Object</code> .
<code>DOMObject</code>	Este tipo es usado para representar una referencia a un objeto. Para Java, el tipo básico <code>DOMObject</code> está acotado por el tipo <code>Object</code> .

### Interfaces fundamentales del módulo DOM Core

Las interfaces descritas en la tabla 2-3 son consideradas fundamentales y deben estar completamente implementadas por todas las implementaciones compatibles con DOM Level 3.

Tabla 2-3. Interfaces fundamentales definidas por DOM Core

Interfaz	Descripción
<code>DOMStringList</code>	Proporciona la abstracción de una colección ordenada de valores



	DOMString, sin definir como está implementada la colección. Los ítems contenidos en la lista DOMStringList son accedidos a través de un índice que comienza desde cero (0).
NameList	Proporciona la abstracción de una colección ordenada de pares paralelos de valores de nombres y de espacio de nombres. Los ítems contenidos en la lista NameList son accedidos a través de un índice que comienza desde cero (0).
DOMImplementationList	Proporciona la abstracción de una colección ordenada de implementaciones DOM, sin definir cómo está implementada la colección. Los ítems contenidos en la lista DOMImplementationList son accedidos a través de un índice que comienza desde cero (0).
DOMImplementationSource	Permite obtener una o más implementaciones a partir de la versión y la propiedad que son requeridos.
DOMImplementation	Proporciona un número de métodos para llevar a cabo operaciones que son independientes de cualquier instancia particular del modelo de objetos del documento.
DocumentFragment	Esta interfaz es usada cuando se quiere extraer una porción del árbol del documento o para crear un nuevo fragmento de un documento. Un objeto DocumentFragment representa un objeto Document mínimo.
Document	La interfaz Document representa el documento completo, ya sea XML o HTML. Conceptualmente es la raíz del árbol del documento y proporciona acceso principal a los datos del documento.
Node	Es el tipo de dato principal de DOM, el cual representa a un nodo simple en el árbol del documento.
NodeList.	Proporciona la abstracción ordenada de un conjunto de nodos, sin definir como está implementada la colección.
NamedNodeMap	Los objetos implementados en esta interfaz son usados para representar colecciones de nodos que pueden ser accedidos por nombre.
CharacterData	La interfaz CharacterData extiende la interfaz Node con un conjunto de atributos y de métodos para acceder datos de caracteres en el DOM.
Attr	La interfaz Attr representa un atributo en un objeto Element.



Element	La interfaz Element representa un elemento en un documento XML o HTML.
Text	Esta interfaz hereda de CharacterData y representa el contenido textual de un objeto Element o Attr.
Comment	Esta interfaz hereda de CharacterData y representa el contenido de un comentario.
TypeInfo	La interfaz TypeInfo representa a un tipo referenciado desde nodos Element o Attr, especificados en el esquema asociado con el documento.
UserDataHandler	Esta interfaz define un manejador que es llamado cuando un nodo asociado a un objeto a través del método Node.setUserData() está siendo clonado, importado o renombrado.
DOMErrorHandler	Esta interfaz puede ser llamada por la implementación DOM cuando se presentan errores en el procesamiento de datos XML, o cuando se hace algún otro procesamiento.
DOMLocator	Esta interfaz describe una ubicación.
DOMConfiguration	La interfaz DOMConfiguration representa la configuración de un documento y mantiene una tabla de parámetros reconocidos.

La jerarquía de estas clases se muestra en la figura 2-2.

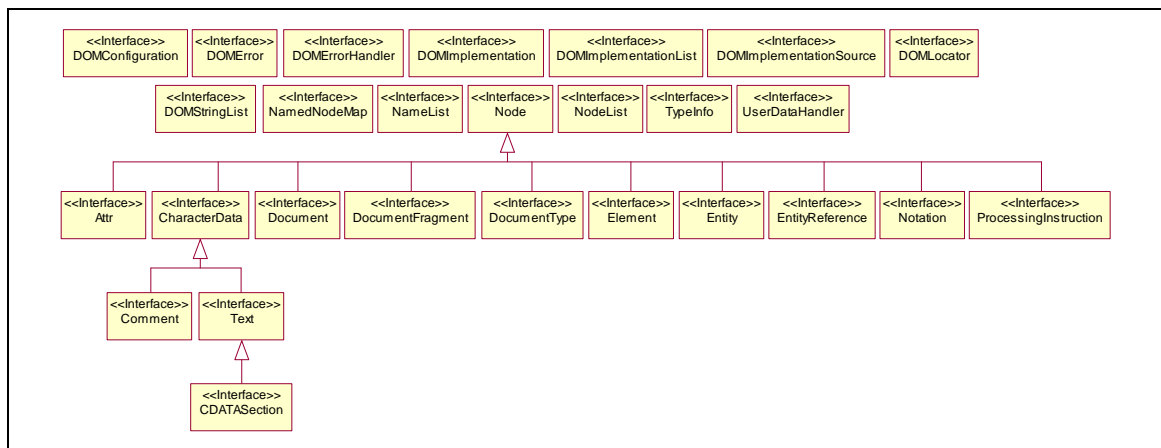


Figura 2-2. Interfaces del módulo DOM Core



## Interfaz DOMImplementation

La interfaz `DOMImplementation` proporciona métodos que se aplican a cualquier documento (ver figura 2-3).

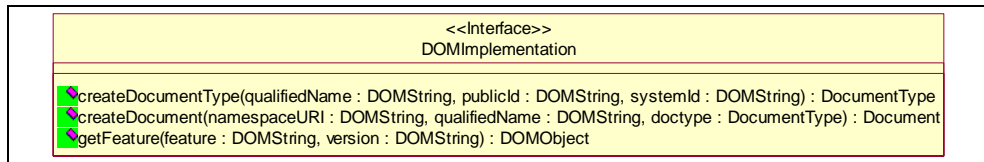


Figura 2-3. Interfaz `DOMImplementation`

Los métodos asociados a esta interfaz permiten:

- Crear un nodo `DocumentType` vacío.
- Crear un objeto `Document`.
- Obtener un objeto DOM especializado para una propiedad y una versión específica.

## Interfaz Node

`Node` es la interfaz fundamental principal de DOM, ya que la mayoría de las interfaces extienden de ésta (ver figura 2-4). Los objetos que implementan la interfaz `Node` y no aceptan nodos hijos, como por ejemplo los objetos `Text`, se les denomina nodos hoja. Si se intenta añadir un hijo a tales nodos puede resultar en un error de excepción.

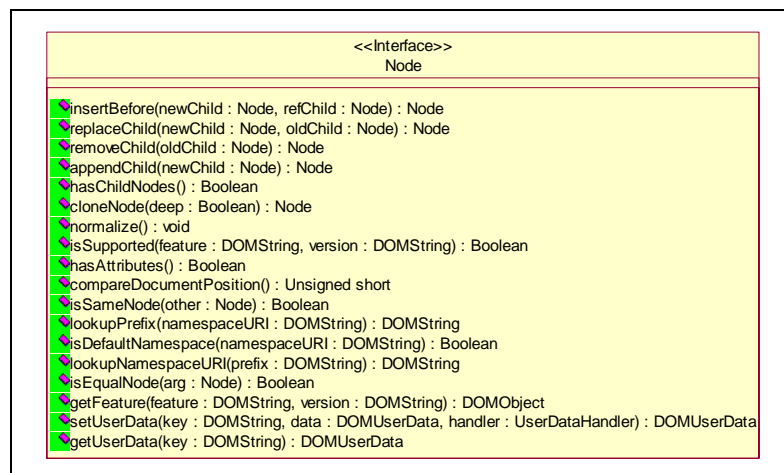


Figura 2-4. Interfaz `Node`



Los métodos asociados a esta interfaz permiten:

- Añadir, reemplazar o eliminar los nodos hijo que pertenecen a un nodo específico.
- Conocer si un nodo tiene hijos.
- Conocer si un nodo tiene atributos.
- Conocer si un nodo es el mismo nodo o si es igual a otro nodo.
- Duplicar un nodo (clonar un nodo).

### Interfaz Document

La interfaz `Document` extiende la interfaz `Node` y es usada para representar un documento completo XML o HTML (ver figura 2-5). Los demás elementos no pueden existir fuera del contexto de un objeto `Document`, por esta razón la interfaz `Document` contiene los métodos de fábrica necesarios para crear estos objetos. Los objetos de tipo `Node` que han sido creados tienen un atributo llamado `ownerDocument` el cual lo asocia con el objeto `Document` en cuyo contexto fueron creados.

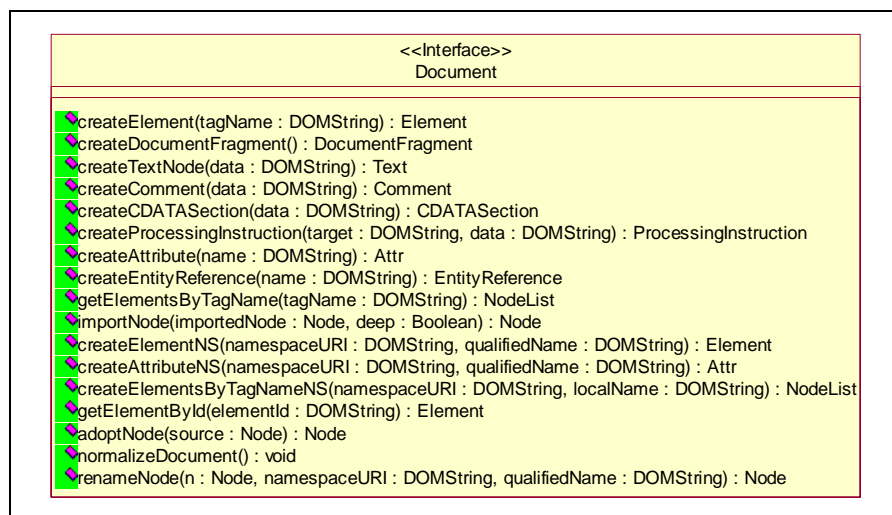


Figura 2-5. Interfaz Document

Los métodos asociados a esta interfaz permiten:



- Crear todos los demás objetos que pueden estar contenidos dentro del contexto del documento, tales como elementos, atributos, procesamiento de instrucciones y comentarios, entre otros.
- Importar o adoptar un nodo que existe dentro del contexto de otro documento.
- Obtener un elemento dado el valor de su atributo ID.
- Obtener una lista de nodos de todos los elementos dentro del documento que tengan un nombre de etiqueta específico.
- Cambiar el nombre de un nodo.
- Normalizar el documento.

### Interfaz Attr

La interfaz `Attr` representa un atributo en un objeto `Element` (ver figura 2-6). Los objetos `Attr` difieren de los demás objetos que heredan de la interfaz `Node` en ciertos aspectos:

- Solamente pueden estar asociados con los objetos `Element` y no pueden ser hijos de ningún otro objeto.
- Los atributos `parentNode`, `previousSibling` y `nextSibling`, los cuales son heredados de `Node`, tienen un valor nulo para los objetos `Attr`.
- Los objetos `Attr` pueden ser accedidos a través de los métodos correspondientes de los objetos `Element`.

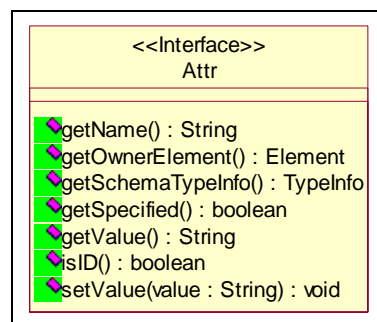


Figura 2-6. Interfaz `Attr`

Los métodos asociados a esta interfaz permiten:



- Obtener el nombre del atributo.
- Obtener el objeto `Element` al cual esta asociado el atributo.
- Obtener el tipo de información que está asociada con el atributo.
- Conocer si al atributo le fue dado un valor en la instancia del documento.
- Obtener o asignar el valor del atributo.
- Conocer si el atributo es de tipo ID.

### Interfaz `Element`

La interfaz `Element` extiende la interfaz `Node` y es usada para representar un elemento dentro del documento XML o HTML, donde cada objeto `Element` puede tener asociado un conjunto de atributos (ver figura 2-7).

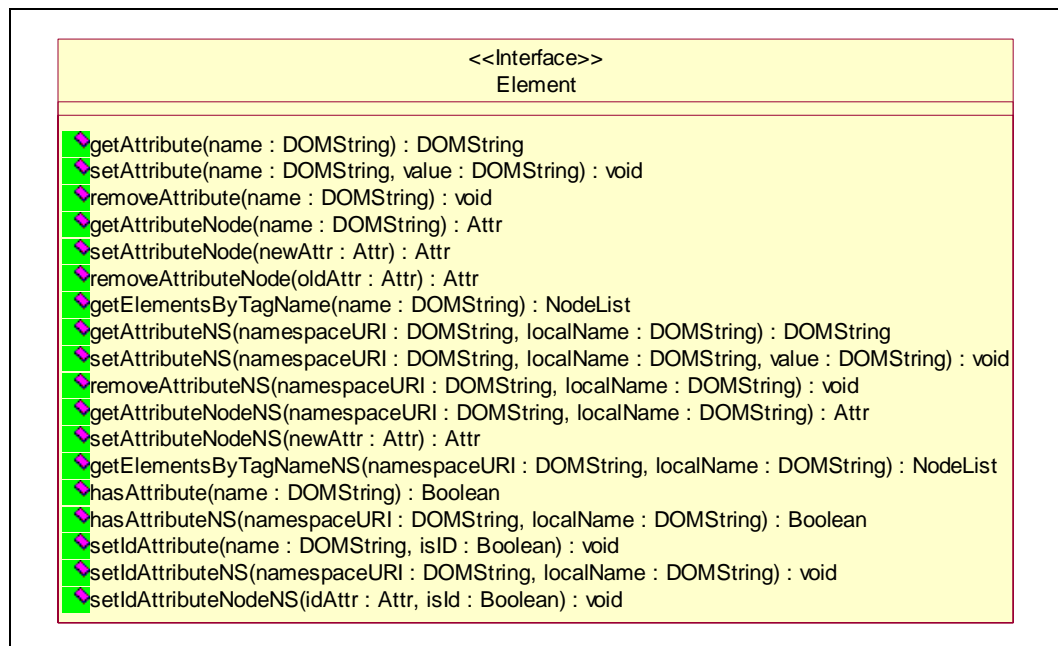


Figura 2-7. Interfaz `Element`

Los métodos asociados a esta interfaz permiten:

- Obtener o asignar el valor de un atributo.
- Obtener, añadir o eliminar un nodo atributo.



- Obtener una lista de nodos que tengan un nombre de etiqueta específico.
- Conocer si un nodo elemento tiene atributos.
- Establecer un atributo de tipo ID.

### 2.1.4 DOM Level 3 Load and Save

La especificación DOM Level 3 Load and Save es una plataforma e interfaz de lenguaje neutral que permite cargar el contenido de un documento XML dentro de un documento DOM y serializar el documento DOM dentro de un documento XML. También permite filtrar contenido del documento en tiempo de carga o en tiempo de serialización<sup>1</sup>.

#### Tipos Básicos definidos en DOM Level 3 Load and Save

Para asegurar la interoperabilidad, esta especificación define los siguientes tipos básicos usados en varios módulos DOM (ver tabla 2-4).

*Tabla 2-4. Tipos Básicos definidos en DOM Level 3 Load and Save*

Tipo Básico	Descripción
LSInputStream	Representa una secuencia de bytes de entrada.
LSOutputStream	Representar una secuencia de bytes de salida.
LSReader	Representar una secuencia de caracteres de entrada en unidades de 16 bits (UTF-16).
LSWriter	Representar una secuencia de caracteres de salida en unidades de 16 bits (UTF-16).

---

<sup>1</sup> Serialización es el proceso de escribir el estado de un objeto en u flujo de bytes.





## Interfaces fundamentales de DOM Level 3 Load and Save

Las interfaces involucradas con el cargar y guardar documentos XML se describen en la tabla 2-5.

Tabla 2-5. Interfaces definidas por DOM Level 3 Load and Save

Interfaz	Descripción
DOMImplementationLS	Es una extensión de la interfaz DOMImplementation que proporciona métodos de fábrica para la creación de objetos que se requieren para cargar y guardar documentos.
LSParser	Es usada para analizar sintácticamente los datos contenidos en los documentos XML.
LSInput	Encapsula información acerca de los datos que están siendo cargados.
LSResourceResolver	Proporciona funciones para redirigir referencias a recursos externos cuando están siendo analizados sintácticamente.
LSParserFilter	Proporciona la habilidad de examinar y eliminar nodos cuando están siendo procesados mientras se analizan sintácticamente.
LSSerializer	Una interfaz usada para la serialización de documentos o nodos DOM.
LSOutput	Encapsula información sobre el destino para los datos de salida.
LSSerializerFilter	Proporciona la habilidad de examinar y filtrar nodos DOM cuando están siendo procesados para la serialización.

### Interfaz DOMImplementationLS

Esta interfaz contiene métodos de fábrica para la creación de objetos para cargar y guardar documentos (ver figura 2-8).

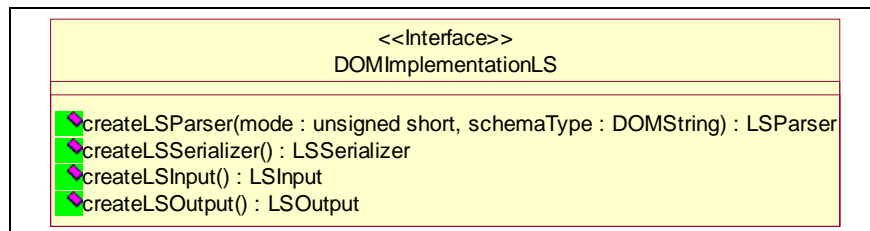


Figura 2-8. Interfaz DOMImplementationLS



Los métodos que ofrece esta interfaz permiten:

- Crear un nuevo objeto `LSParser`. El nuevo objeto construido puede ser configurado a través del objeto `DOMConfiguration` y puede usarse para analizar sintácticamente documentos por medio del método `parse`.
- Crear un nuevo objeto `LSSerializer`.
- Crear un objeto `LSInput` vacío.
- Crear un objeto `LSOutput` vacío.

### Interfaz `LSParser`

La interfaz `LSParser` proporciona un API para analizar sintácticamente documentos XML y construir la estructura DOM correspondiente (ver figura 2-9).

Una instancia `LSParser` puede obtenerse a través de la invocación del método `DOMImplementationLS.createLSParser()`.

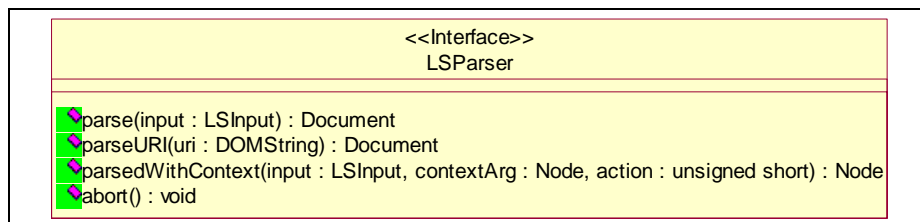


Figura 2-9. Interfaz `LSParser`

Los métodos que ofrece esta interfaz permiten:

- Analizar sintácticamente un documento XML representado por un objeto `LSInput`.
- Analizar un documento XML a partir de una referencia URI.
- Analizar sintácticamente un fragmento XML representado por un objeto `LSInput` e insertar el contenido dentro de un documento existente en la posición especificada con los argumentos `context` y `action`.



- Cancelar la carga del documento que esta siendo leído por el objeto `LSParser`. Si el objeto `LSParser` no esta actualmente ocupado, una llamada a este método no realiza ninguna operación.

### Interfaz `LInput`

La interfaz `LInput` representa un origen de entrada para los datos (ver figura 2-10). Permite encapsular la información de un origen de entrada en un objeto simple, el cual puede incluir a un un flujo de caracteres (`characterStream`), un flujo de bytes (`byteStream`), un identificador del sistema (`systemId`), identificador público (`publicId`), un identificador de recurso base (`baseURI`) o una posible codificación.

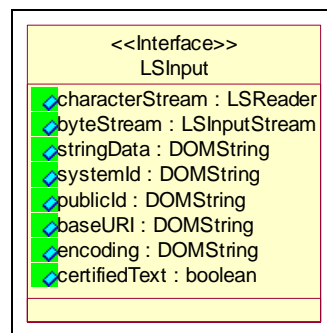


Figura 2-10. Interfaz `LInput`

Un objeto `LSParser` podría verificar en orden las siguientes entradas especificadas en el objeto `LInput` para determinar la manera en que van a ser leídos los datos:

1. `LInput.characterStream`
2. `LInput.byteStream`
3. `LInput.stringData`
4. `LInput.systemId`
5. `LInput.publicId`



La primera entrada que no sea nula y no sea una cadena vacía es la que será usada. Si todas las entradas son nulas, el objeto `LSParser` podría retornar un error.

### Interfaz `LSPOutput`

Esta interfaz representa un destino de salida para datos (ver figura 2-11). Permite encapsular la información de un destino de entrada en un objeto simple, el cual puede incluir a un flujo de caracteres (`characterStream`), un flujo de bytes (`byteStream`), un identificador del sistema (`systemId`) y una posible codificación.

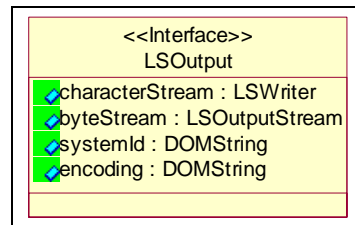


Figura 2-11. Interfaz `LSPOutput`

### Interfaz `LSParserFilter`

Esta interfaz proporciona la capacidad de examinar los nodos cuando están siendo construidos mientras se están analizando sintácticamente (ver figura 2-12). Durante la revisión cada nodo puede ser modificado o eliminado, o el análisis completo puede finalizarse anticipadamente.

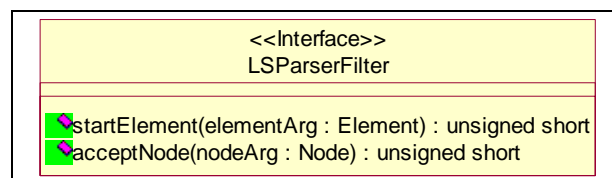


Figura 2-12. Interfaz `LSParserFilter`



### Interfaz LSProgressEvent

Esta interfaz representa un objeto de progreso de eventos que notifica a la aplicación sobre el progreso cuando se está analizando un documento (ver figura 2-13). Esta interfaz extiende de la interfaz `Event` definida en DOM Level 3 Events.

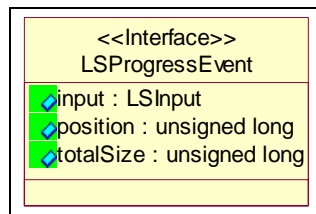


Figura 2-13. Interfaz `LSProgressEvent`

### Interfaz LSLoadEvent

Esta interfaz representa un objeto de carga de eventos que señala la finalización de la carga de un documento (ver figura 2-14).

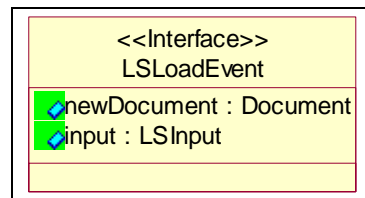


Figura 2-14. Interfaz `LSLoadEvent`

### Interfaz LSSerializer

La interfaz `LSSerializer` proporciona un API para la serialización de un documento DOM en un documento XML (ver figura 2-15). `LSSerializer` acepta cualquier tipo de nodo para la serialización.

La salida serializada para un nodo `Document` bien formado es un documento XML, pero la serialización de un nodo no siempre genera un documento XML bien formado y el objeto `LSParser` puede arrojar errores durante la serialización. Los errores pueden ocurrir



cuando cualquier ocurrencia de un caracter no puede representarse en la codificación de caracteres de salida.

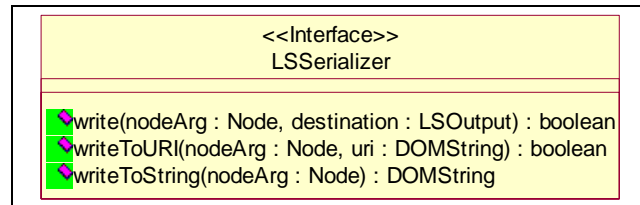


Figura 2-15. Interfaz *LSSerializer*

Los métodos que ofrece esta interfaz permiten:

- Serializar un nodo específico en un objeto *LSOutput* de salida para los datos.
- Serializar un nodo específico en un *DOMString*.

### Interfaz *LSSerializerFilter*

Proporciona a las aplicaciones la habilidad de examinar nodos cuando están siendo serializados y decidir que nodos podrían ser serializados o no. La interfaz *LSSerializerFilter* esta basada en la interfaz *NodeFilter* definida en DOM Level 2 Traversal and Range (ver figura 2-16).

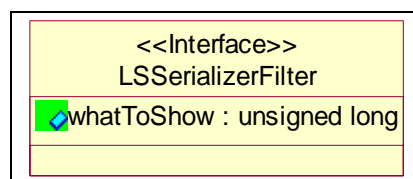


Figura 2-16. Interfaz *LSSerializerFilter*



## 2.2 Esquemas XML

El Esquema XML del Consorcio WWW (W3C) es una de las recomendaciones más importantes relacionadas con XML, donde se especifica un nuevo lenguaje de esquemas escrito en XML que permite restringir la estructura y los tipos de datos de los documentos XML. Surge con la necesidad de integrar XML con las tecnologías existentes más la necesidad de intercambiar información expresada en XML.

### 2.2.1 Documento esquema XML

Un documento esquema XML comienza con la declaración XML, seguida de la declaración del elemento `schema`, que es la raíz del documento en la que se incluyen todas las definiciones y declaraciones que determinan el contenido de un documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:x="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

Cada uno de los elementos en el documento esquema tiene un prefijo que lo asocia a uno de los espacios de nombres que aparecen en la declaración `schema`. Por convención se usa el prefijo `xs` para diferenciar el vocabulario del lenguaje esquema XML de otros vocabularios que estén siendo utilizados.

Uno de los usos más comunes de los esquemas es la validación de documentos XML, la cual asegura que el contenido de los documentos este conforme al conjunto de reglas dado.

Un lenguaje de esquema XML es una formalización de restricciones, expresadas como un conjunto de reglas o modelo de estructura que se aplican a una clase de documentos XML, denominadas instancias.

### 2.2.2 Modelo de Datos de un esquema XML

Un esquema XML puede ser descrito en términos de un modelo de datos a nivel conceptual, el cual está constituido por componentes. Un componente de un esquema es el



término genérico empleado para describir los bloques de construcción que comprenden el modelo de datos de un esquema XML (ver figura 2-17).

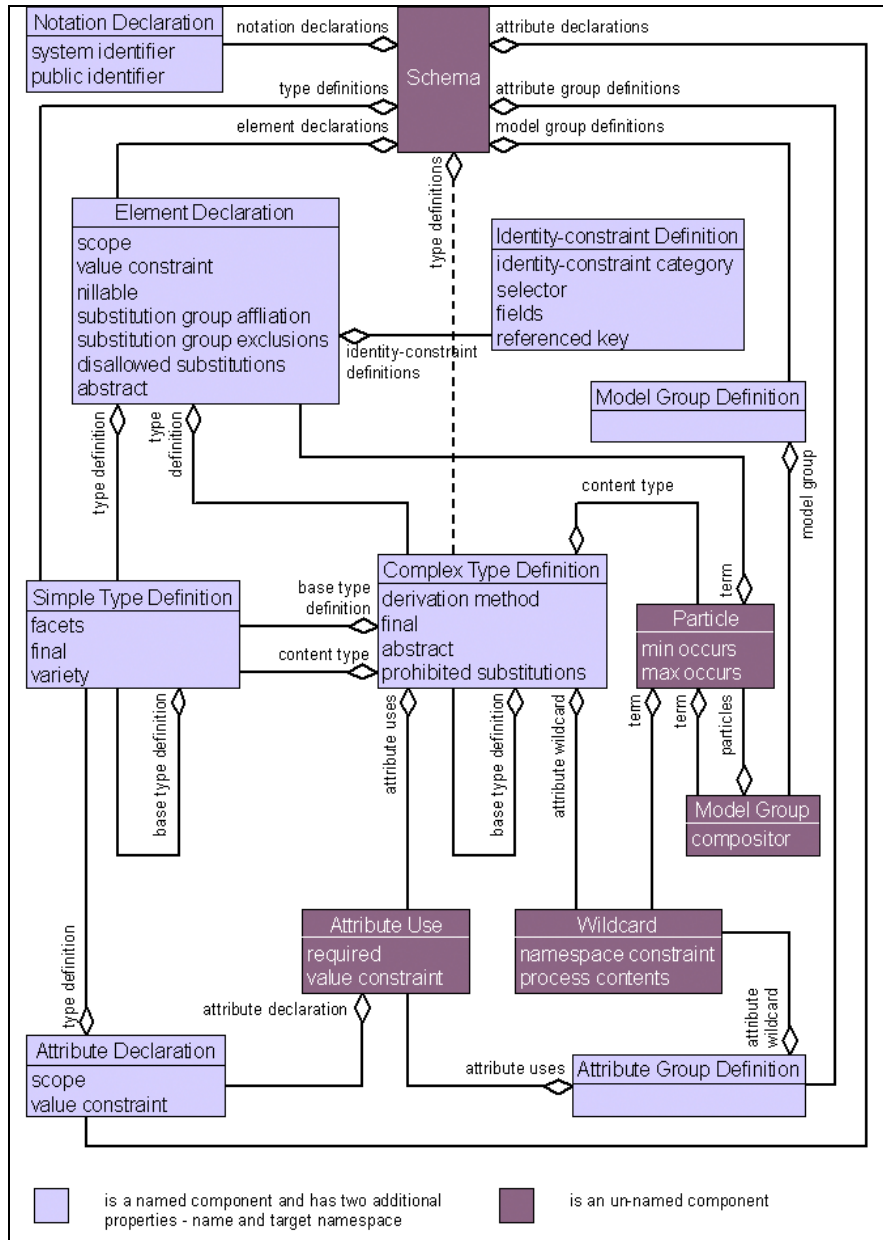


Figura 2-17. Modelo de Datos de un esquema XML [9]

Un esquema XML es un conjunto de componentes. Existen trece tipos de componentes en total, que se dividen en tres grupos: componentes primarios, secundarios y de ayuda.





- Componentes primarios:
  - Definiciones de tipos simple.
  - Definiciones de tipo complejo.
  - Declaraciones de atributos.
  - Declaraciones de elementos.
- Componentes secundarios:
  - Definiciones de grupo de atributos.
  - Definiciones de restricciones de identidad.
  - Definiciones de grupo de modelos.
  - Declaraciones de notaciones.
- Componentes de ayuda:
  - Anotaciones.
  - Grupo de modelos.
  - Partículas.
  - Comodines.
  - Uso de atributos.

### 2.2.3 Tipos de Datos

La recomendación Esquema XML: Tipos de Datos [10], es la segunda parte de la especificación del lenguaje Esquema XML en la que se especifican los fundamentos para la definición de los tipos de datos usados por la recomendación Esquema XML.

Los tipos de datos definidos por esta especificación se dividen en dos categorías: primitivos y derivados. Los tipos de datos primitivos son aquellos que no están definidos en términos de otros tipos de datos. En cambio, los tipos de datos derivados son aquellos que sí están definidos en términos de otros tipos de dato.

La recomendación define un tipo de dato conceptual, llamado `anySimpleType`, que puede ser considerado como el tipo base de todos los tipos primitivos y cuyo espacio de valores esta formado por la unión de los espacios de valores de todos los tipos de dato primitivos.



Los tipos de datos primitivos y derivados pueden usarse para derivar otros tipos de datos, los cuales son definidos por el diseñador de esquemas. Un tipo de dato derivado por el usuario puede definirse de tres maneras:

1. Mediante la asignación de una propiedad no fundamental (§2.2.4.1), la cual sirve para restringir el espacio de valores del tipo base.
2. Mediante la creación de un tipo de dato lista (§2.2.5.2), cuyo espacio de valores consiste de una secuencia de longitud finita de valores del mismo tipo.
3. Mediante la creación de un tipo de dato unión (§2.2.5.3), cuyo espacio de valores consiste de la unión de espacios de valores de diferentes tipos.

La especificación introduce un desacoplamiento entre el dato que puede ser leído de los documentos instancias XML (el espacio léxico) y el valor que es interpretado acorde al tipo de dato (el espacio de valores). De este modo, cada tipo de dato tiene un espacio léxico y un espacio de valores propio y un conjunto de reglas para asociar una representación léxica con un valor perteneciente al espacio de valores.

Un tipo de dato es una tupla que consiste de un conjunto de valores distintos llamado espacio de valores, un conjunto de representaciones léxicas llamado espacio léxico y un conjunto de restricciones que caracterizan las propiedades del espacio de valores.

La figura 2-18 describe la jerarquía de los tipos de datos definidos por la especificación.

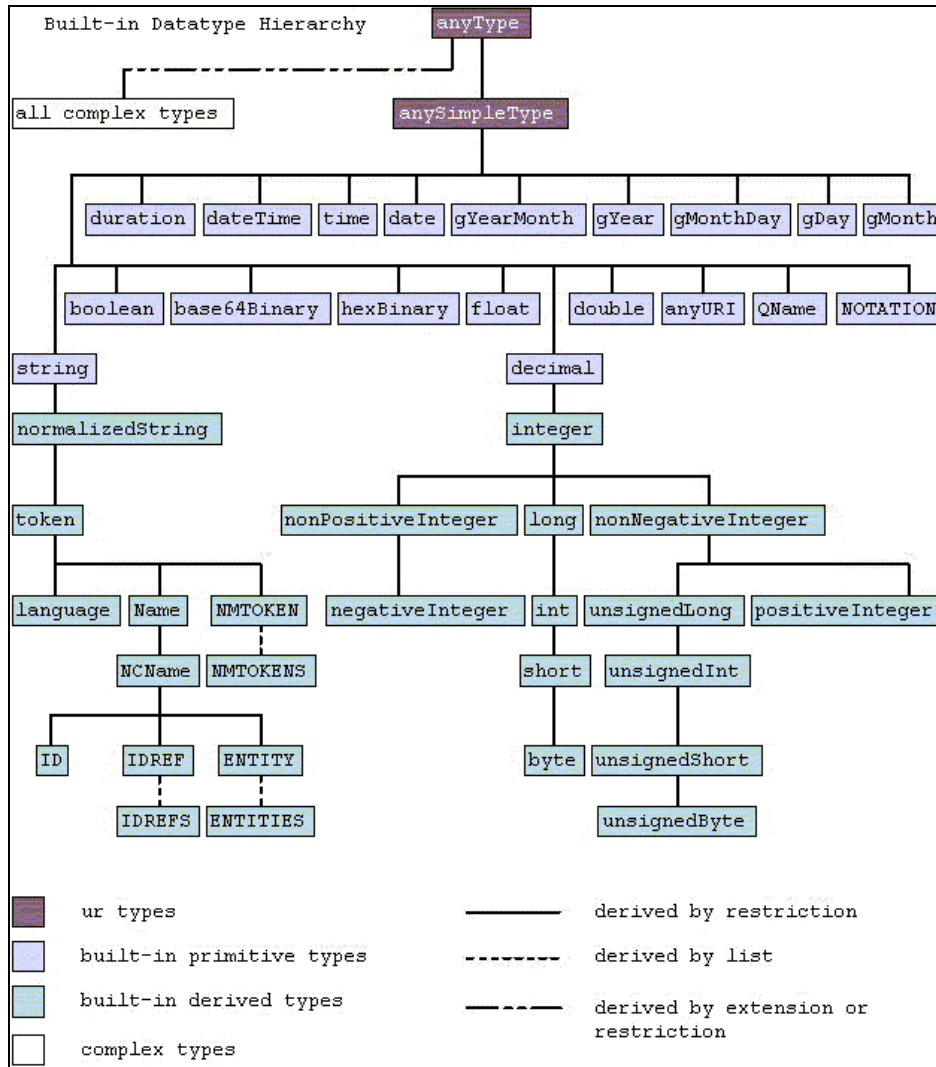


Figura 2-18. Tipos de datos definidos por la especificación Esquema XML [10]

## Espacio de Valores

El espacio de valores es el conjunto de valores para un tipo de dato. Cada valor en el espacio de valores de un tipo de dato es denotado por uno o más literales en el espacio léxico.



Un espacio de valores tiene ciertas propiedades (de cardinalidad, de igualdad o de orden) mediante las cuales los valores individuales dentro del espacio de valores pueden ser comparados con cualquier otro valor.

### Espacio Léxico

El espacio léxico es el conjunto de literales válidos para un tipo de dato. La mayoría de los tipos de datos definidos en la especificación tienen una representación léxica simple, donde cada valor en el espacio de valores es denotado por un único literal en el espacio léxico; pero éste no siempre es el caso. Por ejemplo, 100 y 1.0E2 son dos literales diferentes del espacio léxico de un tipo de dato flotante (float) donde ambos denotan el mismo valor.

Una representación léxica canónica es un conjunto de literales entre el conjunto de literales válido para un tipo de dato tal que existe una conversión uno a uno entre los literales en la representación léxica canónica y los valores en el espacio de valores.

#### 2.2.4 Propiedades de los Tipos de Datos

Una propiedad de un tipo de dato es una definición simple de un aspecto de un espacio de valores. En general, cada restricción caracteriza un espacio de valores a lo largo de ejes independientes o dimensiones.

Las propiedades sirven para distinguir aquellos aspectos de un tipo de dato que lo diferencian de otros tipos de dato. Así, los tipos de datos son definidos en términos de propiedades de valores, las cuales determinan el espacio de valores y las propiedades del tipo de dato.

Existen dos tipos de propiedades: Las propiedades fundamentales, que definen el tipo de dato y las propiedades no fundamentales o restricciones, que limitan los valores permitidos por un tipo de dato.



### Propiedades fundamentales

Una propiedad fundamental es una propiedad abstracta que sirve para caracterizar semánticamente los valores en el espacio de valores.

Las propiedades fundamentales de un tipo de dato determinan las operaciones que pueden efectuarse sobre los valores de un tipo de dato. La tabla 2-6 describe cinco propiedades fundamentales que pueden aplicarse a todos los espacios de valores.

Tabla 2-6. Propiedades fundamentales

Propiedad fundamental	Descripción
equal	Permite hacer comparaciones entre valores para determinar o no la igualdad ( $X=Y$ ó $X \neq Y$ ).
ordered	Indica si una relación de orden (parcial o total) está definida sobre un espacio de valores.
bounded	Indica si un espacio de valores está acotado.
cardinality	Define el número valores permitidos en un espacio de valores. La cardinalidad puede ser cero, uno o ilimitada, o algún número entero positivo específico.
numeric	Indica si un espacio de valores es numérico.

### Propiedades no fundamentales o restricciones

Una propiedad no fundamental o restricción es una propiedad opcional que puede aplicarse a un tipo de dato para restringir su espacio de valores.

Las restricciones aplicadas al espacio de valores restringen consecuentemente el espacio léxico. La tabla 2-7 describe doce propiedades no fundamentales.

Tabla 2-7. Propiedades no fundamentales o restricciones

Propiedad no fundamental	Descripción
length	Restringe un espacio de valores a valores con un número específico de unidades de longitud.
minLength	Restringe un espacio de valores a valores que tengan por lo menos un número específico de unidades de longitud.
maxLength	Restringe un espacio de valores a valores que tengan a lo más un número



	específico de unidades de longitud.
pattern	Restringe un espacio de valores a aquellos que están denotados por literales, los cuales coinciden con una expresión regular específica.
enumeration	Restringe un espacio de valores a un conjunto de valores específicos.
whiteSpace	Restringe un espacio de valores según las reglas de normalización de espacios en blanco.
minInclusive	Límite inferior del rango en el espacio de valores.
maxInclusive	Límite superior del rango en el espacio de valores.
minExclusive	Límite inferior del rango en el espacio de valores, excluyéndolo a el.
maxExclusive	Límite superior del rango en el espacio de valores, excluyéndolo a el.
totalDigits	Restringe un espacio de valores a valores con un número máximo específico de dígitos decimales.
fractionDigits	Restringe un espacio de valores a un número máximo de dígitos decimales en la parte fraccionaria.

### 2.2.5 Definiciones de tipos simples

El modelo abstracto de datos proporciona dos tipos de componentes para las definiciones de tipo: simple y complejo.

Los nuevos tipos simples se definen a partir de los tipos simples existentes (predefinidos o derivados). En particular, un nuevo tipo simple se deriva de un tipo simple existente restringiendo el rango existente de valores.

Existen tres métodos de derivación que son:

1. *Derivación por restricción*: Las restricciones impuestas sobre un tipo de dato no cambian su semántica original o significado.
2. *Derivación por lista*: Los nuevos tipos de datos son definidos como una lista de valores de cierto tipo de dato. El nuevo tipo toma la semántica del tipo de dato de la lista.
3. *Derivación por unión*: Los nuevos tipos de datos definidos permiten valores de un conjunto de otros tipos de datos, perdiendo de esta manera su semántica.



La representación XML de un tipo simple se hace mediante el elemento `simpleType`, el cual tiene la sintaxis mostrada en la tabla 2-8.

Tabla 2-8. Sintaxis del elemento `simpleType`

Elemento <code>simpleType</code>
<pre>&lt;simpleType   final = (#all   (list   union   restriction))   id = ID   name = NCName   {any attributes with non-schema namespace . . .}&gt;   Content: (annotation?, (restriction   list   union)) &lt;/simpleType&gt;</pre>
Atributos
<ul style="list-style-type: none"><li>▪ <code>default</code>: El valor por omisión del atributo</li><li>▪ <code>fixed</code>: Especifica un valor fijo para el atributo. <code>default</code> y <code>fixed</code> son mutuamente exclusivos.</li><li>▪ <code>form</code>: Especifica si el atributo es calificado (si tiene un prefijo de espacio de nombres) o no.</li><li>▪ <code>id</code>: Identificador del elemento.</li><li>▪ <code>name</code>: Nombre del atributo.</li><li>▪ <code>ref</code>: Nombre del atributo global que esta siendo referenciado.</li><li>▪ <code>type</code>: tipo de dato del atributo.</li><li>▪ <code>use</code>: Uso del atributo.</li></ul>

### Derivación por restricción

Una derivación por restricción se hace usando el elemento `restriction`, donde las restricciones se definen a través de las propiedades del tipo de dato. El tipo de dato sobre el cual se aplica la restricción se denomina tipo base y puede ser referenciado a través del atributo `base` o definirse en el elemento `restriction`.

Un elemento `restriction` permite añadir restricciones o propiedades no fundamentales a los tipos simples. Todos los tipos simples, a excepción de aquellos que representan listas y uniones, contienen restricciones (ver tabla 2-9).



Tabla 2-9. Sintaxis del elemento *restriction*

Elemento <i>restriction</i>
<pre>&lt;restriction   base = QName   id = ID   {any attributes with non-schema namespace ...}&gt;   Content: (annotation?, (simpleType?, (minExclusive   minInclusive   maxExclusive   maxInclusive   totalDigits   fractionDigits   length   minLength   maxLength   enumeration   whiteSpace   pattern)*)) &lt;/restriction&gt;</pre>
Atributos
<ul style="list-style-type: none"> <li>▪ <i>base</i>: Nombre calificado del tipo base.</li> <li>▪ <i>id</i>: Identificador del elemento.</li> </ul>

### Derivación por lista

La derivación por lista es el mecanismo por el cual un tipo de dato lista puede derivarse de un tipo de dato atómico. Todos los ítems de la lista deben ser del mismo tipo de dato.

El elemento *list* permite realizar la definición de un tipo de dato lista mediante derivación de un tipo de dato existente o incluyendo la definición de un tipo simple, pero en la definición no pueden mezclarse ambas sintaxis (ver tabla 2-10).

Tabla 2-10. Sintaxis del elemento *list*

Elemento <i>list</i>
<pre>&lt;list   id = ID   itemType = QName   {any attributes with non-schema namespace ...}&gt;   Content: (annotation?, (simpleType?)) &lt;/list&gt;</pre>
Atributos
<ul style="list-style-type: none"> <li>▪ <i>id</i>: Identificador del elemento.</li> </ul>





- itemType: Tipo de dato de la lista.

La definición de un tipo de dato lista a partir de un tipo de dato existente se hace a través del atributo itemType.

### Derivación por unión

La derivación por unión permite definir un tipo de dato mediante la unión de varios tipos existentes y de tipos predefinidos por el usuario.

El elemento union permite realizar la definición de un tipo de dato unión mediante derivación de varios tipos de datos existentes o incluyendo definiciones de tipo simple. En la definición pueden mezclarse ambas sintaxis (ver tabla 2-11).

Tabla 2-11. Sintaxis del elemento union

Elemento union
<pre>&lt;union   id = ID   memberTypes = List of QName   {any attributes with non-schema namespace ...}&gt;   Content: (annotation?, (simpleType*)) &lt;/union&gt;</pre>
Atributos
<ul style="list-style-type: none"> <li>▪ id: Identificador del elemento.</li> <li>▪ memberTypes: Lista de tipo de datos.</li> </ul>

La definición de un tipo de dato unión a partir de un tipo de dato existente se hace a través del atributo memberTypes.



### 2.2.6 Definiciones de tipos complejos

La definición de un tipo complejo generalmente contiene una o más declaraciones de elementos, referencias a elementos y declaraciones de atributos. Los elementos y los atributos son declarados usando los elementos `element` y `attribute`, respectivamente.

La representación XML de una definición de tipo complejo se hace mediante el elemento `complexType`, el cual tiene la siguiente sintaxis mostrada en la tabla 2-12.

Tabla 2-12. Sintaxis del elemento `complexType`

Elemento <code>complexType</code>
<pre>&lt;complexType   abstract = boolean : false   block = (#all   List of (extension   restriction))   final = (#all   List of (extension   restriction))   id = ID   mixed = boolean : false   name = NCName   {any attributes with non-schema namespace . . .}&gt;   Content: (annotation?, (simpleContent   complexContent   ((group   all   choice   sequence)?, ((attribute   attributeGroup)*, anyAttribute?)))) &lt;/complexType&gt;</pre>
Atributos
<ul style="list-style-type: none"> <li>▪ <code>abstract</code>: Determina si el tipo complejo es explícitamente no instanciable.</li> <li>▪ <code>block</code>: Determina la capacidad de una instancia XML para especificar una derivación instanciable de un tipo complejo no instanciable.</li> <li>▪ <code>final</code>: Determina si el tipo complejo puede ser derivado por extensión o por restricción para crear nuevos tipos complejos.</li> <li>▪ <code>id</code>: Identificador del elemento.</li> <li>▪ <code>mixed</code>: Define si el modelo de contenido es un modelo de contenido mixto.</li> <li>▪ <code>name</code>: El nombre del tipo complejo.</li> </ul>



Las definiciones de tipos complejos son identificadas por su nombre y espacio de nombres, a excepción de las definiciones de tipos complejos anónimas, dado que las definiciones de tipos, ya sean simples o complejas, deben ser identificadas exclusivamente en un esquema XML.

### Modelos de Contenido Simple

Los tipos complejos con modelo de contenido simple se crean añadiendo una lista de atributos a un tipo simple.

El elemento `simpleContent` permite especificar atributos para los tipos simples. Este elemento también puede extender o restringir atributos a partir de tipos complejos que tengan un modelo de contenido simple (ver tabla 2-13).

Tabla 2-13. Sintaxis del elemento `simpleContent`

Elemento <code>simpleContent</code>
<pre>&lt;simpleContent   id = ID   {any attributes with non-schema namespace . . .}&gt;   Content: (annotation?, (restriction   extension)) &lt;/simpleContent&gt;</pre>
Atributos
<ul style="list-style-type: none"><li>▪ <code>id</code>: Identificador del elemento.</li></ul>

### Modelos de Contenido Complejo

Los tipos complejos con un modelo de contenido complejo se crean por la definición de una lista de elementos y atributos.

El elemento `complexContent` permite definir modelo de contenido complejo por derivación de un tipo complejo (ver tabla 2-14).



Tabla 2-14. Sintaxis del elemento `complexContent`

Elemento <code>complexContent</code>
<pre>&lt;complexContent   id = ID   mixed = boolean   {any attributes with non-schema namespace . . .}&gt;   Content: (annotation?, (restriction   extension)) &lt;/complexContent&gt;</pre>
Atributos
<ul style="list-style-type: none"><li>▪ <code>id</code>: Identificador del elemento.</li><li>▪ <code>mixed</code>: Determina la capacidad del tipo complejo de soportar contenido mixto.</li></ul>

### 2.2.7 Declaración de Atributos

Los atributos son parte de los elementos. Cada atributo tiene un nombre y un valor. Un atributo se declara en un documento esquema XML mediante el elemento `attribute`, el cual tiene la siguiente sintaxis mostrada en la tabla 2-15.

Tabla 2-15. Sintaxis del elemento `attribute`

Elemento <code>attribute</code>
<pre>&lt;attribute   default = string   fixed = string   form = (qualified   unqualified)   id = ID   name = NCName   ref = QName   type = QName   use = (optional   prohibited   required) : optional   {any attributes with non-schema namespace . . .}&gt;   Content: (annotation?, (simpleType?)) &lt;/attribute&gt;</pre>



Atributos
<ul style="list-style-type: none"> <li>▪ <code>default</code>: El valor por omisión del atributo</li> <li>▪ <code>fixed</code>: Especifica un valor fijo para el atributo. <code>default</code> y <code>fixed</code> son mutuamente exclusivos.</li> <li>▪ <code>form</code>: Especifica si el atributo es calificado (si tiene un prefijo de espacio de nombres) o no.</li> <li>▪ <code>id</code>: Identificador del elemento.</li> <li>▪ <code>name</code>: Nombre del atributo.</li> <li>▪ <code>ref</code>: Nombre del atributo global que está siendo referenciado.</li> <li>▪ <code>type</code>: tipo de dato del atributo.</li> <li>▪ <code>use</code>: Uso del atributo.</li> </ul>

Todos los atributos declarados en el nivel superior del esquema son considerados atributos globales. Los atributos globales pueden ser referenciados a través de su nombre en cualquier parte del esquema. Los atributos locales no pueden ser referenciados.

### 2.2.8 Declaración de Elementos

Los elementos son los componentes principales de los documentos XML. Cada elemento tiene un nombre, un conjunto de atributos y contenido.

Un elemento es representado en un documento esquema XML mediante el elemento `element`, el cual tiene la sintaxis mostrada en la tabla 2-16.

Tabla 2-16. Sintaxis del elemento `element`

Elemento <code>element</code>
<pre> &lt;element   abstract = boolean : false   block = (#all   List of (extension   restriction   substitution))   default = string   final = (#all   List of (extension   restriction))   fixed = string   form = (qualified   unqualified)   id = ID   maxOccurs = (nonNegativeInteger   unbounded) : 1 </pre>



```
minOccurs = nonNegativeInteger : 1
name = NCName
nillable = boolean : false
ref = QName
substitutionGroup = QName
type = QName
{any attributes with non-schema namespace . . .}>
Content: (annotation?, ((simpleType | complexType)?, (unique | key | keyref)*))
</element>
```

### Atributos

- **abstract**: Controla si el elemento puede ser usado directamente en un documento instancia.
- **block**: Controla si el elemento puede estar sujeto a un tipo o grupo de sustitución.
- **default**: El valor por omisión del elemento.
- **final**: Controla si el elemento puede ser usado como la cabeza de un grupo de sustitución por elementos cuyos tipos derivan por extensión o restricción del tipo del elemento.
- **fixed**: Usando este atributo el elemento de contenido simple puede fijarse a un valor específico.
- **id**: Identificador del esquema.
- **name**: Nombre del elemento (no lleva el prefijo del espacio de nombres).
- **nillable**: Cuando su valor es verdadero, el elemento puede ser declarado como nulo.
- **substitutionGroup**: Nombre calificado de la cabeza del grupo de sustitución.
- **type**: Nombre calificado de un tipo simple o complejo (debe omitirse en las definiciones locales de tipo simple y compleja).

Todos los elementos definidos en el nivel superior de un esquema son considerados elementos globales. Los elementos declarados globalmente pueden ser referenciados en cualquier parte del esquema.



## 2.3 Esquemas BDOR

El diseño general de una base de datos se denomina esquema de la base de datos. Los sistemas de bases de datos tienen varios esquemas. El esquema físico describe el diseño en el nivel físico, en donde se describe cómo se almacenan físicamente los datos. El esquema lógico describe el diseño de la base de datos en el nivel lógico, en el cual se describe qué datos se almacenan en la base de datos y qué relaciones existen entre esos datos. Además, una base de datos también puede tener varios esquemas en el nivel de vistas, en el que solo se describe parte de la base de datos.

Un sistema de bases de datos proporciona un lenguaje de definición de datos (LDD) para especificar el esquema de la base de datos y un lenguaje de manipulación de datos (LMD) para expresar consultas y modificaciones a la bases de datos.

### 2.3.1 SQL (Structured Query Language)

El lenguaje estructurado de consultas SQL es una herramienta usada para interactuar con un tipo específico de bases de datos, llamadas bases de datos relacionales.

El estándar oficial para SQL fue publicado inicialmente en 1986 por el Instituto Nacional Americano de Estandarización (ANSI) y la Organización Internacional de Estandarización (ISO) y fue revisado y expandido en 1989 (SQL1) y de nuevo en 1992 (SQL2). La versión más reciente es SQL:1999, la cual es un superconjunto de la norma SQL-92.

SQL es usado para controlar todas las funciones que debe proporcionar todo SGBD [Groff y Weingberg 1999], entre las que se encuentran:

- *Definición de datos:* SQL permite definir la estructura y la organización de los datos almacenados y sus relaciones.
- *Recuperación de datos:* SQL permite recuperar datos de la bases de datos.
- *Manipulación de datos:* SQL permite actualizar, insertar o eliminar datos en la base de datos.



- *Control de acceso:* SQL permite restringir la capacidad de un usuario de recuperar, añadir y modificar datos, protegiendo los datos almacenados de los accesos no autorizados.
- *Distribución de datos:* SQL es usado para coordinar los datos compartidos por usuarios concurrentes, asegurándose que éstos no interfieran con otros.
- *Integridad de datos:* SQL define restricciones de integridad en la base de datos, protegiéndola de corrupción debido a actualizaciones inconsistentes o fallas del sistema.

El lenguaje SQL está estructurado en tres sublenguajes: El lenguaje de definición de datos (LDD), el lenguaje de manipulación de datos (LMD) y el lenguaje de control de datos (LCD). En las secciones posteriores se refiere al lenguaje SQL basado en la norma SQL-92 (SQL2), ya que la mayoría de los SGBD es compatible con esta norma.

### Sentencias

El cuerpo principal del lenguaje SQL consiste de acerca 40 sentencias, las cuales son resumidas en la tabla 2-17. Cada sentencia requiere de una acción específica del sistema de gestión de bases de datos (SGBD).

Tabla 2-17. Principales Sentencias SQL [Groff y Weingberg 1999]

Sentencia	Descripción
<b>Manipulación de datos</b>	
SELECT	Recupera datos de la base de datos
INSERT	Añade nuevas filas de datos a la base de datos
DELETE	Elimina filas de datos de la base de datos
UPDATE	Modifica datos existentes en la base de datos
<b>Definición de datos</b>	
CREATE TABLE	Crea una nueva tabla en la base de datos
DROP TABLE	Elimina una tabla de la base de datos





ALTER TABLE	Cambia la estructura de una tabla existente
CREATE VIEW	Añade una nueva vista a la base de datos
DROP VIEW	Elimina una vista de la base de datos
CREATE INDEX	Construye un índice para una columna
DROP INDEX	Elimina el índice para una columna
CREATE SCHEMA	Añade un nuevo esquema a la base de datos
CREATE DOMAIN	Añade un nuevo dominio de valores de datos
ALTER DOMAIN	Cambia una definición de dominio
DROP DOMAIN	Remueve un dominio de la base de datos
 <i>Control de acceso</i>	
GRANT	Otorga privilegios de acceso a un usuario
REVOKE	Elimina privilegios de acceso a un usuario
 <i>Control Transaccional</i>	
COMMIT	Finaliza la transacción actual
ROLLBACK	Aborta la transacción actual
SET TRANSACTION	Define características de acceso a los datos de la transacción actual
 <i>SQL Programático</i>	
DECLARE	Define un cursor para una consulta
EXPLAIN	Describe el plan de acceso para una consulta
OPEN	Abre un cursor para recuperar los resultados de una consulta
FETCH	Cierra un cursor
CLOSE	Prepara una sentencia SQL para ejecución dinámica
DESCRIBE	Describe una consulta preparada

Cada sentencia SQL comienza con un verbo, una palabra que describe lo que hace la sentencia. La sentencia puede continuar con una o más cláusulas. Cada cláusula comienza con una palabra reservada. Algunas cláusulas son opcionales y otras requeridas. Muchas cláusulas contienen nombres de tablas o columnas; algunas pueden contener palabras adicionales, constantes o expresiones [Groff y Weingberg 1999].



### Tipos de Datos

El estándar ANSI/ISO SQL especifica varios tipos de datos que pueden ser almacenados en una base de datos basada en SQL y manipulados por el lenguaje SQL (ver tabla 2-18). El estándar original SQL-89 define solamente un conjunto mínimo de tipos de datos. El estándar SQL-92 expande esta lista para incluir tipos de datos de fecha y hora, cadenas de caracteres de longitud variable y cadenas de bits, entre otros [Groff y Weingberg 1999].

Tabla 2-18. Tipos de Datos ANSI/ISO SQL92 [Groff y Weingberg 1999]

Tipo de Dato	Descripción
CHAR CHARACTER(len)	Cadena de caracteres de longitud fija
VARCHAR(len) CHAR VARYING(len) CHARACTER VARYING(len)	Cadena de caracteres de longitud variable*
NCHAR(len) NATIONAL CHAR(len) NATIONAL CHARACTER	Cadena de caracteres nacional de longitud fija*
NATIONAL VARYING(len) NATIONAL CHAR VARYING(len) NATIONAL CHARACTER VARYING(len)	Cadena de caracteres nacional de longitud variable*
INTEGER INT	Números enteros
SMALLINT	Números enteros cortos
BIT(len)	Cadena de bits de longitud fija*
BIT VARYING(len)	Cadena de bits de longitud variable*



NUMERIC(precisión-escala)	Números decimales
DECIMAL(precisión-escala)	
DEC(precisión-escala)	
FLOAT(precisión)	Números punto flotante
REAL	Números punto flotante de baja precisión
DOUBLE PRECISION	Números punto flotante de alta precisión
DATE	Fecha calendario*
TIME(precisión)	Hora de reloj*
TIMESTAMP(precisión)	Fecha y hora*
INTERVAL	Intervalo de tiempo*
*nuevos tipos de datos en SQL-92	

### 2.3.2 Lenguaje de Definición de Datos (LDD)

El LDD incluye las declaraciones que se usan para la creación, modificación y destrucción de objetos en la base de datos [Houlette 2003].

Las sentencias de definición de datos especificadas en el estándar SQL-92 definen la estructura de una base de datos, incluyendo sus tablas, vistas y los objetos que contiene un SGBD.

La sintaxis utilizada usa las siguientes convenciones:

- Las palabras SQL reservadas aparecen en MAYÚSCULAS.
- Los elementos de la sintaxis aparecen encerrados entre paréntesis angulados (<>).
- Un elemento seguido de la palabra `list` indica una lista de elementos separados por comas.



- La barra vertical (|) indica una selección entre dos o más alternativas.
- Los elementos de la sintaxis encerrados entre corchetes ([ ]) son opcionales.

### CREATE TABLE

La sentencia CREATE TABLE, define una nueva tabla en la base de datos y la prepara para admitir datos. Varias cláusulas de esta sentencia especifican elementos de la definición de tablas [Groff y Weingberg 1999]. La sintaxis para esta sentencia se muestra en la tabla 2-19.

Tabla 2-19. Sintaxis de la sentencia CREATE TABLE.

<pre>CREATE TABLE [ { GLOBAL   LOCAL } TEMPORARY ] &lt;table name&gt; &lt;table element list&gt; [ ON COMMIT { DELETE   PRESERVE } ROWS ]</pre>	
<table element>	<pre>&lt;column definition&gt;   &lt;table constraint definition&gt; column data-type [DEFAULT {literal   USER   NULL}] [column constraint list]</pre>
<column definition>	<pre>&lt;column name&gt; { &lt;data type&gt;   &lt;domain name&gt; } [DEFAULT] [ &lt;column constraint definition&gt; ] [ COLLATE &lt;collation name&gt; ]</pre>
<column constraint definition>	<pre>[CONSTRAINT &lt;constraint name&gt;] {NOT NULL   UNIQUE   PRIMARY KEY}   &lt;references specification&gt;   &lt;check constraint definition&gt; [ &lt;constraint attributes&gt; ]</pre>
<references specification>	<pre>REFERENCES &lt;table name&gt; [ (&lt;column name list&gt;) ] [ MATCH FULL   PARTIAL ] [ &lt;referential triggered action&gt; ]</pre>
<check constraint definition>	<pre>CHECK ( &lt;search condition&gt; )</pre>



<constraint attributes>	INITIALLY DEFERRED   INITIALLY IMMEDIATE [ [NOT] DEFERRABLE ]   [NOT] DEFERRABLE [INITIALLY DEFERRED INITIALLY IMMEDIATE]
<referential triggered action>	ON UPDATE <referential action> [ ON DELETE <referential action> ]   ON DELETE <referential action> [ ON UPDATE <referential action> ]
<referential action>	CASCADE   SET NULL   SET DEFAULT   NO ACTION
<table constraint definition>	[CONSTRAINT <constraint name>] <table constraint> [<constraint attributes>]
<table constraint>	<unique constraint definition>   <referential constraint definition>   <check constraint definition>
<unique constraint definition>	UNIQUE   PRIMARY KEY ( <column name list> )
<referential constraint definition>	FOREIGN KEY ( <column name list> ) <references specification>

### DROP TABLE

La sentencia DROP TABLE es usada para eliminar tablas de la base de datos. El nombre de la tabla en la sentencia indica el nombre de la tabla a ser eliminada (ver tabla 2-10).

Cuando se usa la sentencia DROP TABLE para eliminar una tabla, su definición y todo su contenido se pierde y luego no hay forma de recuperar los datos [Groff y Weingberg 1999].

Tabla 2-20. Sintaxis de la sentencia DROP TABLE.

<b>DROP TABLE &lt;table name&gt; &lt;drop behavior&gt;</b>	
<drop behavior>	CASCADE   RESTRICT



## ALTER TABLE

La sentencia ALTER TABLE permite efectuar los siguientes cambios en la estructura de una tabla:

- Añadir una definición de columna a una tabla.
- Eliminar una columna de una tabla.
- Cambiar el valor por omisión de una columna.
- Añadir o eliminar una clave principal para una tabla.
- Añadir o eliminar una nueva clave foránea para una tabla.
- Añadir o eliminar una restricción única para una tabla.
- Añadir o eliminar una restricción de verificación para una tabla.

La sintaxis para esta sentencia se muestra en la tabla 2-21.

Tabla 2-21. Sintaxis de la sentencia ALTER TABLE.

ALTER TABLE <table name> <alter table action>	
<alter table action>	<add column definition>   <alter column definition>   <drop column definition>   <add table constraint definition>   <drop table constraint definition>
<add column definition>	ADD [ COLUMN ] <column definition>
<alter column definition>	ALTER [ COLUMN ] <column name> <alter column action>
<drop column definition>	DROP [ COLUMN ] <column name> <drop behavior>
<add table constraint definition>	ADD <table constraint definition>
<drop table constraint definition>	DROP CONSTRAINT <constraint name> <drop behavior>
<alter column action>	SET DEFAULT   DROP DEFAULT
<drop behavior>	CASCADE   RESTRICT



<code>&lt;table constraint definition&gt;</code>	<code>[CONSTRAINT &lt;constraint name&gt;]</code>
	<code>&lt;table constraint&gt; [&lt;constraint attributes&gt;]</code>

### CREATE VIEW

La sentencia `CREATE VIEW` es usada para crear una vista. La sentencia asigna un nombre a la vista y especifica la consulta que define la vista. Para crear una vista con éxito, se debe tener permiso para acceder todas las tablas referenciadas en la consulta (ver tabla 2-22).

Tabla 2-22. Sintaxis de la sentencia `CREATE VIEW`.

```
CREATE VIEW <table name> [ (<column name list> ) ]
AS <query expression>
[ WITH [ <levels clause> ] CHECK OPTION ]
<levels clause> CASCADED | LOCAL
```

### DROP VIEW

A diferencia del estándar SQL-1, el estándar SQL-2 soporta la eliminación de vistas a través de la sentencia `DROP VIEW`. También proporciona control detallado en casos donde un usuario intenta eliminar una vista cuando la definición de otra vista depende de ésta. La sintaxis para esta sentencia se muestra en la tabla 2-23.

Tabla 2-23. Sintaxis de la sentencia `DROP VIEW`.

<code>DROP VIEW &lt;table name&gt; &lt;drop behavior&gt;</code>
<code>&lt;drop behavior&gt;</code> CASCADE   RESTRICT

### CREATE INDEX

El estándar SQL-2 no habla sobre índices o cómo crearlos. Sin embargo, el uso de índices es esencial para lograr un desempeño adecuado en cualquier base de datos de tipo empresarial.

Los SGBD como Oracle, MS SQLServer, MySQL y PostgreSQL soportan índices a través de la sentencia `CREATE INDEX`, la cual difiere del manejador (ver tabla 2-24).



Tabla 2-24. Sintaxis de la sentencia `CREATE INDEX` usada por MySQL

```
CREATE [UNIQUE|FULLTEXT] INDEX <index name>
ON <table name> (<column name>[(length)],... )
```

### DROP INDEX

La sentencia `DROP INDEX` es usada para eliminar un índice que ha sido creado sobre una tabla (ver tabla 2-25).

Tabla 2-25. Sintaxis de la sentencia `DROP INDEX` usada por MySQL

```
DROP INDEX <index name>
ON <table name>
```

### CREATE SCHEMA

Un esquema es creado con la sentencia `CREATE SCHEMA`. El esquema es un contenedor de alto nivel para los objetos en una estructura de base de datos SQL-92 [Groff y Weingberg 1999]. Un esquema es una entidad dentro de la base de datos que incluye las definiciones de:

- *Tablas*: Descritas por columnas, claves principales y foráneas, restricciones, etc.
- *Vistas*: Las cuales son tablas virtuales derivadas de las tablas reales en la base de datos.
- *Dominios*: Funcionan como extensiones de tipos de datos para la definición de columnas dentro de las tablas del esquema.
- *Validaciones*: Las cuales son restricciones de integridad de la base de datos que restringen las relaciones de datos entre las tablas del esquema.
- *Privilegios*: Controlan las capacidades que se dan a varios usuarios para acceder y actualizar los datos en la base de datos y modificar la estructura de la base de datos.
- *Conjunto de caracteres*: Son estructuras de la base de datos usados para el soporte de lenguajes internacionales y para la gestión de su representación.





- *Ordenamiento*: Definen la secuencia de ordenamiento para un conjunto de caracteres.
- *Traducciones*: Controlan la conversión de datos tipo texto de un conjunto de caracteres a otro y cómo se realizan las comparaciones entre conjunto de caracteres diferentes.

La sintaxis para esta sentencia se muestra en la tabla 2-26.

Tabla 2-26. Sintaxis de la sentencia *CREATE SCHEMA*.

<code>CREATE SCHEMA &lt;schema name clause&gt;</code>	
<code>[ DEFAULT CHARACTER SET &lt;character set specification&gt; ]</code>	
<code>[ &lt;schema element&gt; ]</code>	
<code>&lt;schema element&gt;</code>	<code>&lt;domain definition&gt;</code>
	<code>&lt;table definition&gt;</code>
	<code>&lt;view definition&gt;</code>
	<code>&lt;grant statement&gt;</code>
	<code>&lt;assertion definition&gt;</code>
	<code>&lt;character set definition&gt;</code>
	<code>&lt;collation definition&gt;</code>
	<code>&lt;translation definition&gt;</code>





# Capítulo 3. Transformación UML-XML-BDOR-UML

En el siguiente capítulo se presenta el problema de la transformación UML-XML-BDOR-UML y se exponen un conjunto de reglas que permiten establecer equivalencias entre los diagramas de clase UML, los documentos esquema XML y los esquemas de bases de bases de datos objeto relacionales, logrando así fundar las bases para realizar intercambio de datos entre documentos XML y bases de datos objeto relacionales (BDOR).

En la primera parte se explica la forma de usar los diagramas de clases para la creación de documentos esquema XML. La estructura de los documentos es modelada mediante un conjunto de recorridos sobre el diagrama de clases UML. Estos recorridos se les denomina vistas XML, las cuales definen la estructura jerárquica de los documentos XML.

En la segunda parte, los documentos esquema XML obtenidos de la primera transformación son convertidos en esquemas BDOR, tomando en cuenta las transformaciones que se deben realizar entre los tipos de datos soportados por la especificación Esquema XML y los tipos de datos soportados por el estándar SQL-92. Estos últimos, varían dependiendo del sistema de gestión de bases de datos objeto relacional que se esté utilizando.

Por último, se presentan un conjunto de reglas para obtener los diagrama de clases UML a partir de los esquemas BDOR obtenidos en la segunda transformación.



## 3.1 Usando diagramas de clases UML para generar esquemas XML

El diseño de una base de datos relacional parte normalmente de un modelo de alto nivel como lo es el modelo Entidad Relación Extendido (ERE) o un diagrama de clases UML [Silberschatz et al. 2002]. Esto quiere decir, que el esquema relacional de la base de datos debe cumplir con las reglas de modelado impuestas en el modelo semántico. De la misma forma el esquema de un documento XML debe ser generado de un modelo de nivel superior, si se quiere llevar a cabo un intercambio bidireccional de datos entre documentos XML y BDOR.

Un diagrama de clase UML está compuesto principalmente por clases y de asociaciones entre clases. La definición de una clase incluye la definición de sus atributos y operaciones. Un atributo puede pertenecer a un tipo de dato simple o a un tipo de dato complejo. Una asociación es una abstracción de una relación entre objetos que se puede clasificar como: asociación/agregación o de generalización/especialización.

La figura 3-1 muestra el diagrama de clases UML que modela las distintas entidades que conforman una empresa y será usado a lo largo de todo el capítulo. A continuación se da una breve explicación acerca del mismo:

- Cada empleado tiene cédula, nombre, fecha de nacimiento, dirección y teléfono de habitación, su descendencia (hijos), si los tiene, cónyuge (si tiene), sus empleados subalternos (si tiene), los proyectos donde participa (si es el caso), el proyecto que coordina (si tiene), el departamento donde trabaja, el departamento que dirige (si es el caso), salario y fecha de ingreso a la compañía.
- Cada departamento tiene nombre, presupuesto anual, dirección de ubicación, personal adscrito, su director y los proyectos que desarrolla.
- Cada proyecto tiene título, presupuesto, fecha de inicio, fecha de finalización, quien lo coordina, sus participantes, los equipos que utiliza y el departamento que lo realiza.
- Cada equipo tiene: código, descripción, cantidad en existencia, precio de adquisición y los equipos donde está conectado.

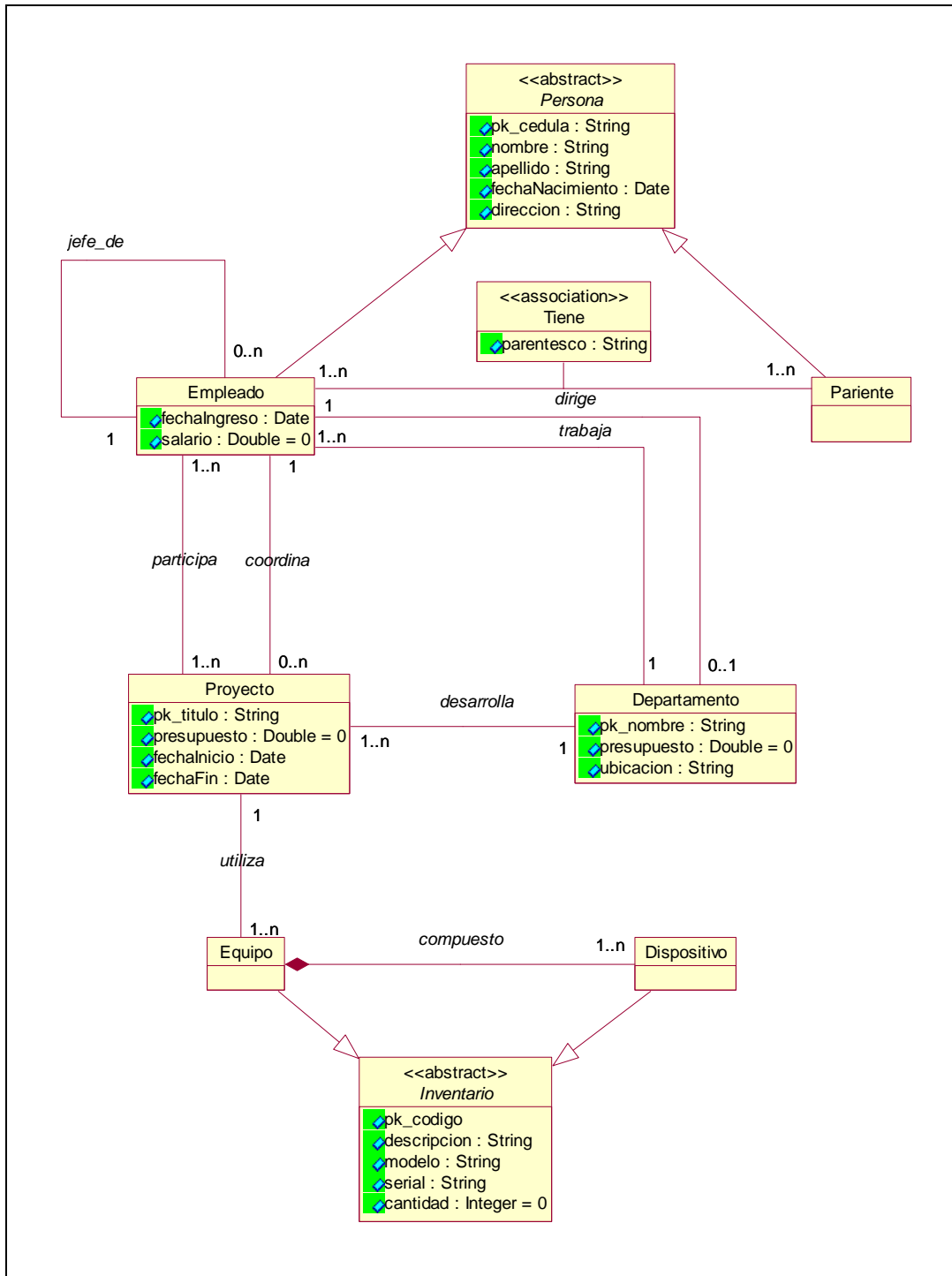


Figura 3-1. Diagrama de Clases



El diagrama de clases de la figura 3-1 es similar a un modelo ERE, donde las clases representan a las entidades y las asociaciones simbolizan las relaciones entre entidades. A continuación se presentan las convenciones usadas para la construcción de este diagrama:

- Las clases abstractas se identifican mediante el estereotipo <<abstract>>.
- Las clases-asociación se identifican mediante el estereotipo <<association>>.
- Las clases que no poseen estereotipo solamente pueden ser especializaciones de las clases abstractas.
- Los atributos que representan claves primarias se identifican mediante el predicado `pk_`.
- Todas las asociaciones entre clases deben tener un nombre y sus respectivas cardinalidades.

En §3.1.2 se mostrará cómo cada clase del diagrama de la figura 3-1 se puede representar en un documento esquema XML como una definición de tipo complejo.

### 3.1.1 Representación de los atributos

Los atributos que pertenecen a una clase UML se definen mediante su nombre, tipo de dato y valor por omisión. Adicionalmente, se puede especificar el nivel de visibilidad del atributo.

La representación de un atributo UML en un documento esquema XML se hace a través del elemento `attribute` cuya sintaxis se especificó en la tabla 2-17. La declaración de este elemento puede incluir los atributos `name`, `type`, y `default` cuyos valores corresponden al nombre del atributo, tipo de dato y al valor por omisión, lo que equivale a la definición de un atributo UML. Adicionalmente, la declaración puede incluir el atributo `use`, el cual puede ser usado para declarar el atributo como requerido.

Por ejemplo, el atributo `pk_cedula` de la clase `Persona` puede representarse en un esquema mediante la siguiente declaración:

```
<xs:attribute name="pk_cedula" type="xs:string" use="required"/>
```



Si al atributo UML se le ha asignado un valor por omisión en su definición, la declaración del elemento `attribute` debe incluir el atributo `default`. Por ejemplo, el atributo `presupuesto` de la clase `Departamento`, puede ser representado en un esquema XML de la siguiente forma:

```
<xs:attribute name="presupuesto" type="xs:double" default="0"/>
```

El uso de UML permite adoptar casi cualquier tipo de dato, pero solamente los tipos de dato: `boolean`, `date`, `dateTime`, `decimal`, `float`, `double`, `integer`, `string` y `time`, son compatibles con los tipos de datos soportados tanto por la especificación Esquema XML como por el estándar SQL-92.

Los tipos de dato multivalorados, como los tipo de dato lista pueden ser representados en un esquema XML a través de las definiciones de tipo simple, usando el elemento `list`, cuya sintaxis se especificó en la tabla 2-10. Por ejemplo, un atributo UML que esté definido como una lista de enteros `List ( integer )`, se representa de la siguiente manera:

```
<xs:simpleType name="listInteger">  
  <xs:list itemType="xs:integer"/>  
</xs:simpleType>
```

### 3.1.2 Representación de las clases

Las clases de un diagrama UML pueden representarse de diferentes formas en un esquema XML. Así como la definición de una clase en el diagrama UML incluye la definición de sus atributos, la declaración de un elemento en un esquema XML también puede incluir declaraciones de atributos. Por ejemplo, la clase `Departamento` puede representarse en un esquema XML de la siguiente manera:

```
<xs:element name="Departamento">  
  <xs:complexType>  
    <xs:attribute name="pk_nombre" type="xs:string" use="required"/>  
    <xs:attribute name="presupuesto" type="xs:double" default="0"/>  
    <xs:attribute name="ubicacion" type="xs:string"/>  
  </xs:complexType>
```



```
</xs:element>
```

Pero no siempre es posible transformar todas las clases en declaraciones de elementos y esto se puede observar en el siguiente esquema XML, donde se intenta representar a la clase abstracta `Persona` y a las subclases `Empleado` y `Pariente`.

```
<xs:complexType name="AbstractPersona">
  <xs:attribute name="pk_cedula" type="xs:string" use="required"/>
  <xs:attribute name="nombre" type="xs:string"/>
  <xs:attribute name="apellido" type="xs:string"/>
  <xs:attribute name="fechaNacimiento" type="xs:date"/>
  <xs:attribute name="direccion" type="xs:string"/>
</xs:complexType>
<xs:element name="empleado">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="AbstractPersona">
        <xs:attribute name="fechaIngreso" type="xs:date"/>
        <xs:attribute name="salario" type="xs:double"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="pariente">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="AbstractPersona"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

En el esquema anterior, la clase abstracta `Persona` tuvo que representarse por medio del tipo complejo `AbstractPersona`, del cual derivan por extensión las declaraciones de los elementos, `empleado` y `pariente`.

Sin embargo, a pesar de que se puede adoptar este tipo de convención, lo que se desea hacer es tratar de representar a todas las clases de una forma similar y esto se logra transformando cada clase del diagrama UML en una definición de tipo complejo dentro del esquema XML.

Siguiendo este enfoque, las definiciones de las clases `Persona`, `Empleado` y `Pariente` quedan de la siguiente manera:





```
<xs:complexType name="AbstractPersona">
  <xs:attribute name="pk_cedula" type="xs:string" use="required"/>
  <xs:attribute name="nombre" type="xs:string"/>
  <xs:attribute name="apellido" type="xs:string"/>
  <xs:attribute name="fechaNacimiento" type="xs:date"/>
  <xs:attribute name="direccion" type="xs:string"/>
</xs:complexType>
<xs:complexType name="Empleado">
  <xs:complexContent>
    <xs:extension base="AbstractPersona">
      <xs:attribute name="fechaIngreso" type="xs:date" use="required"/>
      <xs:attribute name="salario" type="xs:double" default="0"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Pariente">
  <xs:complexContent>
    <xs:extension base="AbstractPersona"/>
  </xs:complexContent>
</xs:complexType>
```

En este último esquema se representaron las clases `Persona`, `Empleado` y `Pariente` como definiciones de tipos complejos. Pero son los elementos y no los tipos complejos los que pueden ser instanciados en un documento XML, por lo tanto es necesario declarar un elemento por cada definición de tipo complejo no abstracto definido en el esquema, tal como se muestra a continuación:

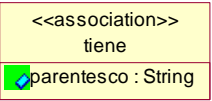
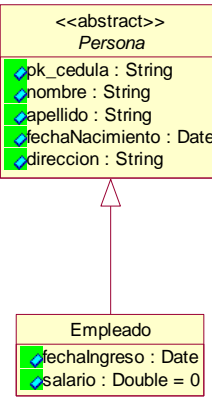
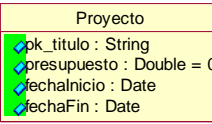
```
<xs:element name="empleado">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="Empleado"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="pariente">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="Pariente"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```



Más adelante se mostrará que estas declaraciones de elementos pueden ser usadas para definir la estructura jerárquica de un documento instancia XML.

La tabla 3-1 muestra los casos más comunes que se presentan cuando se realiza la transformación de clases UML a definiciones de tipos complejos en un esquema XML.

Tabla 3-1. Ejemplo de representación de clases UML en esquemas XML.

Clase UML	Esquema XML
 <pre> classDiagram     class tiene {         parentesco : String     }     </pre>	<pre> &lt;xs:complexType name="AsociationTiene"&gt;   &lt;xs:attribute name="parentesco" type="xs:string"/&gt; &lt;/xs:complexType&gt;     </pre>
 <pre> classDiagram     class Persona {         &lt;&lt;abstract&gt;&gt;         pk_cedula : String         nombre : String         apellido : String         fechaNacimiento : Date         direccion : String     }     class Empleado {         fechaIngreso : Date         salario : Double = 0     }     Persona &lt; -- Empleado     </pre>	<pre> &lt;xs:complexType name="AbstractPersona"&gt;   &lt;xs:attribute name="pk_cedula" type="xs:string" use="required"/&gt;   &lt;xs:attribute name="nombre" type="xs:string"/&gt;   &lt;xs:attribute name="apellido" type="xs:string"/&gt;   &lt;xs:attribute name="fechaNacimiento" type="xs:date"/&gt;   &lt;xs:attribute name="direccion" type="xs:string"/&gt; &lt;/xs:complexType&gt;  &lt;xs:complexType name="Empleado"&gt;   &lt;xs:complexContent&gt;     &lt;xs:extension base="AbstractPersona"&gt;       &lt;xs:attribute name="fechaIngreso" type="xs:date"/&gt;       &lt;xs:attribute name="salario" type="xs:double" default="0"/&gt;     &lt;/xs:extension&gt;   &lt;/xs:complexContent&gt; &lt;/xs:complexType&gt;     </pre>
 <pre> classDiagram     class Proyecto {         pk_titulo : String         presupuesto : Double = 0         fechaInicio : Date         fechaFin : Date     }     </pre>	<pre> &lt;xs:complexType name="Proyecto"&gt;   &lt;xs:attribute name="pk_titulo" type="xs:string" use="required"/&gt;   &lt;xs:attribute name="presupuesto" type="xs:double" default="0"/&gt;   &lt;xs:attribute name="fechaInicio" type="xs:date"/&gt;   &lt;xs:attribute name="fechaFin" type="xs:date"/&gt; &lt;/xs:complexType&gt;     </pre>



### 3.1.3 Representación de las asociaciones

Un diagrama de clases UML no solamente está compuesto por clases, sino de asociaciones entre estas clases. Una asociación se describe en el diagrama UML como una línea que enlaza dos clases y en ella se especifican los roles y las restricciones de cardinalidad existentes entre estas clases. En las siguientes secciones se mostrará cómo se puede hacer uso de la estructura jerárquica del documento XML para representar estas asociaciones.

#### Representación de las asociaciones a través de contenido

Generalmente, los elementos en un documento instancia XML están asociados con otros elementos a través de su contenido. Esto quiere decir, que los hijos de un elemento son los elementos con los que está asociado y esto se cumple para cualquier elemento presente en el documento instancia.

Como ejemplo, supóngase la asociación simple *desarrolla*, presente entre las clases Departamento y Proyecto descrita en la figura 3-3.

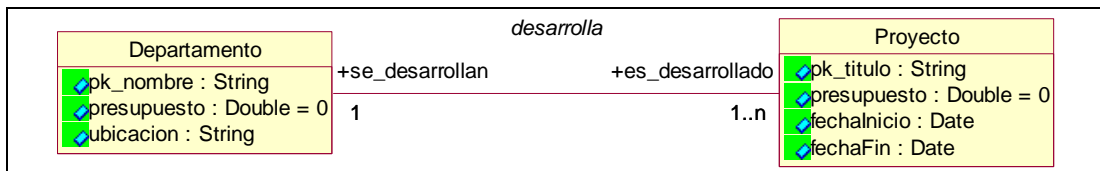


Figura 3-3. Asociación simple entre las clases Departamento y Proyecto.

La asociación mostrada puede leerse de dos formas en lenguaje natural dependiendo del sentido en que sea recorrida la asociación, por ejemplo:

- Departamento → Proyecto:
  - En un Departamento *se\_desarrollan* uno o muchos (1..n) Proyectos.
- Proyecto → Departamento:
  - Un Proyecto *es\_desarrollado* en un único (1) Departamento.



Supóngase que se quiere generar un esquema XML que permita asociar por contenido a cada departamento los proyectos que ha desarrollado. Esto se puede lograr anidando las declaraciones de los elementos de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Departamento">
    <xs:attribute name="pk_nombre" type="xs:string" use="required"/>
    <xs:attribute name="presupuesto" type="xs:double" default="0"/>
    <xs:attribute name="ubicacion" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Proyecto">
    <xs:attribute name="pk_titulo" type="xs:string" use="required"/>
    <xs:attribute name="presupuesto" type="xs:double" default="0"/>
    <xs:attribute name="fechaInicio" type="xs:date"/>
    <xs:attribute name="fechaFin" type="xs:date"/>
  </xs:complexType>
  <xs:element name="proyecto">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Proyecto"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="departamento">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Departamento">
          <xs:sequence>
            <xs:element ref="proyecto" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="empresa">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="departamento" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



En el código anterior se ha añadido dentro de la declaración del elemento `departamento` una secuencia de una o muchas declaraciones de elementos que representan los proyectos que ha desarrollado el departamento. La declaración del elemento `empresa` al final del esquema, corresponde a la declaración del elemento raíz del documento instancia XML, el cual se explicará con más detalle en §3.1.4. Así, un documento XML válido podría contener instancias tales como:

```
<departamento pk_nombre="Departamento A" presupuesto="50000000" ubicacion="Edificio A">
  <proyecto pk_titulo="Proyecto A1" presupuesto="5000000"/>
  <proyecto pk_titulo="Proyecto A2" presupuesto="1000000"/>
  <proyecto pk_titulo="Proyecto A3" presupuesto="8000000"/>
</departamento>
```

La forma de asociar los elementos a través de contenido, aunque es utilizada por la mayoría de los autores, tiene dos grandes desventajas:

- *No puede utilizarse si existen asociaciones múltiples entre clases.* Los elementos asociados por contenido no pueden ser catalogados o diferenciados unos de otros ya que no se toma en cuenta a las asociaciones.
- *El esquema pierde toda la información semántica referente a las asociaciones existentes en el diagrama de clases UML original.* El esquema XML final no incluye declaraciones que contengan información acerca de las asociaciones existentes entre los elementos.

### **Representación de las asociaciones a través de declaraciones de elementos**

Una forma alternativa y mucho más adecuada de representar las asociaciones existentes entre clases es mediante el uso de declaraciones de elementos, las cuales permiten solventar el problema que se presenta cuando existen asociaciones múltiples entre clases.

Un ejemplo de asociaciones múltiples puede observarse en el diagrama de clases UML de la figura 3-4.

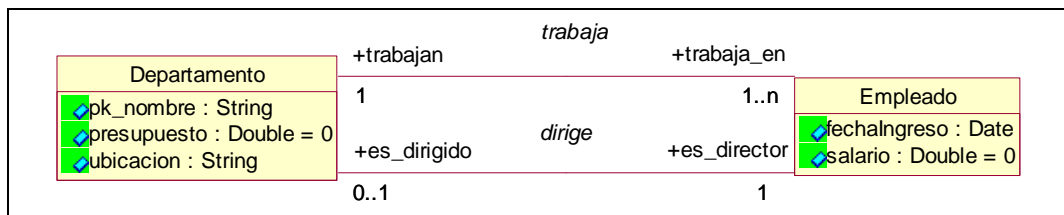


Figura 3-4. Asociaciones múltiples entre las clases Departamento y Empleado.

Las asociaciones presentes entre las clases Departamento y Empleado, mostradas en la figura 3-4 pueden leerse en lenguaje natural como:

- Departamento → Empleado:
  - En un Departamento *trabajan* uno o muchos (1..n) Empleados.
  - Un Departamento *es\_dirigido* por un único (1) Empleado.
- Empleado → Departamento
  - Un Empleado *trabaja* en un único (1) Departamento.
  - Un Empleado *es\_director* de ningún o un único (0..1) Departamento.

Supóngase que en este caso se desea generar un esquema XML que permita validar un documento instancia que contenga información detallada referente a los departamentos y sus empleados. Esto significa que los empleados deben ser clasificados según sus roles ya que existe más de una asociación entre las clases. Sin embargo, los roles tienen un nombre muy específico y es necesario usar a cambio el nombre general de la asociación, para poder incluirlas como declaraciones de elementos dentro del esquema XML.

El siguiente ejemplo muestra la forma de representar en un esquema XML estas asociaciones.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Departamento">
    <xs:attribute name="pk_nombre" type="xs:string" use="required"/>
    <xs:attribute name="presupuesto" type="xs:double" default="0"/>
    <xs:attribute name="ubicacion" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Empleado">
```



```
<xs:attribute name="pk_cedula" type="xs:string" use="required"/>
<xs:attribute name="fechaIngreso" type="xs:date"/>
<xs:attribute name="salario" type="xs:double" default="0"/>
</xs:complexType>
<xs:element name="empleado">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="Empleado"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="departamento">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="Departamento">
        <xs:sequence>
          <xs:element name="trabaja">
            <xs:complexType>
              <xs:sequence>
                <xs:element ref="empleado" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="dirige">
            <xs:complexType>
              <xs:sequence>
                <xs:element ref="empleado"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="empresa">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="departamento" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Usando este enfoque, cada elemento que representa una clase, incluye las declaraciones de los elementos que representan a sus asociaciones y cada elemento que representa una asociación incluye una referencia a la declaración del elemento con el que esta asociado.



Este tipo de esquema tiene grandes ventajas, ya que contiene gran parte de la información semántica descrita en el diagrama de clases, lo que hace posible eventualmente efectuar ingeniería inversa y así obtener nuevamente el diagrama de clases original.

Un documento XML válido para este esquema puede contener instancias tales como:

```
<departamento pk_nombre="Departamento A" presupuesto="20000000" ubicacion="Edificio A">
  <trabaja>
    <empleado pk_cedula="V12000000" fechaIngreso="2000-01-01" salario="1500000"/>
    <empleado pk_cedula="V13000000" fechaIngreso="2001-01-01" salario="1200000"/>
    <empleado pk_cedula="V14000000" fechaIngreso="2002-01-01" salario="900000"/>
  </trabaja>
  <dirige>
    <empleado pk_cedula="V10000000" fechaIngreso="1999-01-01" salario="2000000"/>
  </dirige>
</departamento>
```

### Representación de las clases-asociación

Un elemento que representa una asociación permite agrupar y categorizar los elementos en el documento instancia XML, pero en el caso de una clase-asociación que posea definiciones de atributos, el elemento que la representa solamente puede catalogar a elementos particulares en el documento instancia. Este es el caso de la clase asociación Tiene, presente entre las clases Empleado y Pariente mostrada en la figura 3-5.

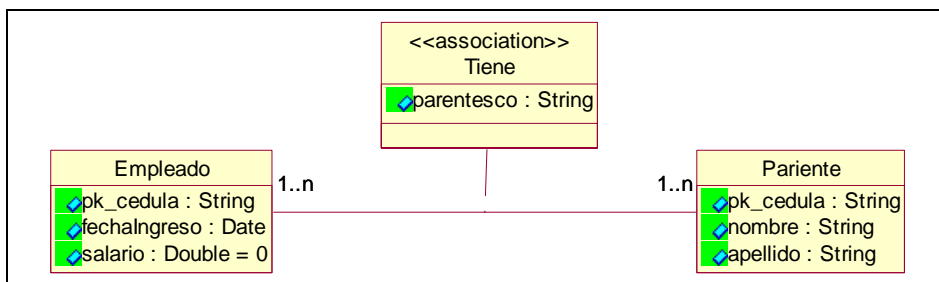


Figura 3-5. Clase-Asociación Tiene.

Supóngase que se desea crear un esquema XML que valide un documento instancia que contenga información referente a los parientes de cada uno de los empleados de una empresa. En este caso, la forma de representar la clase-asociación Tiene presente entre las





clases Empleado y Pariente se hace de forma similar cuando se trabaja con asociaciones, tal como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="AsociacionTiene">
    <xs:attribute name="parentesco" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="Empleado">
    <xs:attribute name="pk_cedula" type="xs:string" use="required"/>
    <xs:attribute name="nombre" type="xs:string" />
    <xs:attribute name="apellido" type="xs:string" />
    <xs:attribute name="salario" type="xs:double" default="0"/>
  </xs:complexType>
  <xs:complexType name="Pariente">
    <xs:attribute name="pk_cedula" type="xs:string" use="required"/>
    <xs:attribute name="nombre" type="xs:string"/>
    <xs:attribute name="apellido" type="xs:string"/>
  </xs:complexType>
  <xs:element name="pariente">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Pariente"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="empleado">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Empleado">
          <xs:sequence>
            <xs:element name="tiene" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="AsociacionTiene">
                    <xs:sequence>
                      <xs:element ref="pariente"/>
                    </xs:sequence>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```
<xs:element name="empresa">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="empleado" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

De esta manera, se declara un elemento que deriva por extensión del tipo complejo que representa a la clase-asociación y en el cual se incluyen las restricciones `minOccurs` y `maxOccurs`, que indican la multiplicidad de la asociación.

Un documento XML válido para este tipo de esquema, podría contener instancias tales como:

```
<empleado pk_cedula="V12352854" nombre="Janmarco" apellido="Rojas" salario="2500000">
  <tiene parentesco="hermana">
    <pariente pk_cedula="V4000000" nombre="Nayrobis" apellido="Rojas"/>
  </tiene>
  <tiene parentesco="hermano">
    <pariente pk_cedula="V5000000" nombre="René" apellido="Rojas"/>
  </tiene>
  <tiene parentesco="hermano">
    <pariente pk_cedula="V6000000" nombre="Delver" apellido="Rojas"/>
  </tiene>
  <tiene parentesco="hermano">
    <pariente pk_cedula="V7000000" nombre="Edgardo" apellido="Rojas"/>
  </tiene>
</empleado>
```

Una de las ventajas de este tipo de representación, es que se pueden realizar consultas usando *XPath* mucho más específicas a través de los elementos que son de tipo clase-asociación. Por ejemplo, la consulta mostrada a continuación devuelve todas las hermanas que tiene el empleado con cedula “V12352854”.

```
//empleado[@pk_cedula="V12352854"]/tiene[@parentesco="hermana"]/pariente
```

Esta consulta devuelve el siguiente elemento:



```
<pariente pk_cedula="V4000000" nombre="Nayrobis" apellido="Rojas"/>
```

### 3.1.4 Declaración del elemento raíz del documento instancia

Todo documento esquema XML incluye una declaración de elemento que define la estructura del elemento raíz del documento instancia XML.

La declaración del elemento raíz estará compuesta por el conjunto de declaraciones de elementos que definen la estructura del documento instancia. Sin embargo, se adoptará por convención que el elemento raíz contendrá únicamente la referencia al elemento que define la vista del documento instancia XML. El nombre de éste elemento es definido por el usuario y debe estar acorde con el contenido de la vista.

Por ejemplo, el siguiente fragmento de código contiene la declaración de un elemento llamado `empresa` que hace referencia al elemento `departamento`, el cual define la estructura o vista del documento instancia XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
  <xs:element name="empresa">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="departamento" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

### 3.1.5 Representación gráfica de la vista del documento instancia

Un documento esquema XML define la estructura de los elementos que contendrá el documento instancia XML. Es decir, que las declaraciones de elementos presentes en un esquema XML deben definir la vista del documento XML.

Una vista establece la forma como serán anidados los elementos en el documento instancia XML y ésta puede definirse a través de las declaraciones de elementos en el esquema XML. Una vista puede describirse gráficamente a través de un conjunto de recorridos sobre las clases y asociaciones de un diagrama UML. Si se quiere que el



documento instancia contenga la mayor parte de la información descrita en el diagrama UML, los recorridos deben incluir la mayor parte de las clases y de las asociaciones sin llegar a formar recorridos cíclicos. La figura 3-6 muestra un ejemplo de esto, donde la clase Departamento es el punto de partida de los recorridos.

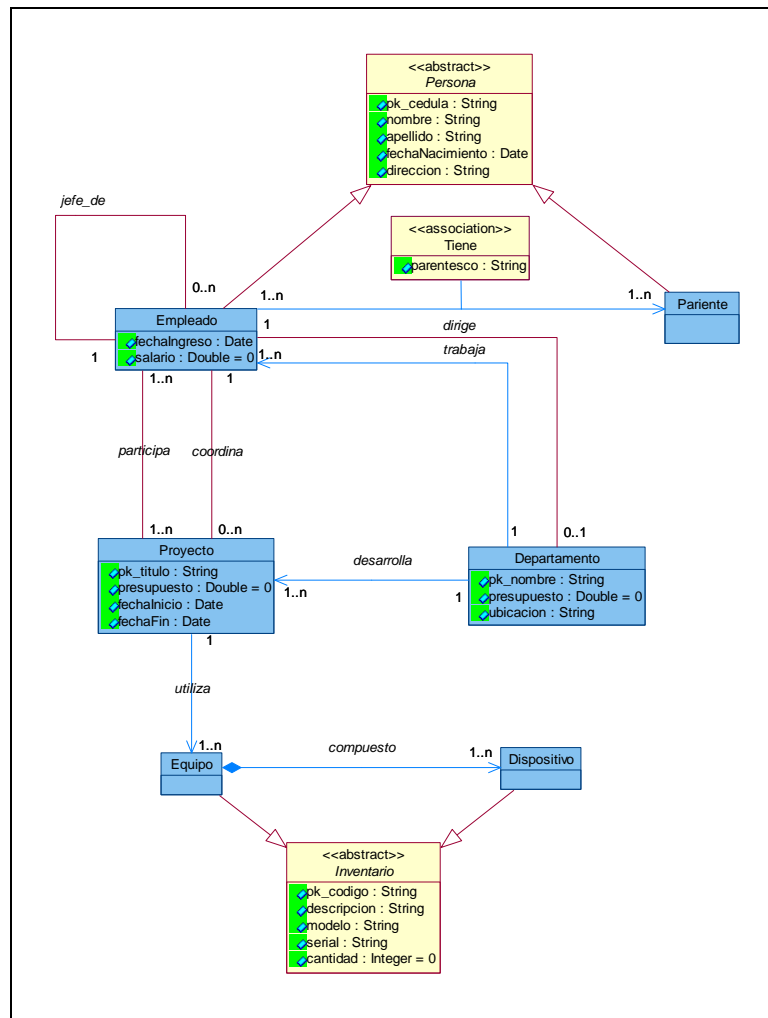


Figura 3-6. Representación gráfica de una vista XML.

El problema de definir una vista que contenga la mayor parte de la información descrita en el diagrama de clases UML se puede solucionar haciendo uso de una estructura de grafo, cuyos vértices representen a las clases principales (clases que ameritan persistencia) y sus aristas a las asociaciones. De esta manera se puede obtener un árbol abarcador sobre dicho



grafo cuya estructura contenga a la mayor parte de los vértices y de las aristas. Dicho árbol abarcador no es más que un subgrafo cuya estructura puede ajustarse fácilmente a la estructura jerárquica de un documento XML.

### 3.1.6 Reglas para construir un documento esquema XML a partir de un diagrama de clases UML

Las siguientes reglas permiten obtener un esquema XML que puede ser utilizado para generar posteriormente un esquema BDOR equivalente.

<b>Regla 1. Definir la vista que tendrá el documento instancia XML.</b>
---

La estructura del documento instancia puede ser definida gráficamente mediante un conjunto de recorridos (no cíclicos) sobre el diagrama de clases UML. Los recorridos pueden trazarse siguiendo los siguientes pasos:
--

- |   |
|---|
| <ul style="list-style-type: none"><li>▪ Hacer una analogía entre el diagrama de clases UML y una estructura de grafo, donde las clases de negocio representan a los vértices y las asociaciones representan a las aristas.</li><li>▪ Seleccionar la clase inicial que será el punto de partida de todos los recorridos.</li><li>▪ Obtener un árbol abarcador sobre el grafo formado por el conjunto de clases y asociaciones seleccionado anteriormente. El árbol abarcador obtenido define la vista o estructura jerárquica que tendrá el documento instancia XML.</li></ul> |
|---|

Lo primero que se debe hacer antes de comenzar a crear la estructura de un esquema XML es definir la vista de los datos que estarán contenidos en el documento instancia XML. Como ejemplo, se tomará la vista descrita en la figura 3-6 presentada en §3.1.5.

<b>Regla 2. Seleccionar las clases y asociaciones a representar en el esquema XML.</b>
--

Inicialmente se seleccionan las clases que poseen estereotipo <code>&lt;&lt;abstract&gt;&gt;</code> , esto es, las clases del diagrama UML que definen los tipos abstractos. Luego se seleccionan todas las clases que definen la vista del documento XML.
--

Finalmente, se selecciona cada clase-asociación (clases con estereotipo <code>&lt;&lt;association&gt;&gt;</code> ) que forma parte del conjunto de recorridos descrito por la vista.
--



Cuando se define una vista, las clases que definen los tipos abstractos no son tomadas en cuenta ya que no forman parte del conjunto de clases de negocio. Sin embargo, todas las clases de negocio se definen en base a éstas y es por ello que deben ser seleccionadas antes que todas las demás clases.

### Regla 3. Declarar el elemento `schema`

El elemento `schema` es donde van incluidas todas las definiciones y declaraciones que conforman el documento esquema.

Todo documento esquema XML debe comenzar con el prólogo XML seguido de la declaración del elemento `schema`, dentro de la cual van incluidas todas las demás definiciones.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

### Regla 4. Representar las clases que hayan sido seleccionadas como definiciones de tipos complejos dentro del esquema XML.

Las clases que poseen estereotipo se representan en el esquema de la siguiente forma:

- Cada clase con estereotipo `<<abstract>>` se representa como un tipo complejo abstracto cuyo nombre estará formado por el predicado `Abstract` seguido del nombre de la clase.
- Cada clase con estereotipo `<<association>>` se representa como un tipo complejo cuyo nombre estará formado por el predicado `Association` seguido del nombre de la clase.

Finalmente, las clases que definen la vista del documento XML (clases sin estereotipo) se representan de la siguiente manera:

- Cada clase se representa como una definición de tipo complejo que posee el mismo nombre de la clase, el cual debe iniciar siempre con letra mayúscula.
- Si la clase hereda de una superclase, la definición debe derivar por extensión del tipo complejo abstracto que representa a la superclase.



Se recomienda añadir inicialmente las definiciones de tipos complejos que representan las clases abstractas y las clases-asociaciones y finalmente todas las demás definiciones. Esto con el objetivo de mejorar la legibilidad del documento.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="AbstractPersona"/>
  <xs:complexType name="AbstractInventario"/>
  <xs:complexType name="AssociationTiene"/>
  <xs:complexType name="Pariente">
    <xs:complexContent>
      <xs:extension base="AbstractPersona"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Empleado">
    <xs:complexContent>
      <xs:extension base="AbstractPersona"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Dispositivo">
    <xs:complexContent>
      <xs:extension base="AbstractInventario"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Equipo">
    <xs:complexContent>
      <xs:extension base="AbstractInventario"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Proyecto"/>
  <xs:complexType name="Departamento"/>
</xs:schema>
```

**Regla 5. Representar cada atributo definido en UML como una declaración de atributo dentro del esquema XML.**

Cada atributo definido dentro de una clase UML debe ser representado como una declaración de atributo local en la definición de un tipo complejo.



- Si el atributo representa una clave primaria, esto es, si su nombre comienza con el predicado `pk_`, se debe declarar como un atributo requerido en el esquema (`use="required"`).
- Si el atributo UML es multivaluado, definido como `List ( type )`, donde `type` es el tipo de dato de la lista, se debe añadir un tipo simple (`simpleType`) al esquema XML, el cual debe definir un elemento `list` cuyo atributo `itemType` toma como valor un tipo de dato equivalente al tipo de dato del atributo.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:complexType name="AbstractPersona">
    <xs:attribute name="pk_cedula" type="xs:string" use="required"/>
    <xs:attribute name="nombre" type="xs:string"/>
    <xs:attribute name="apellido" type="xs:string"/>
    <xs:attribute name="fechaNacimiento" type="xs:date"/>
    <xs:attribute name="direccion" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="AbstractInventario">
    <xs:attribute name="pk_codigo" type="xs:string" use="required"/>
    <xs:attribute name="descripcion" type="xs:string"/>
    <xs:attribute name="modelo" type="xs:string"/>
    <xs:attribute name="serial" type="xs:string"/>
    <xs:attribute name="cantidad" type="xs:integer" default="0"/>
  </xs:complexType>
  <xs:complexType name="AssociationTiene">
    <xs:attribute name="parentesco" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Pariente">
    <xs:complexContent>
      <xs:extension base="AbstractPersona"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Empleado">
    <xs:complexContent>
      <xs:extension base="AbstractPersona">
        <xs:attribute name="fechaIngreso" type="xs:date" use="required"/>
        <xs:attribute name="salario" type="xs:double" default="0"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Dispositivo">
    <xs:complexContent>
      <xs:extension base="AbstractInventario"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Equipo">
    <xs:complexContent>
```





```
<xs:extension base="AbstractInventario"/>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="Proyecto">
  <xs:attribute name="pk_titulo" type="xs:string" use="required"/>
  <xs:attribute name="presupuesto" type="xs:double" default="0"/>
  <xs:attribute name="fechaInicio" type="xs:date"/>
  <xs:attribute name="fechaFin" type="xs:date"/>
</xs:complexType>
<xs:complexType name="Departamento">
  <xs:attribute name="pk_nombre" type="xs:string" use="required"/>
  <xs:attribute name="presupuesto" type="xs:double" default="0"/>
  <xs:attribute name="ubicacion" type="xs:string"/>
</xs:complexType>
</xs:schema>
```

Las reglas 6 y 7 presentadas a continuación, son las que definen la estructura del documento instancia dependiendo de la vista que haya sido elegida.

**Regla 6. Declarar los elementos que definen la vista del documento XML.**

Se realiza un recorrido por el conjunto de clases y asociaciones pertenecientes a la vista del documento y por cada clase visitada se debe declarar un elemento que tenga el mismo nombre de la clase, escrito en minúsculas y que derive por extensión del tipo complejo que representa a la clase. Cada declaración de elemento que representa una clase, debe incluir las declaraciones de los elementos que representan a sus asociaciones y cada elemento que representa una asociación debe incluir una referencia al elemento asociado. La multiplicidad de la asociación se representa añadiendo los atributos `minOccurs` y `maxOccurs` a la declaración del elemento referencia.

Cada elemento que representa una clase-asociación debe derivar por extensión del tipo complejo que la representa. En este caso, los atributos `minOccurs` y `maxOccurs` deben ser añadidos a esta declaración y no a la declaración de elemento al cual se hace referencia.

**Regla 7. Declarar el elemento raíz del documento instancia**

Esta declaración debe incluir la referencia al elemento raíz que define la vista, con multiplicidad `maxOccurs="unbounded"`.

Por convención, la declaración del elemento raíz debe aparecer al final de todas las definiciones y declaraciones del documento esquema XML. En otras palabras, es el último



hijo del elemento `schema`. El siguiente esquema simplificado muestra las declaraciones de elementos que definen la vista del documento instancia XML.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- Definiciones de tipos complejos -- >
  <xs:element name="pariente">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Pariente"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="empleado">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Empleado">
          <xs:sequence>
            <xs:element name="tiene" maxOccurs="unbounded">
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="AssociationTiene">
                    <xs:sequence>
                      <xs:element ref="pariente"/>
                    </xs:sequence>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="dispositivo">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Dispositivo"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="equipo">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="Equipo">
          <xs:sequence>
            <xs:element name="compuesto">
              <xs:complexType>
```



```
        <xs:sequence>
            <xs:element ref="dispositivo" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>
<xs:element name="proyecto">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="Proyecto">
                <xs:sequence>
                    <xs:element name="utiliza">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="equipo" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<xs:element name="departamento">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="Departamento">
                <xs:sequence>
                    <xs:element name="trabaja">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="empleado" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="desarrolla">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="proyecto" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
```



```
</xs:complexType>
</xs:element>
<xs:element name="empresa">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="departamento" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

### 3.1.7 Resultados de la transformación UML → XML

El esquema XML resultante de la transformación tiene ciertas características que lo distinguen de otro tipo de esquema, las cuales se listan a continuación:

- Las definiciones de tipos complejos representan a las clases de un diagrama UML. De igual forma una definición de tipo complejo puede ser vista como una entidad en un modelo ERE.
- Las declaraciones de atributos representan definiciones de atributos UML. Igualmente pueden representar los atributos de las entidades en un modelo ERE.
- Las declaraciones de atributos que llevan en su nombre el predicado `pk_` representan a identificadores o atributos clave. Una declaración de éstas también pueden ser vista como la clave primaria de una entidad en un modelo ERE.
- Las definiciones de tipo complejo que llevan el predicado `Abstract` en su nombre, representan a las clases abstractas de un diagrama de clases. Estas definiciones también representan a entidades en un modelo ERE.
- Las definiciones de tipo complejo que llevan el predicado `Association` en su nombre, representan las clases-asociación de un diagrama UML. Una definición de éstas representa una relación (muchos a muchos) entre entidades en un modelo ERE.
- Cada definición de tipo complejo no abstracto, exceptuando los tipos que representan clases-asociación, tienen asociada una declaración de elemento definida bajo la raíz del documento esquema (`schema`). Cada una de estas declaraciones puede tener un



conjunto de declaraciones de elementos hijos, la cuales representan asociaciones entre clases en un diagrama UML (o relaciones entre entidades en un modelo ERE).

- Cada declaración de elemento que representa una asociación o relación entre elementos, contiene una declaración de elemento hija que hace referencia a una declaración de elemento definido bajo la raíz del documento esquema.
- El esquema posee una declaración de elemento que define el elemento raíz del documento instancia XML. Esta declaración no deriva por extensión de ningún tipo complejo y además contiene una referencia al elemento que define la estructura del documento instancia XML.

## 3.2 Usando esquemas XML para construir esquemas BDOR

El esquema XML obtenido en §3.1.6, generado a partir del diagrama de clases UML y mostrado en la figura 3.6, puede ser usado para crear un esquema BDOR usando el lenguaje de definición de datos (LDD) de SQL-92. En las siguientes secciones se explicará con detalle cómo son transformados los componentes de un esquema XML en un conjunto de sentencias SQL que conforman un esquema BDOR.

### 3.2.1 Representación de las declaraciones de atributos

Las declaraciones de atributos de un esquema XML pueden ser transformadas en cláusulas de definiciones de columnas embebidas en una sentencia SQL de creación de tabla en un esquema BDOR. La transformación de atributos implica llevar a cabo la conversión de los tipos de datos, ya que los SGBDOR no soportan todos los tipos de datos definidos por la especificación Esquema XML. La tabla 3-2 muestra los diferentes tipos de conversión a seguir según el tipo de SGBDOR que se esté usando.



Tabla 3-2. Tipos de datos XML que son compatibles con los tipos de datos de los SGBDOR.

Esquema XML	SQL Server	MySQL	PostgreSQL	Oracle
boolean	bit	bit	bit	-
date	-	date	date	date
datetime	datetime	datetime	timestamp	-
decimal	decimal	decimal	decimal	number
double	-	double	float8	-
float	float	float	float4	float
integer	int	int	integer	number
string	varchar	varchar	varchar	varchar2
time	-	time	time	-

La tabla 3-3, muestra algunas transformaciones de declaraciones de atributos XML en cláusulas SQL de definición de columnas usando el LDD de MySQL.

Tabla 3-3. Ejemplos de transformación de declaraciones de atributos XML en cláusulas SQL para MySQL.

Declaración de atributo XML	Cláusula SQL
<xs:attribute name="pk_cedula" type="xs:string" use="required"/>	cedula <b>varchar(50) not null, primary key</b> (cedula)
<xs:attribute name="pk_nombre" type="xs:string"/>	nombre <b>varchar(50) not null, primary key</b> (nombre)
<xs:attribute name="presupuesto" type="xs:double" default="0"/>	presupuesto <b>double default 0</b>
<xs:attribute name="cantidad" type="xs:integer" default="0"/>	cantidad <b>integer default 0</b>
<xs:attribute name="fechaIngreso" type="xs:date"/>	fechaIngreso <b>date</b>

Nótese que cada atributo cuyo nombre tiene predicado `pk_` se transforma en una cláusula de definición de columna primaria. El uso del predicado `pk_` es indispensable si se desea identificar cuáles de las declaraciones de atributos XML serán transformadas en cláusulas de definición de columnas primarias dentro de un esquema BDOR.



### 3.2.2 Representación de las definiciones de tipos complejos

Las definiciones de tipos complejos no abstractos que forman parte de un documento esquema XML pueden ser transformadas en sentencias SQL de creación de tablas dentro de un esquema BDOR. Se ha adoptado por convención en §3.1.7 que los tipos complejos no abstractos son aquellos cuyo nombre no lleva el predicado `Abstract` en su definición.

A continuación se muestra un ejemplo de cómo se realiza este tipo de transformación:

```
<xs:complexType name="Departamento">
  <xs:attribute name="pk_nombre" type="xs:string" use="required"/>
  <xs:attribute name="presupuesto" type="xs:double" default="0"/>
  <xs:attribute name="ubicacion" type="xs:string"/>
</xs:complexType>
```

La definición del tipo complejo `Departamento` puede ser transformada en la siguiente sentencia de creación de tabla, usando el LDD de MySQL.

```
CREATE TABLE Departamento(
  nombre VARCHAR(50) NOT NULL,
  PRIMARY KEY (nombre),
  presupuesto DOUBLE DEFAULT 0,
  ubicacion VARCHAR(50)
) TYPE=InnoDB;
```

Si la definición del tipo complejo no abstracto deriva por extensión de un tipo complejo abstracto, entonces se asume que el subtipo hereda todas las declaraciones de atributos del supertipo. Esto se hace ya que la norma SQL-92 no soporta la definición de herencia. A continuación se muestra un ejemplo de cómo resultan estas transformaciones.

```
<xs:complexType name="AbstractPersona">
  <xs:attribute name="pk_cedula" type="xs:string" use="required"/>
  <xs:attribute name="nombre" type="xs:string"/>
  <xs:attribute name="apellido" type="xs:string"/>
  <xs:attribute name="fechaNacimiento" type="xs:date"/>
  <xs:attribute name="direccion" type="xs:string"/>
</xs:complexType>
<xs:complexType name="Empleado">
  <xs:complexContent>
    <xs:extension base="AbstractPersona">
```



```
<xs:attribute name="fechaIngreso" type="xs:date" use="required"/>
<xs:attribute name="salario" type="xs:double" default="0"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

En este caso el tipo complejo `Empleado` deriva por extensión del tipo complejo abstracto `AbstractPersona`, heredando así, las declaraciones de atributos: `fechaIngreso` y `salario`, las cuales también deben ser transformadas en una lista de cláusulas SQL de definición de columnas. Por consiguiente, la transformación resultante queda de la siguiente forma:

```
CREATE TABLE Empleado(
  cedula VARCHAR(50) NOT NULL,
  PRIMARY KEY (cedula),
  nombre VARCHAR(50),
  apellido VARCHAR(50),
  fechaNacimiento DATE,
  direccion VARCHAR(50),
  fechaIngreso DATE,
  salario DOUBLE DEFAULT 0
) TYPE=InnoDB;
```

### 3.2.3 Representación de las declaraciones de elementos

Un esquema XML define la estructura que tendrá el documento instancia XML a través de las declaraciones de elementos, las cuales definen el modelo de contenido que tendrá cada elemento en el documento instancia. En otras palabras, las declaraciones de elementos definen la estructura jerárquica de cada uno de los elementos del documento instancia.

En §3.1.3 se mostró la forma de definir relaciones entre elementos usando elementos que representan relaciones. El siguiente código el elemento compuesto establece una relación entre los elementos `equipo` y `dispositivo`.

```
<xs:complexType name="AbstractInventario">
  <xs:attribute name="pk_codigo" type="xs:string" use="required"/>
  <xs:attribute name="descripcion" type="xs:string"/>
  <xs:attribute name="modelo" type="xs:string"/>
  <xs:attribute name="serial" type="xs:string"/>
```





```
<xs:attribute name="cantidad" type="xs:integer" default="0"/>
</xs:complexType>
<xs:complexType name="Equipo">
  <xs:complexContent>
    <xs:extension base="AbstractInventario"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Dispositivo">
  <xs:complexContent>
    <xs:extension base="AbstractInventario"/>
  </xs:complexContent>
</xs:complexType>
<xs:element name="dispositivo">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="Dispositivo"/>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="equipo">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="Equipo">
        <xs:sequence>
          <xs:element name="compuesto">
            <xs:complexType>
              <xs:sequence>
                <xs:element ref="dispositivo" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

En primer lugar, las declaraciones de elementos equipo y dispositivo que derivan por extensión, respectivamente, de los tipos complejos no abstractos Equipo y Dispositivo, son transforman en tablas que llevan sus nombres dentro del esquema BDOR. Por consiguiente, el elemento compuesto que representa una relación entre estos dos elementos se debe transformar en una tabla que relaciona a las tablas equipo y dispositivo a través de sus claves foráneas. A continuación se muestra el esquema BDOR resultante de la transformación.



```
CREATE TABLE Equipo(  
  codigo VARCHAR(50) NOT NULL,  
  PRIMARY KEY (codigo),  
  descripcion VARCHAR(50),  
  modelo VARCHAR(50),  
  serial VARCHAR(50),  
  cantidad INTEGER DEFAULT 0  
) TYPE=InnoDB;  
  
CREATE TABLE Dispositivo(  
  codigo VARCHAR(50) NOT NULL,  
  PRIMARY KEY (codigo),  
  descripcion VARCHAR(50),  
  modelo VARCHAR(50),  
  serial VARCHAR(50),  
  cantidad INTEGER DEFAULT 0  
) TYPE=InnoDB;  
  
CREATE TABLE compuesto(  
  codigoEquipo VARCHAR(50),  
  FOREIGN KEY (codigoEquipo) REFERENCES Equipo(codigo),  
  codigoDispositivo VARCHAR(50),  
  FOREIGN KEY (codigoDispositivo) REFERENCES Dispositivo(codigo)  
) TYPE=InnoDB;
```

### 3.2.4 Reglas para construir un esquema BDOR a partir de un documento esquema XML

Las siguientes reglas permiten obtener un esquema BDOR a partir de un esquema XML, el cual debe poseer las características expuestas en §3.1.8.

<b>Regla 1. Convertir las definiciones de tipos complejos en tablas que poseen claves primarias.</b>
--

Para cada definición de tipo complejo que representa una entidad de negocio, es decir, aquellas cuyo nombre no comience con alguno de los predicados <i>Abstract</i> o <i>Association</i> , se crea una sentencia <code>CREATE TABLE &lt;nombre_tabla&gt;</code> dentro del esquema BDOR, donde <code>&lt;nombre_tabla&gt;</code> corresponde al nombre del tipo complejo. Luego, las declaraciones de atributos pertenecientes a cada definición de tipo complejo, incluyendo las que han sido derivadas por extensión, se transforman en una lista de cláusulas de definición de columnas separadas por comas, embebidas en la sentencia de creación de tabla anterior. Para ello se siguen los siguientes pasos:
---



- Cada declaración de atributo se transforma en una cláusula `<nombre_columna> TIPO_SQL`, donde `<nombre_columna>` tomará como valor el nombre del atributo y `TIPO_SQL` es un tipo de dato SQL equivalente al tipo de dato XML con el que se ha declarado el atributo. Si el atributo XML ha sido declarado con un valor por omisión, se debe extender la cláusula de definición de columna a `<nombre_columna> TIPO_SQL DEFAULT <valor_inicial>`, donde `<valor_inicial>` corresponderá a dicho valor por omisión.
- Cada declaración de atributo cuyo nombre comience con el predicado `pk_` se transforma en una cláusula `PRIMARY KEY <nombre_columna>`, donde `<nombre_columna>` toma como valor el nombre del atributo restándole el predicado `pk_`.

La regla 1, aplicada al documento esquema obtenido en §3.1.6, genera el siguiente conjunto de sentencias SQL, utilizando el LDD de MySQL.

```
CREATE TABLE Empleado(  
  cedula VARCHAR(50) NOT NULL,  
  PRIMARY KEY (cedula),  
  nombre VARCHAR(50),  
  apellido VARCHAR(50),  
  fechaNacimiento DATE,  
  direccion VARCHAR(50),  
  fechaIngreso DATE,  
  salario DOUBLE DEFAULT 0  
) TYPE=InnoDB;  
  
CREATE TABLE Pariente(  
  cedula VARCHAR(50) NOT NULL,  
  PRIMARY KEY (cedula),  
  nombre VARCHAR(50),  
  apellido VARCHAR(50),  
  fechaNacimiento DATE,  
  direccion VARCHAR(50)  
) TYPE=InnoDB;  
  
CREATE TABLE Proyecto(  
  titulo VARCHAR(50) NOT NULL,  
  PRIMARY KEY (titulo),  
  presupuesto DOUBLE DEFAULT 0,  
  fechaInicio DATE,  
  fechaFin DATE  
) TYPE=InnoDB;
```



```
CREATE TABLE Departamento(  
    nombre VARCHAR(50) NOT NULL,  
    PRIMARY KEY (nombre),  
    presupuesto DOUBLE DEFAULT 0,  
    ubicacion VARCHAR(50)  
) TYPE=InnoDB;  
  
CREATE TABLE Equipo(  
    codigo VARCHAR(50) NOT NULL,  
    PRIMARY KEY (codigo),  
    descripcion VARCHAR(50),  
    modelo VARCHAR(50),  
    serial VARCHAR(50),  
    cantidad INTEGER DEFAULT 0  
) TYPE=InnoDB;  
  
CREATE TABLE Dispositivo(  
    codigo VARCHAR(50) NOT NULL,  
    PRIMARY KEY (codigo),  
    descripcion VARCHAR(50),  
    modelo VARCHAR(50),  
    serial VARCHAR(50),  
    cantidad INTEGER DEFAULT 0  
) TYPE=InnoDB;
```

En el conjunto de sentencias SQL anterior se puede observar que la longitud del tipo de dato VARCHAR se ha establecido en cincuenta (50) caracteres, para todas las definiciones de columnas que son de este tipo, pero bien pudo haberse tomado cualquier otro valor entre cero (0) y doscientos cincuenta y cinco (255) caracteres.

Otro punto importante a destacar, es que el conjunto de tablas obtenido inicialmente no está relacionado entre sí. La regla que sigue a continuación permite establecer relaciones entre este conjunto de tablas.

**Regla 2. Convertir las declaraciones de elementos que expresan relaciones en tablas que poseen claves foráneas.**

Los elementos declarados bajo la raíz del esquema XML que derivan por extensión de un tipo complejo que representa una entidad de negocio, pueden contener una secuencia de declaraciones de elementos locales que expresan relaciones entre elementos. Cada elemento perteneciente a esta secuencia establece una relación entre su elemento padre y su elemento hijo con ciertas restricciones de multiplicidad. En consecuencia, las relaciones entre las tablas de la base de datos se deben crear siguiendo los siguientes



pasos:

- Crear una sentencia `CREATE TABLE <nombre_tabla>`, donde `<nombre_tabla>` corresponde al nombre del elemento que asocia al elemento padre con el elemento hijo.
- Si el elemento deriva por extensión de un tipo complejo (con predicado `Association`), se añade a la sentencia anterior, una lista de cláusulas de definición de columnas separadas por comas, creada a partir de las declaraciones de atributos que han sido heredadas de este tipo.
- Añadir una cláusula de definición de columna `<nombre_columna> TIPO_SQL`, creada a partir del atributo principal del elemento padre. Añadir seguidamente una cláusula de definición de clave foránea `FOREIGN KEY <nombre_columna> REFERENCES <nombre_tabla> (<nombre_clave>)`, en donde `<nombre_columna>` tendrá como valor el nombre del atributo principal concatenado al nombre del tipo complejo del cual extiende el elemento padre; `<nombre_tabla>` es el nombre del tipo complejo del cual extiende el elemento padre y `<nombre_columna>` es el nombre del atributo principal del elemento padre.
- Añadir una cláusula de definición de columna `<nombre_columna> TIPO_SQL`, creada a partir del atributo principal del elemento hijo. Añadir seguidamente una cláusula de definición de clave foránea `FOREIGN KEY <nombre_columna> REFERENCES <nombre_tabla> (<nombre_clave>)`, en donde `<nombre_columna>` tendrá como valor el nombre del atributo principal concatenado al nombre del tipo complejo del cual extiende el elemento hijo; `<nombre_tabla>` es el nombre del tipo complejo del cual extiende el elemento hijo y `<nombre_columna>` es el nombre del atributo principal del elemento hijo.

El conjunto de tablas resultante de esta transformación se encargan de establecer las relaciones entre las tablas de negocio (que poseen claves primarias), y además se identifican por poseer claves foráneas.

La regla 2, aplicada al documento esquema obtenido en §3.1.6, genera el siguiente conjunto de sentencias SQL que completan el esquema BDOR.

```
CREATE TABLE tiene(  
  parentesco VARCHAR(50),  
  cedulaEmpleado VARCHAR(50),  
  FOREIGN KEY (cedulaEmpleado) REFERENCES Empleado(cedula),  
  cedulaPariente VARCHAR(50),  
  FOREIGN KEY (cedulaPariente) REFERENCES Pariente(cedula)  
) TYPE=InnoDB;
```

```
CREATE TABLE compuesto(  
  codigoEquipo VARCHAR(50),  
  FOREIGN KEY (codigoEquipo) REFERENCES Equipo(codigo),
```



```
codigoDispositivo VARCHAR(50),
FOREIGN KEY (codigoDispositivo) REFERENCES Dispositivo(codigo)
) TYPE=InnoDB;

CREATE TABLE utiliza(
    tituloProyecto VARCHAR(50),
    FOREIGN KEY (tituloProyecto) REFERENCES Proyecto(titulo),
    codigoEquipo VARCHAR(50),
    FOREIGN KEY (codigoEquipo) REFERENCES Equipo(codigo)
) TYPE=InnoDB;

CREATE TABLE trabaja(
    nombreDepartamento VARCHAR(50),
    FOREIGN KEY (nombreDepartamento) REFERENCES Departamento(nombre),
    cedulaEmpleado VARCHAR(50),
    FOREIGN KEY (cedulaEmpleado) REFERENCES Empleado(cedula)
) TYPE=InnoDB;

CREATE TABLE desarrolla(
    nombreDepartamento VARCHAR(50),
    FOREIGN KEY (nombreDepartamento) REFERENCES Departamento(nombre),
    tituloProyecto VARCHAR(50),
    FOREIGN KEY (tituloProyecto) REFERENCES Proyecto(titulo)
) TYPE=InnoDB;
```

### 3.2.5 Resultados de la transformación XML→BDOR

El esquema BDOR mostrado en §3.2.4 generado a partir de un esquema XML, tiene ciertas características que lo distinguen de otros esquemas BDOR:

- El esquema posee un conjunto de tablas principales, donde cada una de éstas posee una clave primaria y una serie de columnas que describen las propiedades de las entidades de negocio.
- El esquema posee un conjunto de tablas que tiene como función establecer las relaciones entre las tablas principales. Cada una de estas tablas contiene las claves foráneas de las tablas que relaciona. A este conjunto de tablas se les denominará tablas de relación, o bien, tablas asociación.



## 3.3 Usando esquemas BDOR para construir diagramas de clases UML

Un esquema BDOR que posea las características presentadas en §3.2.5, puede ser descrito mediante un diagrama de clases UML, haciendo que las tablas del esquema BDOR se representen como un conjunto de clases y asociaciones. En las siguientes secciones se mostrará qué parte del conjunto de tablas representan las clases y cuáles representan asociaciones.

### 3.3.1 Representación de las tablas que poseen claves primarias

El conjunto de tablas que poseen definiciones de claves primarias, se representa en un diagrama UML como un conjunto de clases. Por ejemplo, cada una de las tablas Proyecto y Departamento definidas a continuación, poseen una definición de columna primaria.

```
CREATE TABLE Proyecto(  
    titulo VARCHAR(50) NOT NULL,  
    PRIMARY KEY (titulo),  
    presupuesto DOUBLE DEFAULT 0  
) TYPE=Innodb;  
  
CREATE TABLE Departamento(  
    nombre VARCHAR(50) NOT NULL,  
    PRIMARY KEY (nombre),  
    presupuesto DOUBLE DEFAULT 0,  
    ubicacion VARCHAR(50)  
) TYPE=Innodb;
```

Estas tablas pueden ser representadas como clases, tal cual como se muestra en la figura 3-7.



Figura 3-7. Clases generadas a partir de las tablas Departamento y Proyecto del esquema BDOR.

La transformación de las definiciones de columnas de las tablas en definiciones de atributos, implica llevar a cabo una conversión de tipo de dato SQL a un tipo de dato equivalente UML. Por ejemplo, la columna ubicación de la tabla departamento definida como tipo VARCHAR, se transforma como un atributo de tipo de dato String en el diagrama UML.

### 3.3.2 Representación de las tablas que poseen claves foráneas.

El conjunto de tablas que poseen definiciones de claves foráneas se representa en un diagrama UML como un conjunto de asociaciones. Por convención, una tabla que representa una relación entre dos tablas, debe contener dos claves foráneas definidas a partir de las claves principales de las tablas que relaciona. Por ejemplo, la tabla desarrolla posee las claves foráneas nombreDepartamento y tituloProyecto, que relacionan a las clases Departamento y Proyecto.

```
CREATE TABLE desarrolla(
  nombreDepartamento VARCHAR(50),
  FOREIGN KEY (nombreDepartamento) REFERENCES Departamento(nombre),
  tituloProyecto VARCHAR(50),
  FOREIGN KEY (tituloProyecto) REFERENCES Proyecto(titulo)
) TYPE=Innodb;
```

Esta tabla se representa entonces, una asociación entre las clases Departamento y Proyecto en el diagrama de clases UML (ver figura 3-8).



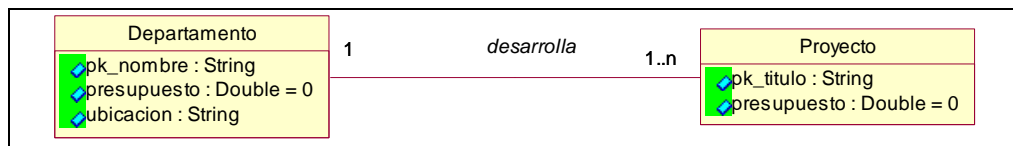


Figura 3-8. Asociación generada a partir de la tabla desarrolla del esquema BDOR.

### 3.3.3 Reglas para construir un diagrama UML a partir de un esquema BDOR

Las reglas mostradas a continuación, permiten obtener un diagrama UML a partir de un esquema BDOR, el cual debe poseer las características expresadas en §3.2.5.

**Regla 1. Representar las tablas que poseen definiciones de claves primarias como clases.**

Por cada clase del esquema BDOR que posea una definición de columna primaria en su definición se describe en el diagrama de clases de la siguiente manera:

- Se define una clase en el diagrama UML con el mismo nombre con el que está definida la tabla.
- Se añade a la clase que ha sido definida anteriormente, una lista de definiciones de atributos creada a partir de las definiciones de columnas de la tabla, siguiendo los siguientes pasos:
  - Cada columna se debe definir como un atributo UML que posee nombre, tipo de dato y valor inicial, donde el tipo de dato del atributo debe ser un tipo de dato UML equivalente al tipo de dato SQL con el cual se ha definido la columna.
  - Si la columna esta definida como clave primaria, entonces el atributo debe llevar el predicado pk\_, para que pueda ser diferenciado del resto de atributos.

Al aplicar la regla 1 al esquema BDOR presentado en §3.2.4, se obtiene el diagrama de clases UML de la figura 3-9.

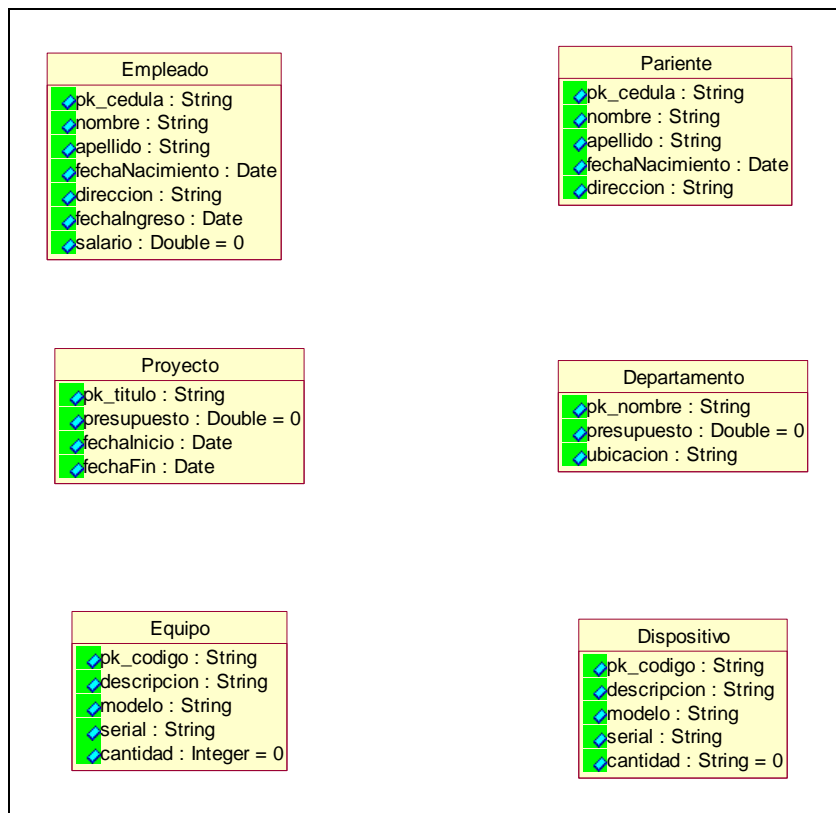


Figura 3-9. Clases creadas a partir del esquema BDOR.

Como se puede observar en la figura 3-9, el conjunto de clases resultante no están asociadas entre sí, por lo tanto, la regla que se presenta a continuación permite obtener del esquema BDOR el conjunto de asociaciones faltante.

**Regla 2. Representar las clases que poseen definiciones de columnas foráneas como asociaciones.**

Por cada clase del esquema BDOR que posea definiciones de claves foráneas que relacionan a dos tablas, se describe en el diagrama de clases de la siguiente manera:

- Se crea una asociación que lleva el mismo nombre de la tabla, la cual debe enlazar las clases que se corresponden con las tablas a las que hacen referencia dichas claves foráneas.
- Si la tabla posee definiciones de columnas que no forman parte de las claves foráneas, se debe crear una clase-asociación que contenga la lista de atributos que han sido derivados de estas columnas. Esta clase-asociación debe llevar el estereotipo <<association>> y su nombre debe ser igual al de la asociación.



El resultado final obtenido al aplicar la segunda regla, es el diagrama de clases UML de la figura 3-10, el cual describe una vista lógica de alto nivel del esquema BDOR.

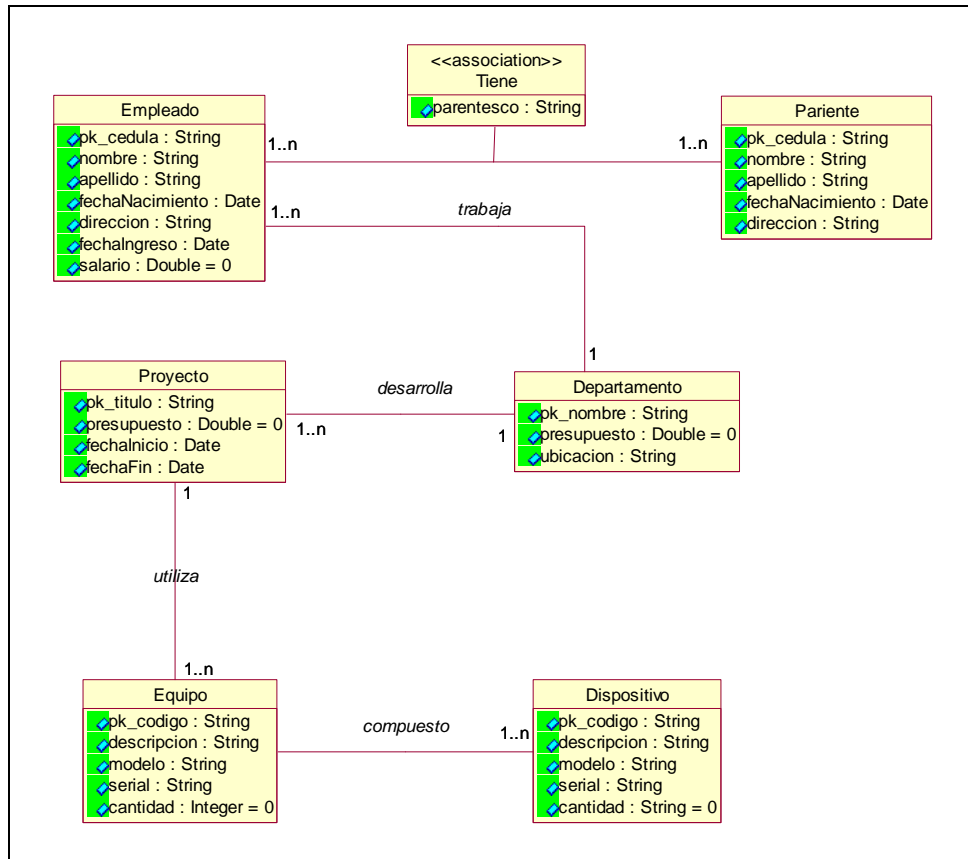


Figura 3-10. Asociaciones creadas a partir del esquema BDOR

### 3.3.4 Resultados de la transformación BDOR→UML

El diagrama de clases UML de la figura 3-10 obtenido de la transformación BDOR→UML, posee las siguientes características:

- No posee clases abstractas, ya que actualmente las BDOR actuales no soportan el concepto de herencia.
- Las clases que llevan el estereotipo <<association>> representan a las clases-asociación.



Ahora, si se compara el diagrama de clases que resulta de la transformación BDOR → UML con el diagrama de clases inicial, presentado en la figura 3-1 se puede observar lo siguiente:

- En el diagrama de clases, resultado de la transformación BDOR → UML, las clases que representan las entidades de negocio han heredado los atributos de las clases abstractas mostradas en el diagrama inicial.
- El diagrama de clases obtenido contiene todas las clases del diagrama inicial, a excepción de las clases abstractas. No obstante, el diagrama obtenido no contiene todas las asociaciones del diagrama inicial.

El hecho de que el diagrama de clases resultante de la transformación no contenga todas las asociaciones descritas en el diagrama de la figura 3-1 era de esperarse, ya que el esquema XML obtenido de la primera transformación (UML → XML), representa solamente una vista de las clases del diagrama. Esto quiere decir, que se necesitan generar varias vistas para obtener el conjunto de relaciones del esquema BDOR que completarán de esta manera las asociaciones faltantes.





## Capítulo 4. Diseño del prototipo

En este capítulo se presentan los resultados obtenidos en las primeras fases del proceso de desarrollo del modelo de procesos Watch, que están relacionadas con el diseño de la aplicación. El capítulo está dividido en cuatro secciones, correspondientes a las fases de: análisis y dominio de la aplicación, definición de requerimientos, análisis y especificación de requerimientos, y diseño del sistema.

En la primera parte se define el dominio de la aplicación y se identifican las entidades y los actores que forman parte de este dominio. Finalmente, se describen los elementos que conformarán la interfaz gráfica del prototipo y los detalles de la arquitectura del sistema a implementar.

### 4.1 Análisis del dominio de la aplicación

En esta sección se describe el dominio de la aplicación, en el cual el sistema de software se podría utilizar. El dominio involucra una serie de conceptos que se representan como un conjunto de entidades que son modeladas posteriormente usando UML.

#### 4.1.1 Definición del alcance del dominio

El dominio en el cual estará enfocado el prototipo, es el dominio correspondiente a las aplicaciones destinadas al diseño de esquemas XML y esquemas BDOR que utilizan UML como herramienta de modelado. Los documentos de esquemas generados por la aplicación constituirán las bases para poder lograr un intercambio bidireccional de datos entre documentos XML y bases de datos objeto-relacionales (BDOR).



### 4.1.2 Modelo Conceptual

El diagrama de la figura 4-1 muestra el conjunto de conceptos de negocio enmarcados dentro del dominio del sistema. Los conceptos son representados como un conjunto de entidades, las cuales se describen a continuación:

- *Usuario*: La persona que interactúa con la aplicación.
- *GUI*: Interfaz gráfica de la aplicación.
- *Diagrama UML*: Diagrama de clases UML.
- *Esquema XML*: Documento esquema XML.
- *Esquema BDOR*: Documento esquema BDOR.
- *Tabla*: Tabla de una BDOR.

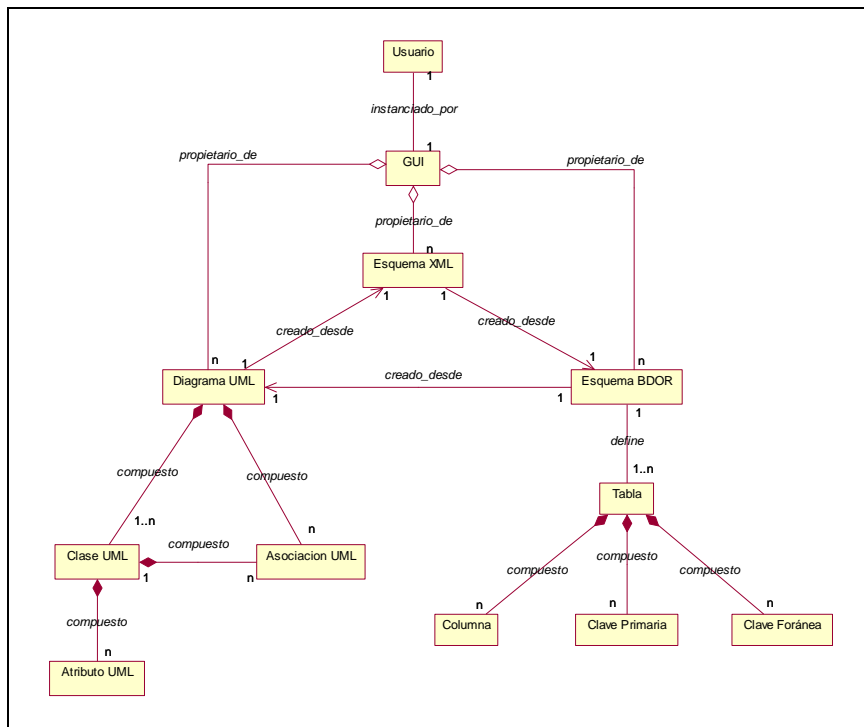


Figura 4-1. Modelo de entidades de negocio.

Las asociaciones presentes entre las distintas entidades se explican en la tabla 4-1.



Tabla 4-1. Asociaciones entre conceptos.

Asociación	Entidades involucradas	Explicación
<i>propietario_de</i>	GUI, Diagrama UML	La interfaz gráfica puede albergar uno o varios diagramas de clase UML.
<i>propietario_de</i>	GUI, Esquema XML	La interfaz gráfica puede albergar uno o varios documentos esquemas XML.
<i>propietario_de</i>	GUI, Esquema BDOR	La interfaz gráfica puede albergar uno o varios documentos esquemas BDOR.
<i>compuesto</i>	Diagrama UML, Clase UML	Un diagrama puede estar compuesto por una o varias clases UML.
<i>compuesto</i>	Diagrama UML, Asociación UML	Un diagrama puede estar compuesto por una o varias asociaciones UML.
<i>compuesto</i>	Asociación UML, Clase UML	Una clase UML puede estar compuesto por una o varias asociaciones UML.
<i>compuesto</i>	Clase UML, Atributo UML	Una clase UML puede estar compuesta por uno o varios atributos.
<i>creado_desde</i>	Esquema XML, Diagrama UML	Un esquema XML puede ser creado a partir de un diagrama de clases UML.
<i>creado_desde</i>	Esquema BDOR, Esquema XML	Un esquema BDOR puede ser creado a partir de un documento esquema XML.
<i>creado_desde</i>	Diagrama UML, Esquema BDOR	Un diagrama UML puede ser creado a partir de un documento esquema BDOR.
<i>define</i>	Esquema BDOR, Tabla	Un esquema BDOR define a un conjunto de tablas.
<i>compuesto</i>	Tabla, Columna	Una tabla esta compuesta por un conjunto de columnas.
<i>compuesto</i>	Tabla, Clave Primaria	Una tabla puede estar compuesta por un conjunto de claves primarias.
<i>compuesto</i>	Tabla, Clave Foránea	Una tabla puede estar compuesta por un conjunto de claves foráneas.

### 4.1.3 Definición del vocabulario del dominio

La tabla 4-2 muestra una recopilación de términos del dominio.





Tabla 4-2. Diccionario de términos del dominio.

Término	Categoría	Descripción
Diagrama UML	Tipo	Diagrama de clases UML.
Clase UML	Tipo	Clase que pertenece a un diagrama UML.
Asociación UML	Tipo	Asociación UML. Línea que enlaza dos clases en un diagrama UML.
Esquema XML	Tipo	Documento esquema XML.
Esquema BDOR	Tipo	Documento esquema BDOR.
Tabla	Tipo	Tabla de una BDOR.
Columna	Tipo	Atributo o campo de una tabla..
Clave primaria	Tipo	Atributo clave de una tabla.
Clave foránea	Tipo	Atributo foráneo de una tabla.
Crear diagrama UML	Caso de uso	Proceso en el cual un usuario construye un diagrama de clases UML.
Crear esquema XML desde diagrama UML	Caso de uso	Proceso en el cual se genera un documento esquema XML a partir de un diagrama de clases UML existente.
Crear esquema BDOR desde esquema XML	Caso de uso	Proceso en el cual se genera un documento esquema BDOR a partir de un documento esquema XML existente.
Crear diagrama UML desde catalogo BDOR	Caso de uso	Proceso en el cual se genera un diagrama de clases UML a partir de los metadatos contenidos en un catalogo BDOR.

## 4.2 Definición de requerimientos

En esta sección se definen informalmente los requerimientos funcionales y no funcionales que debe satisfacer la aplicación a implementar. Los requerimientos funcionales tienen que ver con las operaciones que se podrán realizar con la aplicación desde la perspectiva del usuario, mientras que los requerimientos no funcionales están asociados a las características que tendrá la aplicación.

### 4.2.1 Funciones básicas

La tabla 4-3 muestra una lista detallada de las características operacionales que debe ofrecer la aplicación.



Tabla 4-3. Funciones que debe ofrecer la aplicación

Ref #	Función	Categoría
R1.1	Crear un diagrama de clases UML	Evidente
R1.2	Abrir un diagrama de clases UML	Evidente
R1.3	Abrir un documento esquema XML	Evidente
R1.4	Abrir un documento esquema BDOR	Evidente
R1.5	Guardar un diagrama de clases UML	Evidente
R1.6	Guardar un documento esquema XML	Evidente
R1.7	Guardar un documento esquema BDOR	Evidente
R1.8	Imprimir un diagrama de clases UML	Evidente
R1.9	Imprimir un documento esquema XML	Evidente
R1.10	Imprimir un documento esquema BDOR	Evidente
R1.11	Crear vista	Evidente
R1.12	Destruir vista	Evidente
R1.13	Crear un documento esquema XML a partir de un diagrama de clases UML	Evidente
R1.14	Crear un documento esquema BDOR a partir de un documento esquema XML	Evidente
R1.15	Crear un diagrama de clases UML a partir de un catálogo BDOR	Evidente

La creación de un diagrama de clases UML implica el uso de otras funciones, las cuales son descritas en la tabla 4-2.

Tabla 4-2. Funciones relacionadas con la creación de diagramas UML

Ref #	Función	Categoría
R2.1	Crear clase	Evidente
R2.2	Seleccionar clase	Evidente
R2.3	Eliminar clase	Evidente
R2.4	Modificar clase	Evidente
R2.5	Crear asociación	Evidente
R2.6	Seleccionar asociación	Evidente
R2.7	Eliminar asociación	Evidente
R2.8	Modificar asociación	Evidente



### 4.2.2 Atributos del sistema

La aplicación deberá cumplir con ciertas características y algunas restricciones mostradas en la tabla 4-4.

Tabla 4-4. Atributos del sistema

Atributo	Detalles y restricciones de frontera
Metáfora de interfaz	Aplicación que provee una interfaz gráfica de usuario que soporta múltiples documentos.
Lenguaje de programación	Java.
Sistema operativo	Multiplataforma.
Acceso a bases de datos	MySQL, SQLServer, PostgreSQL y ORACLE
Tiempo de respuesta	Menor a cinco segundos.

## 4.3 Análisis y especificación de requerimientos

En esta sección se presentan los requerimientos de una manera técnica o formal. En la primera parte se presentan los casos de uso de alto nivel que describen en forma narrativa los procesos del dominio. Posteriormente se presenta un conjunto de diagramas de interacción para la asignación de responsabilidades.

### 4.3.1 Casos de uso

El diagrama de casos de uso de la figura 4-1, describe los procesos de negocio que podrá llevar a cabo un usuario. Este diagrama presenta a un único actor denominado como *Usuario*, que es la persona que interactúa con la aplicación.

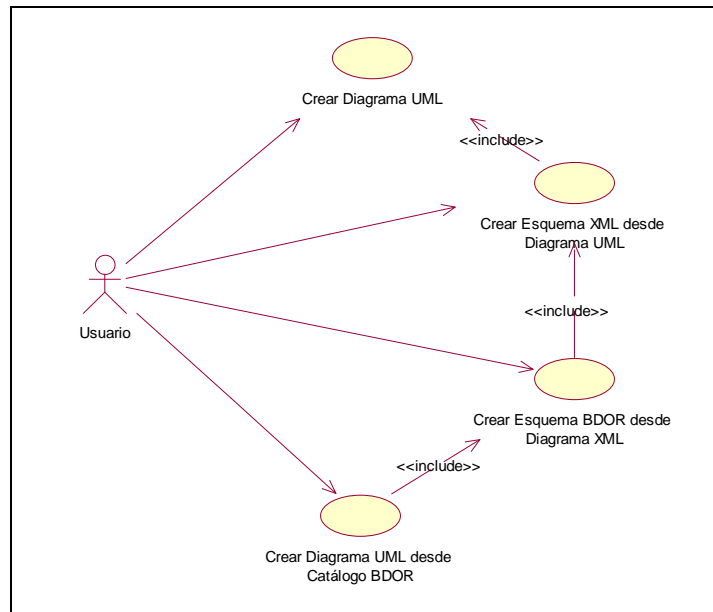


Figura 4-2. Diagrama de casos de uso.

Las tablas 4-5, 4-6, 4-7 y 4-8 explican cada uno de los procesos de negocio descritos en el diagrama de casos de usos de la figura 4-2.

Tabla 4-5. Caso de uso: Crear diagrama UML

<b>Caso de uso</b>	<b>Crear diagrama UML</b>
Actores	Usuario (iniciador).
Propósito	Crear un nuevo diagrama de clases UML.
Resumen	El usuario crea un nuevo diagrama de clases UML al cual puede añadir clases y añadir asociaciones entre clases. Las asociaciones entre clases pueden ser unidireccionales, bidireccionales, de agregación/composición o de especialización/generalización. Las clases y asociaciones que fueron añadidas pueden ser modificadas o eliminadas posteriormente, si se requiere. Una vez creado el diagrama el usuario puede imprimir o guardar el diagrama, si se requiere.
Tipo	Primario.
Referencias cruzadas	Funciones: R1.2, R1.5, R1.8, R2.1, R2.2, R2.3, R2.4, R2.5, R2.6, R2.7 y R2.8.



Tabla 4-6. Caso de uso: Crear esquema XML desde diagrama UML

Caso de uso	Crear esquema XML desde diagrama UML
Actores	Usuario (iniciador).
Propósito	Crear un documento esquema XML a partir de un diagrama de clases UML.
Resumen	El usuario especifica inicialmente sobre el diagrama de clases la vista que definirá la estructura del documento esquema XML. Esto lo hace asignando pesos a las asociaciones del diagrama de clases. Las asociaciones con menor peso son las que constituirán la vista. Luego, selecciona la clase que será el punto inicial de todos los recorridos y finalmente crea la vista. La vista es descrita gráficamente como un conjunto de recorridos no cíclicos sobre las clases y asociaciones del diagrama de clases UML. El usuario puede destruir y volver a construir la vista, si se requiere. Finalmente, el usuario asigna el nombre y la ubicación donde será creado el documento esquema XML. Posteriormente puede abrir e imprimir el documento esquema XML creado.
Tipo	Primario y esencial.
Referencias cruzadas	Funciones: R1.3, R1.6, R1.9, R1.21, R1.12 y R1.13.

Tabla 4-7. Caso de uso: Crear esquema BDOR desde esquema XML.

Caso de uso	Crear esquema BDOR desde esquema XML
Actores	Usuario (iniciador).
Propósito	Generar un esquema BDOR a partir de un documento esquema XML.
Resumen	El usuario selecciona inicialmente el manejador de bases de datos o SGMDOR que definirá la sintaxis del lenguaje SQL que será usado para generar el esquema BDOR. Finalmente, realiza la operación de conversión. El sistema muestra el esquema XML como un documento en formato de texto plano. Posteriormente el usuario puede guardar o imprimir el documento creado, si se requiere.
Tipo	Primario y esencial.
Referencias cruzadas	Funciones: R1.4, R1.10 y R1.14.



Tabla 4-8. Caso de uso: Crear diagrama UML desde esquema BDOR

<b>Caso de uso</b>	<b>Crear diagrama UML desde un catalogo BDOR</b>
Actores	Usuario (iniciador).
Propósito	Crear un diagrama de clases UML a partir del diccionario de datos o catálogo de una BDOR.
Resumen	El usuario selecciona el sistema de gestión de bases de datos relacionales a utilizar. Luego, indica el nombre del catálogo inicial, el nombre de usuario y contraseña con el que se accederá a la base de datos. Si la operación de conexión es satisfactoria, el sistema crea el diagrama de clases UML correspondiente al esquema físico de la BDOR. Posteriormente, el usuario puede guardar o imprimir el diagrama creado, si se requiere.
Tipo	Primario y esencial.
Referencias cruzadas	Funciones: R1.2, R1.5, R1.8 y R1.15.

### 4.3.2 Diagramas de interacción

En las siguientes secciones se presentan los diagramas de colaboración para la asignación de responsabilidades. Las funciones: *Guardar*, *Guardar como* e *Imprimir*, descritas en los diagramas de colaboración de las figuras 4-5, 4-6 y 4-7, también son válidas para las instancias de las clases *EsquemaXML* y *EsquemaBDOR*, en consecuencia se omitirán tales diagramas de colaboración para dichas instancias.

#### Diagrama de colaboración: nuevoDiagramaUML

La operación del sistema nuevoDiagramaUML ocurre cuando un usuario necesita construir un nuevo diagrama de clases UML (ver figura 4-3).

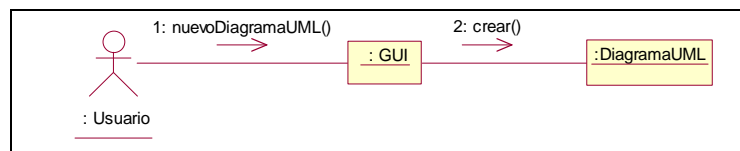


Figura 4-3. Diagrama de Colaboración: nuevoDiagramaUML

En la tabla 4-9 se presenta el contrato de uso para la operación nuevoDiagramaUML.



Tabla 4-9. Contrato de uso: nuevoDiagramaUML

Nombre	nuevoDiagramaUML()
Responsabilidades	Crear un nuevo diagrama UML.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.1
Poscondiciones	Se crea una nueva instancia de DiagramaUML.

### Diagrama de colaboración: abrir

La operación del sistema `abrir` ocurre cuando un usuario necesita abrir un archivo. Para realizar esta operación el nombre del archivo debe tener como extensión: `uml`, `xsd` o `sql` y debe haber sido generado por la aplicación (ver figura 4-4).

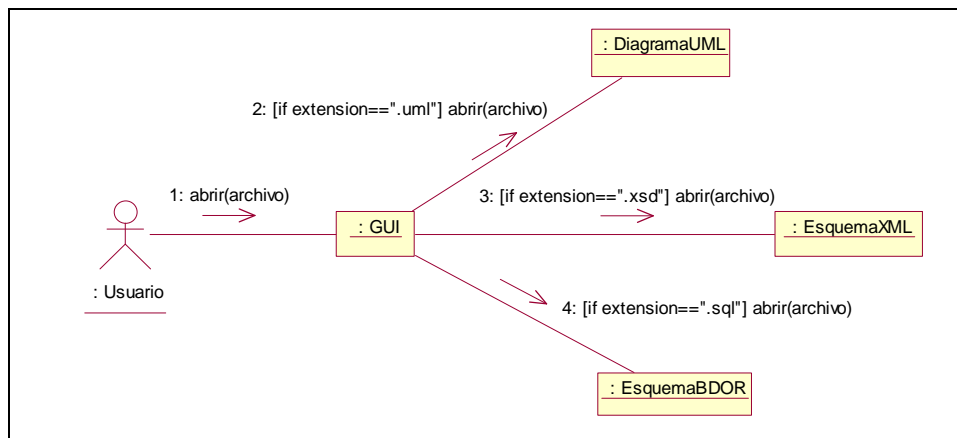


Figura 4-4. Diagrama de Colaboración: abrir

En la tabla 4-10 se presenta el contrato de uso para la operación `abrir`.

Tabla 4-10. Contrato de uso: abrir

Nombre	<b>abrir (Archivo: archivo)</b>
Responsabilidades	Abrir un archivo.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.2, R1.3 y R1.4
Excepciones	Se produce un error si el archivo esta dañado o corrupto.
Precondiciones	El argumento <code>archivo</code> debe ser conocido.



Poscondiciones	<ul style="list-style-type: none"> <li>- Se crea una instancia de DiagramaUML a partir del archivo, si el archivo tiene extensión la cadena “.uml”.</li> <li>- Se crea una instancia de EsquemaXML a partir del archivo, si el archivo tiene como extensión la cadena “.xsd”.</li> <li>- Se crea una instancia de EsquemaBDOR a partir del archivo, si el archivo tiene como extensión la cadena “.sql”.</li> </ul>
----------------	---

### Diagrama de colaboración: guardarDiagramaUML

La operación del sistema guardarDiagramaUML ocurre cuando un usuario necesita guardar un diagrama de clases UML como un archivo (ver figura 4-5).

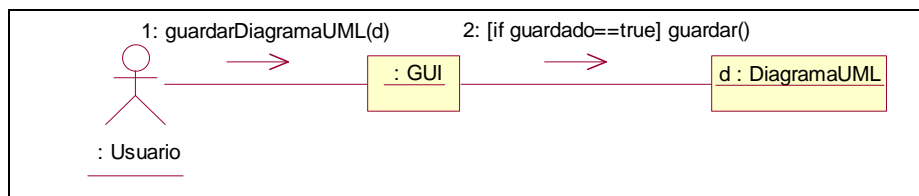


Figura 4-5. Diagrama de Colaboración: guardarDiagramaUML

En la tabla 4-11 se presenta el contrato de uso para la operación guardarDiagramaUML.

Tabla 4-11. Contrato de uso: guardarDiagramaUML

<b>Nombre</b>	<b>guardarDiagramaUML (DiagramaUML: diagramaUML)</b>
Responsabilidades	Guardar un diagrama de clases UML.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.5
Excepciones	Se produce una excepción si existen errores de entrada salida.
Precondiciones	<ul style="list-style-type: none"> <li>- El argumento diagramaUML debe ser conocido.</li> <li>- El diagrama de clases ya ha sido guardado como un archivo.</li> </ul>





### Diagrama de colaboración: guardarDiagramaUMLcomo

La operación del sistema guardarDiagramaUMLcomo ocurre cuando un usuario necesita guardar un diagrama de clases UML como un archivo en cierta ubicación del sistema de archivos (ver figura 4-6).

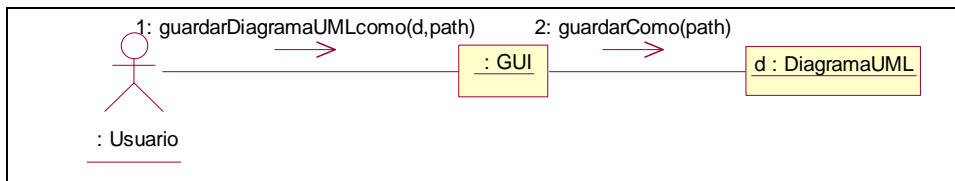


Figura 4-6. Diagrama de Colaboración: guardarDiagramaUMLcomo

En la tabla 4-12 se presenta el contrato de uso para la operación guardarDiagramaUMLcomo.

Tabla 4-12. Contrato de uso: guardarDiagramaUMLcomo

Nombre	<b>guardarDiagramaUMLcomo (DiagramaUML: diagramaUML, Cadena: path)</b>
Responsabilidades	Guardar un diagrama de clases UML en una ubicación específica del sistema de archivos.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.2
Excepciones	Se produce una excepción si existen errores de entrada salida.
Precondiciones	- Los argumentos <code>diagramaUML</code> y <code>path</code> deben ser conocidos.
Poscondiciones	- Se crea un archivo con extensión “.uml”, en la ruta de directorio seleccionada por el usuario.

### Diagrama de colaboración: imprimirDiagramaUML

La operación del sistema imprimirDiagramaUML ocurre cuando un usuario necesita imprimir por impresora un diagrama de clases UML (ver figura 4-7).

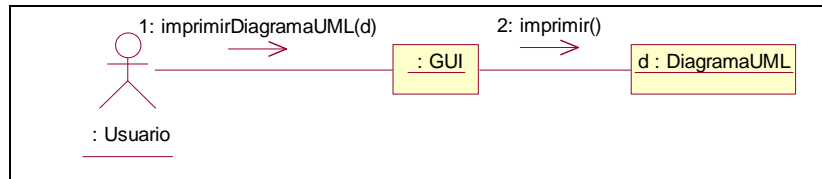


Figura 4-7. Diagrama de Colaboración: imprimirDiagramaUML

En la tabla 4-13 se presenta el contrato de uso para la operación imprimirDiagramaUML.

Tabla 4-13. Contrato de uso: imprimirDiagramaUML

Nombre	<b>imprimirDiagramaUML(DiagramaUML: diagramaUML)</b>
Responsabilidades	Imprimir un diagrama UML por impresora.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.8

### Diagrama de colaboración: crearClaseUML

La operación del sistema crearClaseUML ocurre cuando un usuario necesita añadir o crear una clase en un diagrama UML (ver figura 4-8).

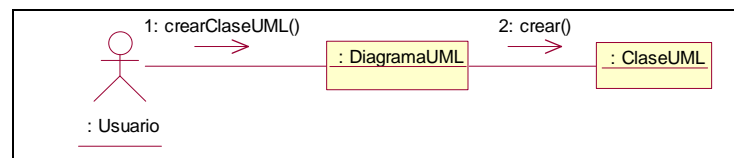


Figura 4-8. Diagrama de Colaboración: crearClaseUML

En la tabla 4-14 se presenta el contrato de uso para la operación crearClaseUML.

Tabla 4-14. Contrato de uso: crearClaseUML

Nombre	<b>crearClaseUML()</b>
Responsabilidades	Crear una nueva clase sobre un diagrama existente.
Tipo	DiagramaUML
Referencias cruzadas	Funciones del sistema: R2.1
Poscondiciones	- Se crea una nueva instancia de ClaseUML.



### Diagrama de colaboración: modificarClaseUML

La operación del sistema `modificarClaseUML` ocurre cuando un usuario necesita modificar las propiedades de una clase UML existente. La aplicación presenta al usuario una ventana de diálogo en la cual puede modificar las propiedades: nombre, estereotipo, superclase y lista de atributos de la clase (ver figura 4-9).

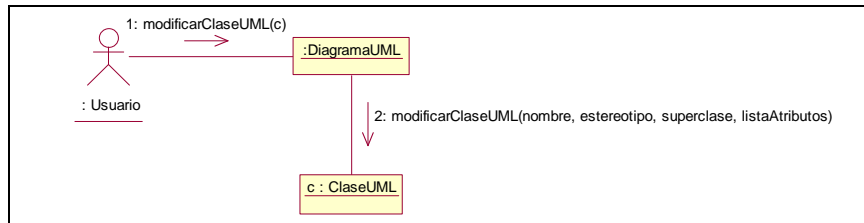


Figura 4-9. Diagrama de Colaboración: `modificarClaseUML`

En la tabla 4-15 se presenta el contrato de uso para la operación `modificarClaseUML`.

Tabla 4-15. Contrato de uso: `modificarClaseUML`

Nombre	<code>modificarClaseUML(ClaseUML: clase)</code>
Responsabilidades	Modificar las propiedades de una clase UML existente.
Tipo	DiagramaUML
Referencias cruzadas	Funciones del sistema: R2.4
Poscondiciones	- Se asigna a la instancia <code>clase</code> las nuevas propiedades.

### Diagrama de colaboración: eliminarClaseUML

La operación del sistema `eliminarClaseUML` ocurre cuando un usuario necesita eliminar una clase del diagrama de clases UML (ver figura 4-9).

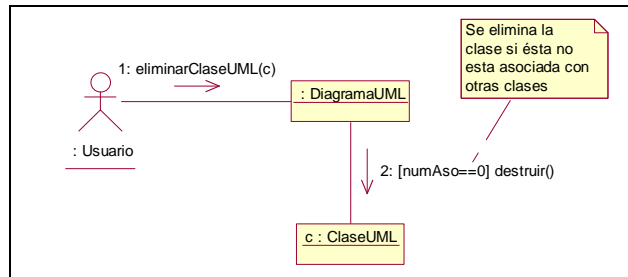


Figura 4-10. Diagrama de Colaboración: eliminarClaseUML

En la tabla 4-16 se presenta el contrato de uso para la operación eliminarClaseUML.

Tabla 4-16. Contrato de uso: eliminarClaseUML

<b>Nombre</b>	<b>eliminarClaseUML(ClaseUML: clase)</b>
<b>Responsabilidades</b>	Eliminar una clase UML del diagrama.
<b>Tipo</b>	DiagramaUML
<b>Referencias cruzadas</b>	Funciones del sistema: R2.3
<b>Precondiciones</b>	- El argumento <code>clase</code> debe ser conocido. - El número de asociaciones de la instancia <code>clase</code> debe ser igual a cero.
<b>Poscondiciones</b>	- Se elimina la clase del diagrama UML.

### Diagrama de colaboración: crearAsociacionUML

La operación del sistema crearAsociacionUML ocurre cuando un usuario necesita crear una asociación entre dos clases del diagrama UML (ver figura 4-11).

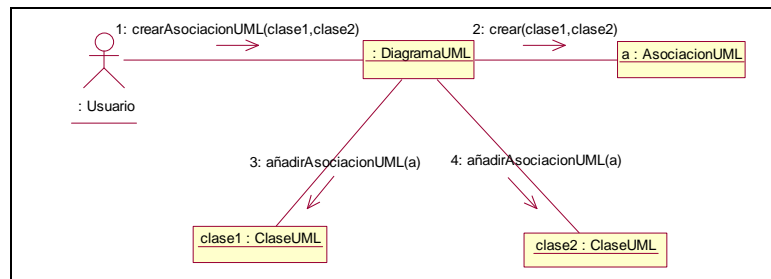


Figura 4-11. Diagrama de Colaboración: crearAsociacionUML



En la tabla 4-17 se presenta el contrato de uso para la operación crearAsociacionUML.

Tabla 4-17. Contrato de uso: crearAsociacionUML

Nombre	crearAsociacionUML( ClaseUML: clase1, ClaseUML: clase2)
Responsabilidades	Crear una asociación entre dos clases existentes de un diagrama de clases UML.
Tipo	DiagramaUML
Referencias cruzadas	Funciones del sistema: R2.5
Precondiciones	- Los argumentos clase1 y clase2 deben ser conocidos.
Poscondiciones	- Se crea una nueva instancia de AsociacionUML. - Se añaden a clase1 y a clase2 la instancia creada.

### Diagrama de colaboración: modificarAsociaciónUML

La operación del sistema modificarAsociacionUML ocurre cuando un usuario necesita modificar las propiedades de una asociación UML. La aplicación presenta al usuario una ventana de diálogo en la cual puede modificar las propiedades: nombre de la asociación, rol de la clase 1, rol de la clase 2, cardinalidad de la clase 1 y cardinalidad de la clase 2 (ver figura 4-12).

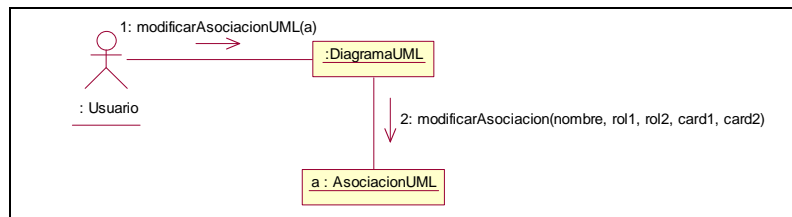


Figura 4-12. Diagrama de Colaboración: modificarAsociacionUML.

En la tabla 4-18 se presenta el contrato de uso para la operación modificarAsociacionUML.

Tabla 4-18. Contrato de uso: modificarAsociacionUML

Nombre	modificarAsociacionUML(AsociacionUML: asociacion)
Responsabilidades	Modificar las propiedades de una asociación.



Tipo	DiagramaUML
Referencias cruzadas	Funciones del sistema: R2.8
Precondiciones	- El argumento <code>asociacion</code> debe ser conocido.
Poscondiciones	- Se asigna a la instancia <code>asociacion</code> las nuevas propiedades.

### Diagrama de colaboración: eliminarAsociaciónUML

La operación del sistema `eliminarAsociacionUML` ocurre cuando un usuario necesita destruir una asociación entre dos clases del diagrama UML (ver figura 4-13).

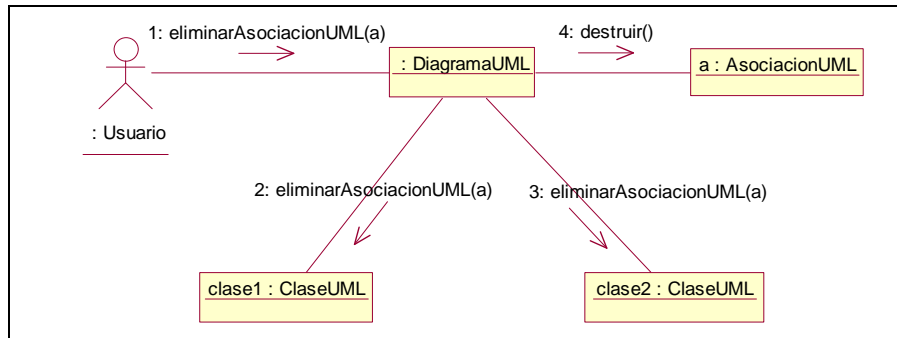


Figura 4-13. Diagrama de Colaboración: `eliminarAsociacionUML`

En la tabla 4-19 se presenta el contrato de uso para la operación `eliminarAsociacionUML`.

Tabla 4-19. Contrato de uso: `eliminarAsociacionUML`

Nombre	<b><code>eliminarAsociacionUML(AsociacionUML: asociacion)</code></b>
Responsabilidades	Eliminar una asociación existente.
Tipo	DiagramaUML
Referencias cruzadas	Funciones del sistema: R2.7
Precondiciones	- El argumento <code>asociacion</code> debe ser conocido.
Poscondiciones	- Se elimina la instancia <code>asociacion</code> de las clases involucradas. - Se elimina la instancia <code>asociacion</code> del diagrama UML.



### Diagrama de colaboración: crearVista

La operación del sistema `crearVista` ocurre cuando un usuario necesita crear la vista que define la estructura del documento esquema XML sobre un diagrama de clases existente (ver figura 4-11).

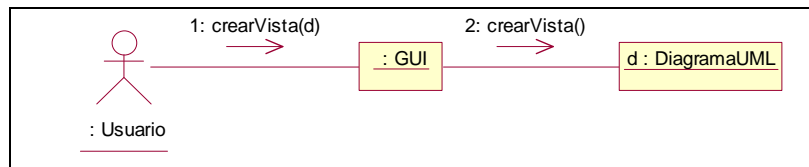


Figura 4-14. Diagrama de Colaboración: `crearVista`

En la tabla 4-20 se presenta el contrato de uso para la operación `crearVista`.

Tabla 4-20. Contrato de uso: `crearVista`

Nombre	<code>crearVista(DiagramaUML: diagrama)</code>
Responsabilidades	Crear una vista sobre un diagrama de clases existente.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.11
Precondiciones	- El argumento <code>diagrama</code> debe ser conocido.
Poscondiciones	- Se genera la vista sobre el la instancia <code>diagrama</code> .

### Diagrama de colaboración: destruirVista

La operación del sistema `destruirVista` ocurre cuando un usuario necesita deshacer una vista que ha sido creada anteriormente (ver figura 4-15).

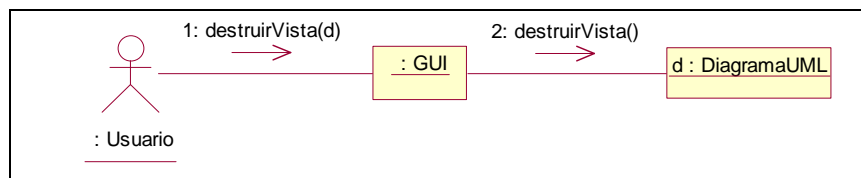


Figura 4-15. Diagrama de Colaboración: `destruirVista`

En la tabla 4-21 se presenta el contrato de uso para la operación `destruirVista`.



Tabla 4-21. Contrato de uso: destruirVista

Nombre	<b>destruirVista(DiagramaUML: diagrama)</b>
Responsabilidades	Deshacer la vista sobre un diagrama de clases.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.2
Precondiciones	- El argumento <i>diagrama</i> debe ser conocido.
Poscondiciones	- Se destruye la vista sobre la instancia <i>diagrama</i> .

### Diagrama de colaboración: crearEsquemaXMLdesdeDiagramaUML

La operación del sistema `crearEsquemaXMLdesdeDiagramaUML` ocurre cuando un usuario necesita crear un documento esquema XML a partir de un diagrama de clases UML (ver figura 4-16).

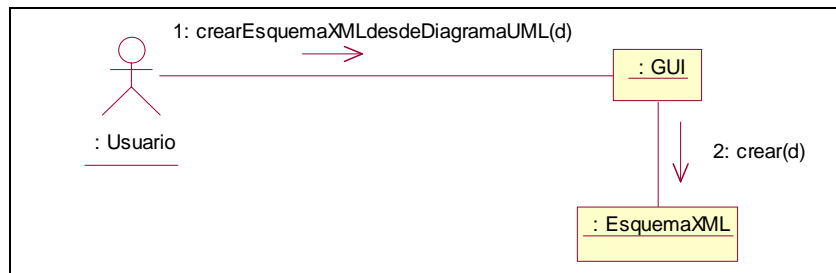


Figura 4-16. Diagrama de Colaboración: `crearEsquemaXMLdesdeDiagramaUML`

En la tabla 4-22 se presenta el contrato de uso para la operación `crearEsquemaXMLdesdeDiagramaUML`.

Tabla 4-22. Contrato de uso: `crearEsquemaXMLdesdeDiagramaUML`

Nombre	<b>crearEsquemaXMLdesdeDiagramaUML(DiagramaUML: diagrama)</b>
Responsabilidades	Crear un documento esquema XML a partir de un diagrama de clases UML.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.13 Caso de uso: <code>CrearEsquemaXMLdesdeDiagramaUML</code>
Precondiciones	- El argumento <i>diagrama</i> debe ser conocido.
Poscondiciones	- Se crea una instancia de <code>EsquemaXML</code> .





### Diagrama de colaboración: crearEsquemaBDORdesdeEsquemaXML

La operación del sistema crearEsquemaBDORdesdeEsquemaXML ocurre cuando un usuario necesita crear un documento esquema BDOR a partir de un documento esquema XML (ver figura 4-17).

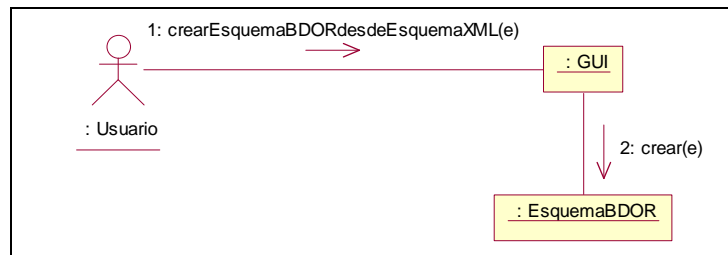


Figura 4-17. Diagrama de Colaboración: crearEsquemaBDORdesdeEsquemaXML

En la tabla 4-23 se presenta el contrato de uso para la operación crearEsquemaBDORdesdeEsquemaXML.

Tabla 4-23. Contrato de uso: crearEsquemaBDORdesdeEsquemaXML

<b>Nombre</b>	<b>crearEsquemaBDORdesdeEsquemaXML(EsquemaXML: esquema)</b>
Responsabilidades	Crear un documento esquema BDOR a partir de un documento esquema XML.
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.14 Caso de uso: CrearEsquemaBDORdesdeEsquemaXML
Precondiciones	- El argumento esquema debe ser conocido.
Poscondiciones	- Se crea una instancia de EsquemaBDOR.

### Diagrama de colaboración: crearDiagramaUMLdesdeCatalogoBDOR

La operación del sistema crearDiagramaUMLdesdeCatalogoBDOR ocurre cuando un usuario necesita crear un diagrama UML a partir del catálogo de una BDOR. La aplicación presenta al usuario una ventana de diálogo en la cual debe especificar el URL, catálogo, usuario y contraseña de la BDOR (ver figura 4-18).

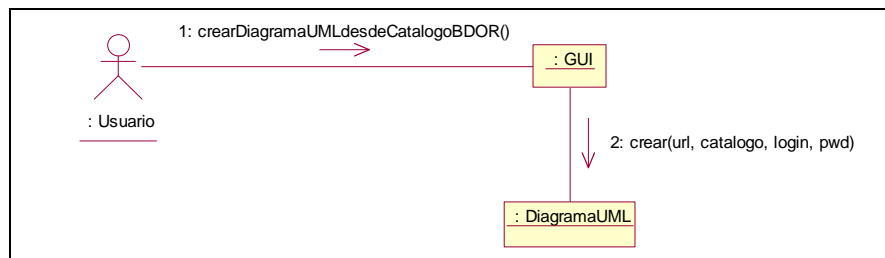


Figura 4-18. Diagrama de Colaboración: crearDiagramaUMLdesdeCatalogoBDOR

En la tabla 4-24 se presenta el contrato de uso para la operación crearDiagramaUMLdesdeCatalogoBDOR.

Tabla 4-24. Contrato de uso: crearDiagramaUMLdesdeCatalogoBDOR

Nombre	crearDiagramaUMLdesdeCatalogoBDOR()
Responsabilidades	Crear un diagrama UML a partir del catálogo de una BDOR
Tipo	GUI
Referencias cruzadas	Funciones del sistema: R1.2
Excepciones	Se genera un error de conexión si no se puede acceder a la BDOR.
Poscondiciones	- Se crea una instancia de EsquemaBDOR.

### 4.3.3 Diagramas de actividades

Las figuras 4-19, 4-20 y 4-21 describen los diagramas de actividades para los casos de uso:

- Crear esquema XML desde diagrama UML.
- Crear esquema BDOR desde esquema XML.
- Crear diagrama UML desde catálogo BDOR.

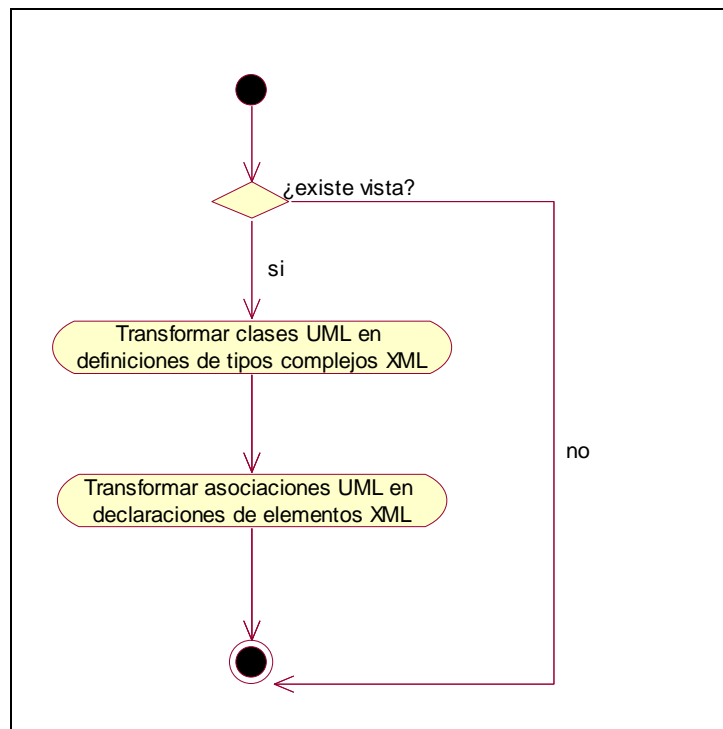


Figura 4-19. Diagrama de Actividades: Crear esquema XML desde diagrama UML

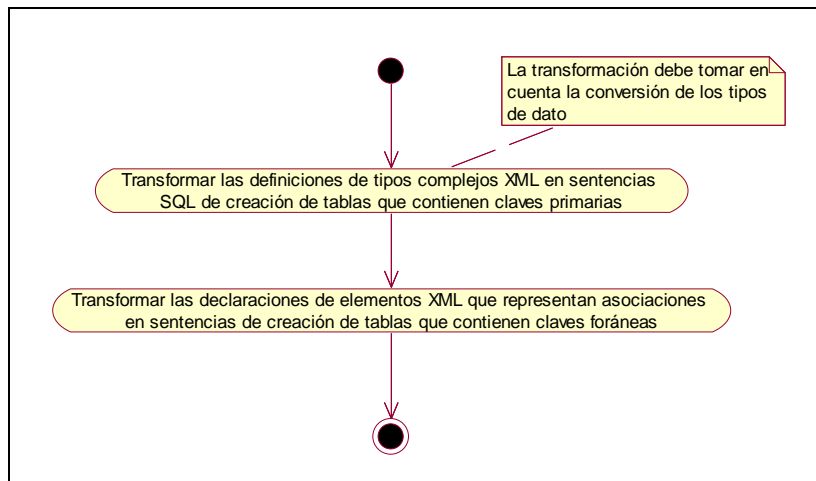


Figura 4-19. Diagrama de Actividades: Crear esquema BDOR desde esquema XML.

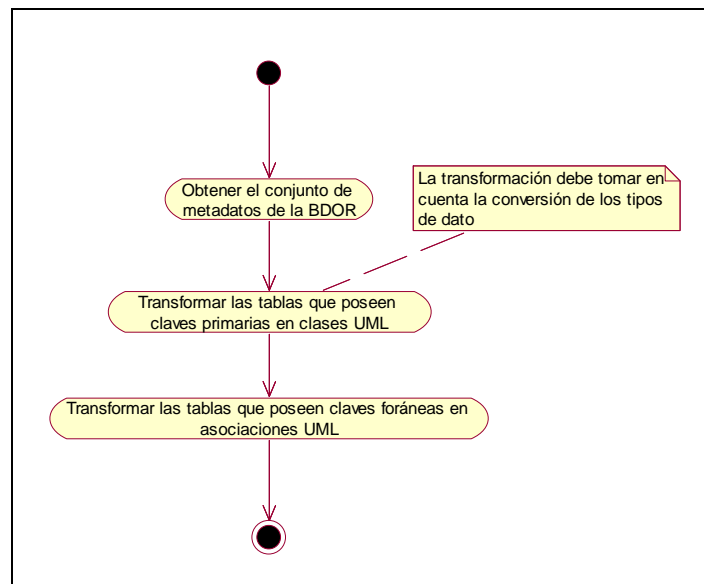


Figura 4-20. Diagrama de Actividades: Crear diagrama UML desde catalogo BDOR.

#### 4.3.4 Diagrama de clases

El diagrama de clases de la figura 4-21 describe las clases de las entidades de negocio a las cuales se le han asignado atributos y operaciones, como resultado de la identificación de responsabilidades a través de los diagramas de colaboración de la sección §4.2.2.

En este diagrama se puede observar que no fueron representadas las entidades Tabla, Columna, Clave Primaria y Clave Foránea que existían en el modelo conceptual de la figura 4-1, ya que estas entidades existen fuera del contexto de la aplicación, sin embargo, éstas se encuentran entre los límites del dominio.

También se puede observar que las asociaciones entre las entidades Diagrama UML, Esquema XML y Esquema BDOR no aparecen en el diagrama de clases de la figura 4-21, ya que existen operaciones que definen implícitamente estas relaciones a través de sus argumentos. Por ejemplo, la operación `crear(EsquemaXML: arg)` de la clase `EsquemaBDOR` representa la relación `creado_desde` entre las entidades `EsquemaBDOR` y `Esquema XML` del modelo conceptual de la figura 4-21.

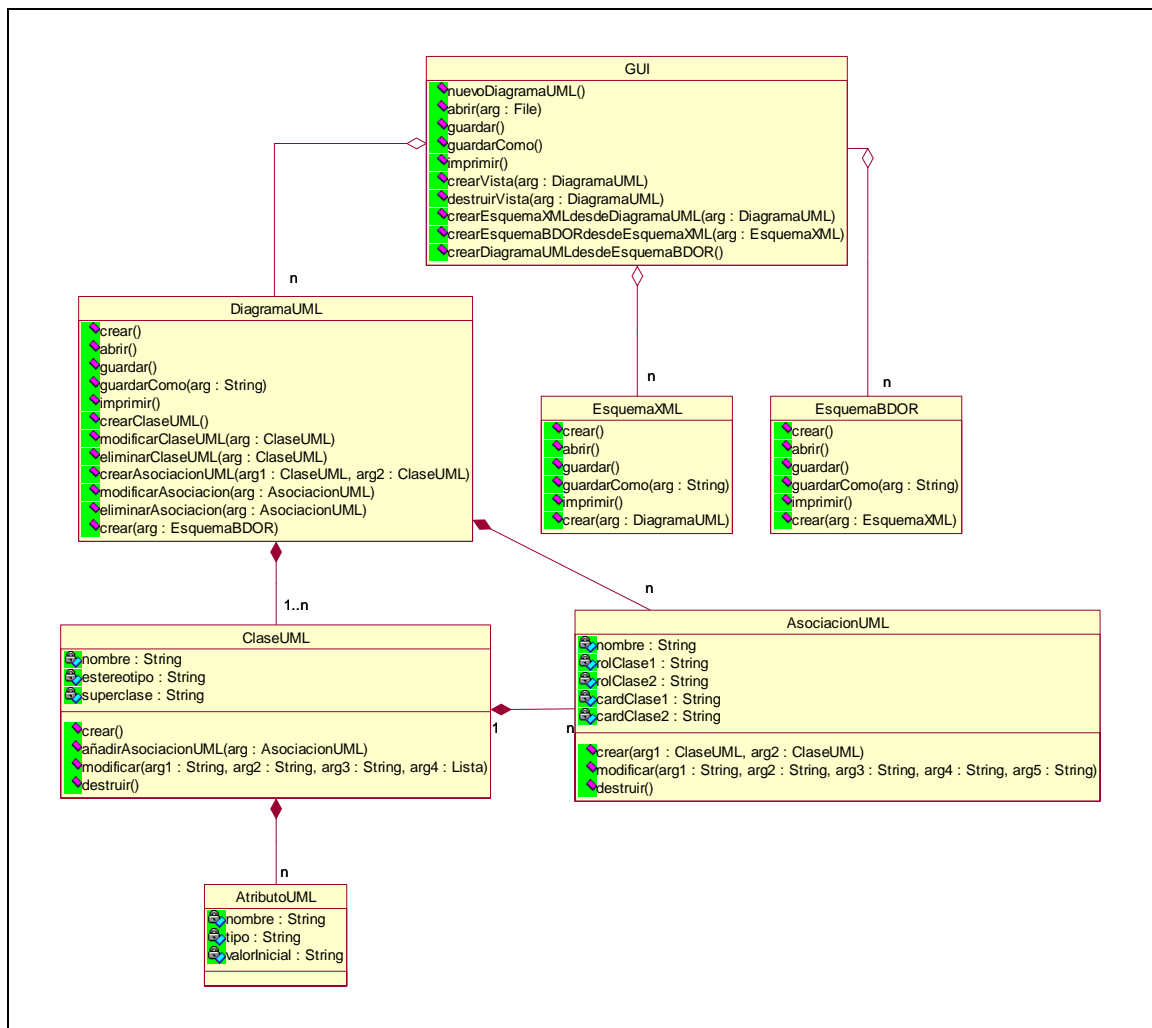


Figura 4-21. Diagrama de clases de negocio con atributos y operaciones.

## 4.4 Diseño del sistema

En esta última sección se presentan las especificaciones de diseño de la aplicación correspondientes a los detalles de la interfaz gráfica de usuario y la posible arquitectura del software.



#### 4.4.1 Diseño de la interfaz de usuario

La interfaz gráfica de usuario deberá ser una interfaz de múltiples documentos, la cual deberá poseer las siguientes características visuales (ver figura 4-22):

- *Barra de título:* Es la barra que se encuentra ubicada en la parte superior de la ventana principal y en ella se muestra el nombre de la aplicación y el título de la ventana activa.
- *Barra de menú:* La barra de menú se localiza debajo de la barra de título y está compuesta por un conjunto de menús desplegables que contienen todas las funciones que se pueden realizar con el sistema.
- *Barra de herramientas estándar:* Esta barra de herramientas está localizada debajo de la barra de menú y está compuesta por un conjunto de botones que permiten acceder de forma rápida las funciones que se encuentran en la barra de menú.
- *Barra de herramientas UML:* Esta barra de herramientas es una barra vertical ubicada en la parte izquierda de la ventana principal y está compuesta por un conjunto de botones que permiten realizar las tareas relacionadas con la creación de diagramas UML.

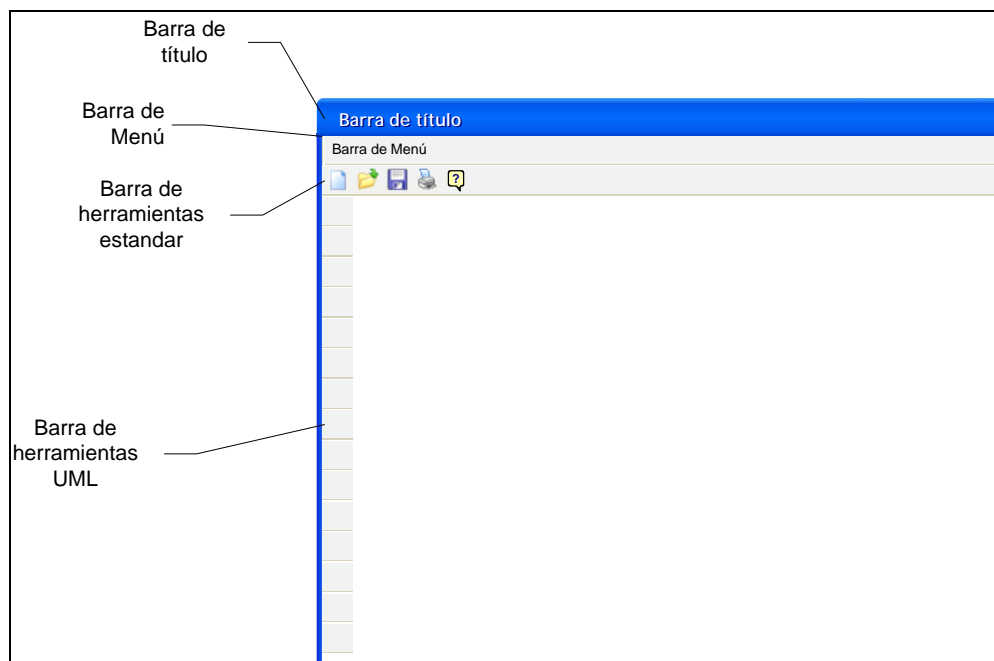


Figura 4-22. Características gráficas de la interfaz gráfica de usuario.



### Barra de Menú

La barra de menú estará compuesta por un conjunto de menús desplegables. Cada menú de la barra puede desplegar en forma vertical un conjunto de submenús. Los submenús también pueden contener a otros submenús y así sucesivamente (ver figura 4.23).

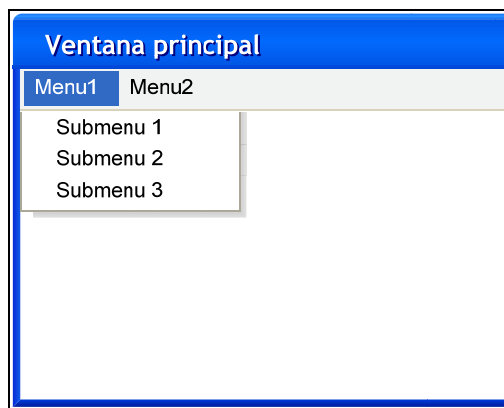


Figura 4-23. Modelo de menú desplegable.

### Barra de herramientas estándar

Esta barra estará compuesta por un conjunto de botones que permiten acceder directamente a las operaciones de los menús más utilizadas por los usuarios, tales como *Abrir*, *Guardar* e *Imprimir*, entre otras (ver figura 4-24).

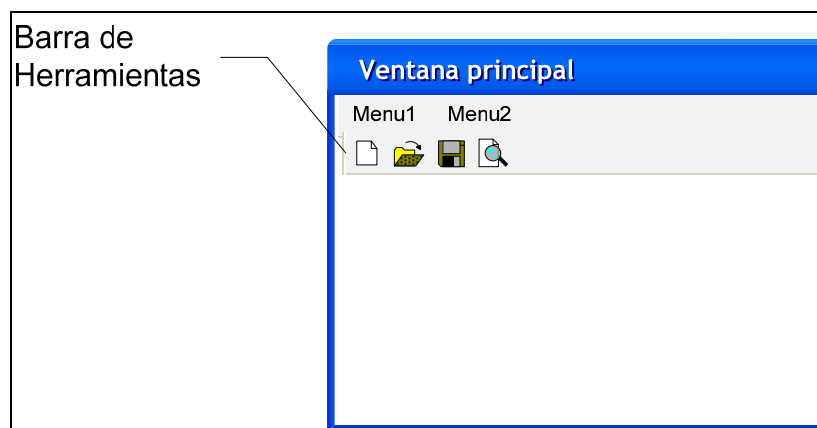









Figura 4-24. Modelo de barra de herramientas.



### Barra de herramientas UML

Esta barra estará compuesta por un conjunto de botones que permiten realizar las operaciones comunes utilizadas en la construcción de diagramas de clase UML. Los posibles íconos a utilizar se describen en la tabla 4-25.

Tabla 4-25. Modelo de íconos para la barra de herramientas UML.

Icono	Descripción
	Seleccionar
	Clase
	Asociación bidireccional
	Asociación unidireccional
	Generalización/Especialización
	Agregación
	Composición

El tamaño de los íconos de la barra de herramientas debe estar entre 16x16 y 32x32. La composición de colores debe ser mayor o igual a 256 colores.

### Estilos para el diseño de las ventanas

Los tipos de fuente a utilizar en las ventanas, menús, botones, etiquetas, cuadros de texto deben ser tipos estándar, tales como Arial y Times. Se recomienda usar solo tres tipos de fuente. El tamaño de fuente debe estar entre 10 y 12 y como máximo se utilizará cuatro tamaños. El color de fuente a utilizar es el negro (ver figura 4-25).



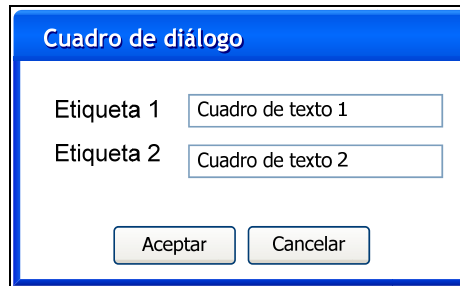


Figura 4-25. Modelo de cuadro de diálogo.

## Mensajes

Se hará uso de mensajes de información, de advertencia, de interrogación y de error para la interacción con el usuario. Los mensajes de advertencia pueden mostrarse al momento que un usuario desee salir de la aplicación. Los mensajes de información pueden ser usados para dar a conocer al usuario que se ha llevado a cabo con éxito la realización de alguna tarea. La interacción con los usuarios se realiza a través de los mensajes de interrogación. Solo se deben mostrar mensajes de error cuando la aplicación haya detectado un error crítico que el usuario no pueda resolver. Las figuras 4-25, 4-26, 4-27 y 4-28 muestran los posibles modelos de mensajes a utilizar.

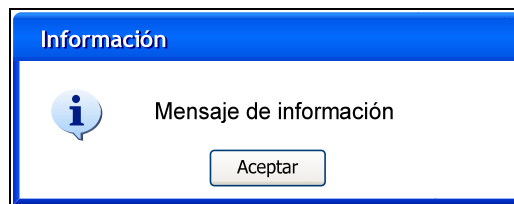


Figura 4-26. Modelo de mensaje de información.

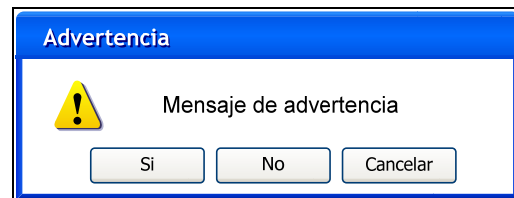


Figura 4-27. Modelo de mensaje de advertencia.

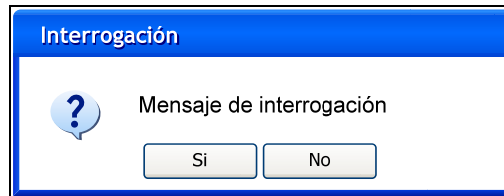


Figura 4-28. Modelo de mensaje de interrogación.

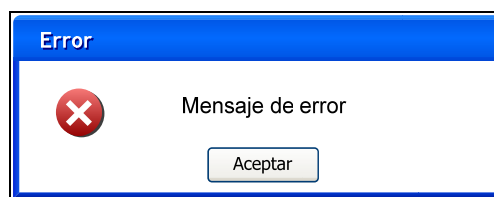


Figura 4-29. Modelo de mensaje de error.

#### 4.4.2 Diseño de la arquitectura

El sistema tendrá una arquitectura simple de dos capas. Las arquitecturas de dos capas consisten de tres componentes distribuidos en dos capas: cliente (solicitador de servicios) y servidor (proveedor de servicios). Los tres componentes son:

1. *Presentación*: Interfaz gráfica del sistema.
2. *Lógica de aplicaciones*: Tareas y reglas que rigen el proceso.
3. *Almacenamiento*: Mecanismos de almacenamiento persistente.

La figura 4-30 presenta los paquetes que conforman cada una de las capas de la arquitectura.

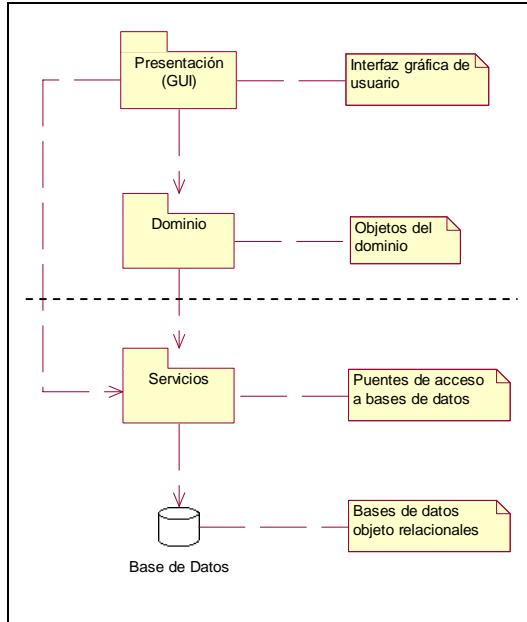


Figura 4-30. Capas de la arquitectura representadas en paquetes UML





## Capítulo 5. Implementación

En este capítulo se presentan los resultados obtenidos en las fases de implementación y pruebas del sistema. La herramienta de software implementada tiene el nombre de Trilogía y fue programada usando el lenguaje de programación Java. Trilogía tiene una interfaz gráfica que permite a los usuarios diseñar diagramas de clases UML, los cuales son utilizados para generar esquemas XML y esquemas BDOR. Los esquemas BDOR obtenidos con esta herramienta son compatibles con los sistemas de gestión de bases de datos: MySQL, SQLServer, PostgreSQL y ORACLE.

### 5.1 Herramientas de desarrollo

La tabla 5-1 presenta las herramientas de desarrollo y el lenguaje de programación utilizados para la implementación de la aplicación.

Tabla 5-1. Herramientas utilizadas para el desarrollo de la aplicación

Nombre	Descripción
JDK 1.5.0	Java Development Kit – Versión 1.5.0
Eclipse - IBM	Ambiente de Desarrollo Integrado Versión 3.0.0
mysql-connector-java-3.0.14-production-bin.jar	Conector JDBC para bases de datos MySQL.
mssqlserver.jar, msbase.jar, msutil.jar	Conector JDBC para bases de datos MS SQLServer
classes12.jar	Conector JDBC para bases de datos ORACLE
jdbc7.1-1.2.jar	Conector JDBC para bases de datos PostgreSQL



Para el desarrollo y codificación de la aplicación se utilizó la herramienta Eclipse de IBM.

## 5.2 Implementación del prototipo

Para la implementación del prototipo, se crearon cinco paquetes que componen el paquete principal de la aplicación (ver figura 5-1).

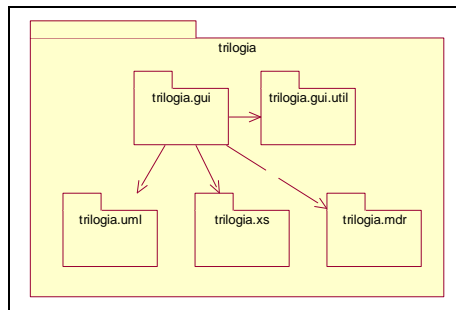


Figura 5-1. Paquetes que conforman la aplicación

El paquete `trilogia.gui` está conformado por cinco clases, las cuales están relacionadas con la interfaz gráfica de la aplicación (ver figura 5-2).

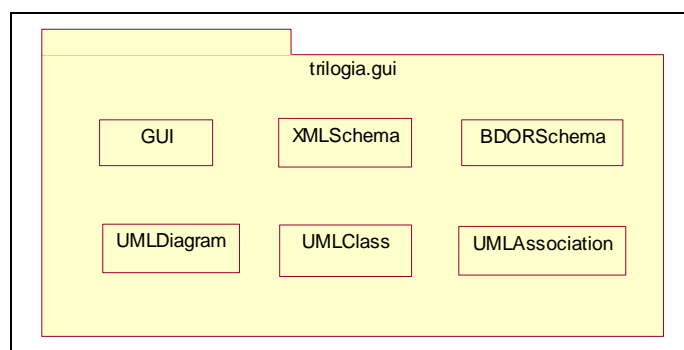


Figura 5-2. Paquete `trilogia.gui`

En la tabla 5-2 se da una breve explicación de las clases que conforman este paquete.



Tabla 5-2. Clases que conforman el paquete *trilogia.gui*

Clase	Descripción
GUI	Es la clase principal que permite la construcción del entorno gráfico de la aplicación. Esta clase es la que maneja todos los eventos que ocurren cuando un usuario realiza una acción sobre la ventana principal de la aplicación.
UMLDiagram	Permite crear una ventana gráfica que representa un diagrama de clases UML dentro de la aplicación. Esta clase recibe todos los eventos relacionados a la creación de diagramas UML y además es un contenedor de objetos UMLClass y UMLAssociation.
UMLClass	Permite crear un objeto gráfico que representa una clase UML dentro del contexto de un diagrama de clases.
UMLAssociation	Permite crear un objeto gráfico que representa una asociación UML dentro del contexto de un diagrama de clases.
BDORSchema	Permite crear una ventana gráfica que contiene un esquema de una base de datos objeto relacional en formato de texto plano.
XMLSchema	Permite crear un objeto que representa un documento esquema XML.

El paquete *trilogia.gui.util* está conformado por seis clases que sirven de soporte para componentes gráficos de la aplicación (ver figura 5-3).

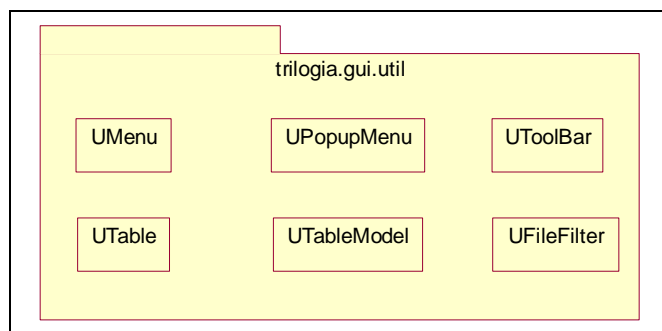


Figura 5-3. Paquete *trilogia.gui.util*

Las clases contenidas en este paquete fueron creadas para optimizar y simplificar la codificación de las clases que permiten construir el entorno gráfico de la aplicación (ver table 5-3).



Tabla 5-3. Clases que conforman el paquete *trilogia.gui.util*

Clase	Descripción
UMenu	Permite crear un componente gráfico para la creación de la barra de menú de la aplicación.
UPopupMenu	Permite crear un componente gráfico para la creación de los menús emergentes de la aplicación.
UToolBar	Permite crear un componente gráfico para la creación de la barra de herramientas de la aplicación.
UTableModel	Permite crear el modelo de tabla, el cual define el conjunto de operaciones para la gestión de los componentes gráficos UTable.
UTable	Permite crear un componente gráfico para la creación de tablas.
UFileFilter	Permite crear un objeto que permite el filtrado de archivos. Este objeto es utilizado por los cuadros de diálogo para abrir y guardar archivos.

El paquete *trilogia.uml* está conformado por seis clases que en su mayoría sirven de soporte para la creación de los diagramas de clase (ver figura 5-4).

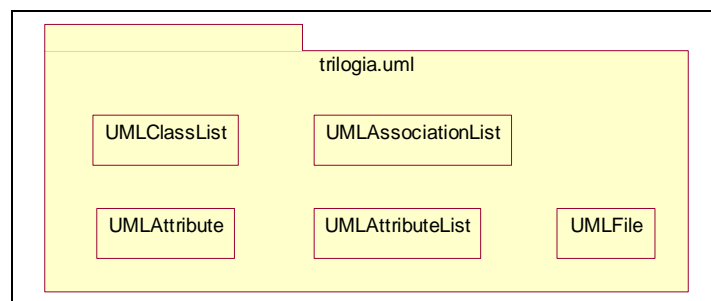


Figura 5-4. Paquete *trilogia.uml*

Estas clases fueron creadas con el fin de simplificar la lógica de negocios asociada a la creación de los diagramas de clase UML (ver tabla 5-4).

Tabla 5-4. Clases que conforman el paquete *trilogia.uml*

Clase	Descripción
UMLClassList	Permite la creación de una colección de objetos <i>UMLClass</i> .
UMLAssociationList	Permite la creación de una colección de objetos <i>UMLAssociation</i> .





UMLAttribute	Representa la definición de un atributo UML.
UMLAttributeList	Permite la creación de una colección de objetos UMLAttribute.
UMLFile	Permite crear un objeto serializable para la creación de archivos que contienen la información de los diagrama de clases UML.

El paquete `trilogia.xs` está conformado por las clases que permiten la creación de documentos esquema XML (ver figura 5-5).

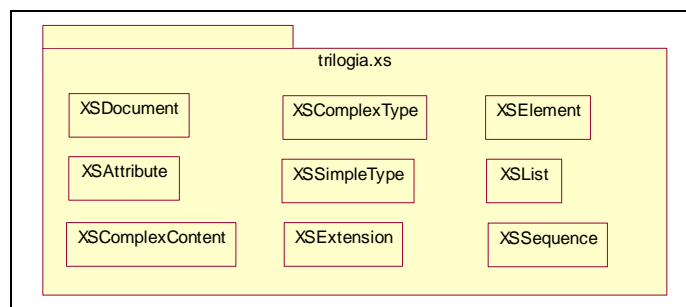


Figura 5-5. Paquete `trilogia.xs`

Para la construcción de estas clases se tomó como referencia la especificación Esquema XML, la cual define las propiedades de cada componente que es usado en la construcción de documentos esquema XML (ver tabla 5-5).

Tabla 5-5. Clases que conforman el paquete `trilogia.xs`

Clase	Descripción
XSDocument	Permite crear un objeto que representa a un documento esquema XML en memoria.
XSComplexType	Representa a una definición de tipo complejo dentro de un documento esquema XML.
XSSimpleType	Representa a una definición de tipo simple dentro de un documento esquema XML.
XSElement	Representa una declaración de elemento dentro de un documento esquema XML.
XSAttribute	Representa la declaración de un atributo dentro de un documento esquema XML.
XSComplexContent	Representa a la definición de tipo complejo dentro de un documento



	esquema XML.
XSExtension	Representa la definición de una extensión de un documento esquema XML.
XSSequence	Representa la definición de una secuencia dentro de un documento esquema XML.
XSLList	Representa la definición de una lista dentro de un documento esquema XML.

El paquete `trilogia.mdr` está conformado por las clases que permiten consultar, organizar y manejar los metadatos provenientes de las bases de datos objeto relacionales (ver figura 5-6).

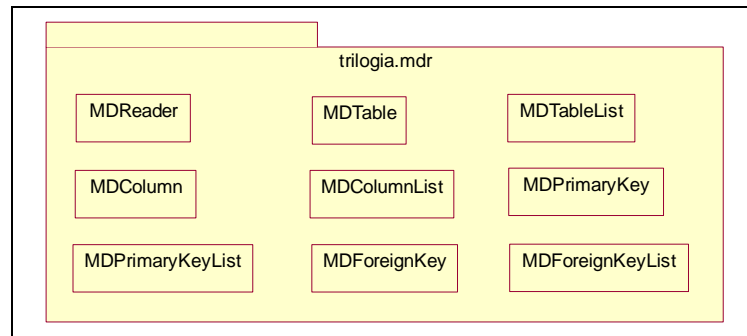


Figura 5-6. Paquete `trilogia.mdr`

En la tabla 5-6 se da una breve explicación de las clases que conforman este paquete.

Tabla 5-6. Clases que conforman el paquete `trilogia.mdr`

Clase	Descripción
<code>MDReader</code>	Esta clase permite consultar los metadatos contenidos en el catálogo de una base de datos objeto relacional y devolverlos como una lista de objetos <code>MDTable</code> .
<code>MDTable</code>	Representa un objeto que contiene los metadatos correspondientes a una tabla de la base de datos. Usado como contenedor de objetos <code>MDColumn</code> , <code>MDPrimaryKey</code> y <code>MDForeignKey</code> .
<code>MDTableList</code>	Permite la creación de una colección de objetos <code>MDTable</code> .
<code>MDColumn</code>	Representa un objeto que contiene los metadatos correspondientes a una columna de una tabla del catálogo.
<code>MDColumnList</code>	Permite la creación de una colección de objetos <code>MDColumn</code> .



MDPrimaryKey	Representa un objeto que contiene los metadatos correspondientes a una clave primaria de una tabla del catálogo.
MDPrimaryKeyList	Permite la creación de una colección de objetos MDPrimaryKey.
MDForeignKey	Representa un objeto que contiene los metadatos correspondientes a una clave foránea de una tabla del catálogo.
MDForeignKeyList	Permite la creación de una colección de objetos MDForeignKey.

### 5.3 Arquitectura de despliegue

La figura 5-7 muestra la arquitectura final de despliegue de la aplicación. La arquitectura está compuesta por dos capas. En la primera capa se encuentra el paquete que contiene la aplicación; en la segunda, el paquete de interfaz de bases de datos objeto relacionales el cual ofrece los mecanismos de conexión con los sistemas de gestión de bases de datos: MySQL, SQLServer, Oracle y PostgreSQL.

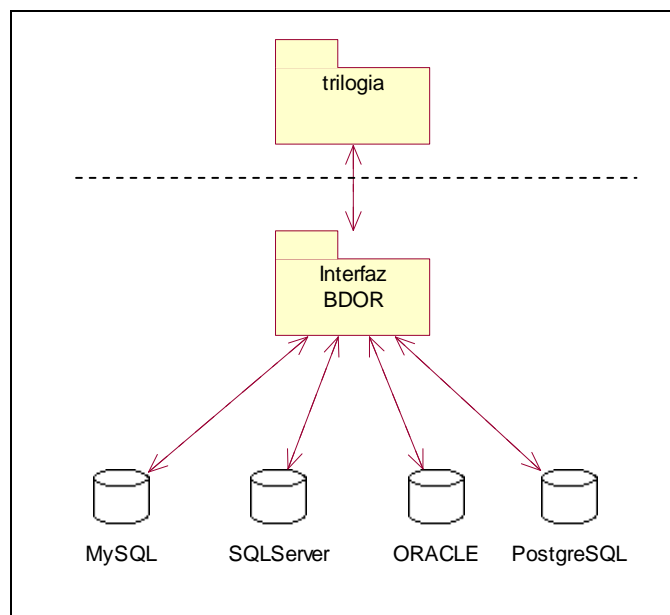


Figura 5-7 Arquitectura de despliegue.



## 5.4 Prototipo

El prototipo implementado lleva por nombre Trilogía, la cual es una herramienta que permite realizar las siguientes funciones:

1. Construir un diagrama de clases UML que permiten modelar la estructura de los documentos esquema XML.
2. Generar un documento esquema XML a partir del diagrama de clases UML.
3. Generar un esquema BDOR a partir del documento esquema generado.
4. Generar un diagrama de clases UML a partir del catálogo de una BDOR.

### 5.4.1 Entorno de la aplicación

La interfaz gráfica de la aplicación fue diseñada aplicando los principios básicos para el diseño de interfaces gráficas de usuarios. La aplicación proporciona un entorno de trabajo amigable y sencillo que facilita el uso a los usuarios (ver figura 5-8).

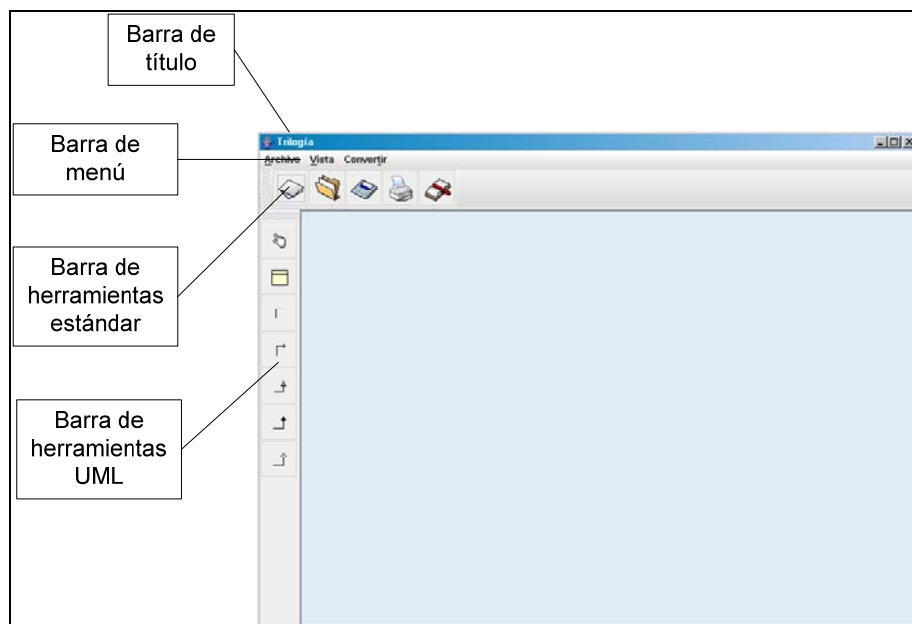


Figura 5-8 Interfaz gráfica de la aplicación.



### Barra de menú

La barra de menú está ubicada debajo de la barra de título y está compuesta por los siguientes menús desplegables:

- *Archivo*: El menú *Archivo* contiene los siguientes ítems:
  - *Nuevo*: Permite crear un nuevo diagrama de clases.
  - *Abrir*: Permite abrir archivos generados por la aplicación que posean extensión `uml` o `sql`, que son archivos de diagramas de clases UML y esquemas BDOR, respectivamente.
  - *Guardar* y *Guardar como*: Los diagramas de clases y los esquemas BDOR creados por la aplicación pueden ser guardados como archivos utilizando estas opciones. La opción *Guardar* puede ser accedida presionando las teclas de acceso rápido `Ctrl-G`.
  - *Imprimir*: Tanto los diagramas de clases como los esquemas BDOR que han sido creados pueden imprimirse en un dispositivo de impresión utilizando esta opción.
  - *Salir*: Esta opción es usada cuando se desea cerrar la aplicación. Se puede hacer uso de las teclas `Ctrl-S` para acceder directamente a esta función.
- *Vista*: Este menú está conformado por dos ítems o submenús:
  - *Construir vista*: Esta función permite describir gráficamente sobre un diagrama de clases un conjunto de recorridos a través de las asociaciones del diagrama. La estructura de los recorridos determinan la estructura que tendrá los documentos esquemas BDOR.
  - *Deshacer vista*: Las vistas que han sido creadas sobre los diagramas de clases UML pueden ser destruidas o anuladas haciendo uso de esta opción.
- *Convertir*: Contiene tres menús que permiten realizar las siguientes funciones:
  - *Crear esquema XML desde diagrama UML*: A partir de los diagramas de clase UML que han sido creados y en los cuales se ha creado una vista, pueden ser transformados en documentos esquema XML. Los documentos generados por la aplicación son guardados como archivos con extensión `xsd`.



- *Crear esquema BDOR desde esquema XML:* Los archivos `xsd` generados por la aplicación pueden ser convertidos en esquemas BDOR haciendo uso de esta opción. Una vez seleccionado el sistema de gestión de bases de datos a utilizar, el sistema presenta una ventana que contiene el esquema BDOR resultante.
- *Crear diagrama UML desde catálogo BDOR:* Se puede especificar el catálogo de una base de datos (creada a partir de un esquema BDOR generado por la aplicación), para generar un diagrama de clases a partir de los metadatos contenidos en el diccionario de datos de la BDOR.

### Barra de herramientas estándar

Esta barra está ubicada debajo de la barra de herramientas y está compuesta por un conjunto de cinco botones, los cuales permiten acceder a las funciones más comunes de la barra de herramientas que son utilizadas por los usuarios. La figura 5-9 describe cada uno de estos botones.

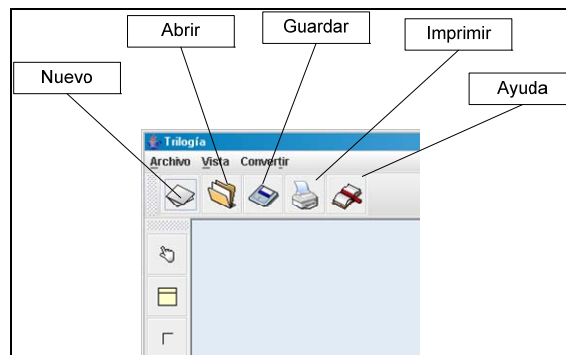


Figura 5-9. Barra de herramientas estándar.

### Barra de herramientas UML

La barra de herramientas UML es la barra vertical ubicada en la parte izquierda de la ventana principal de la aplicación. Esta barra está compuesta por siete botones que permiten la construcción de los diagramas de clases (ver figura 5-10).

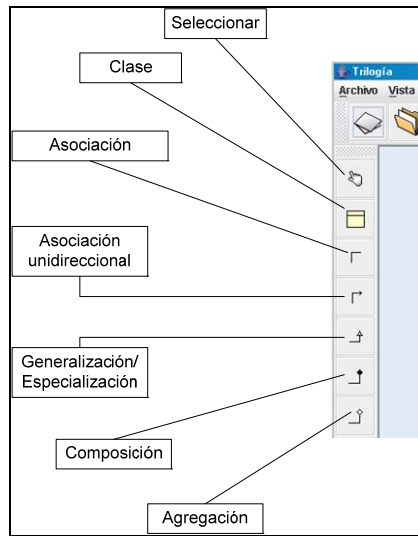



Figura 5-10. Barra de herramientas UML.

### 5.4.2 Construcción de diagramas de clases

Los diagramas de clases UML pueden ser creados de una forma rápida y sencilla haciendo uso de la barra de herramientas UML de la aplicación. Antes de realizar alguna tarea sobre el diagrama de clases es necesario seleccionar una acción, presionando uno de las opciones o botones que aparecen en la barra de herramientas UML.

Para crear un nuevo diagrama de clases, se hace clic en el menú *Archivo* → *Nuevo*. Seguidamente se abre la ventana *NuevoDiagrama1* (ver figura 5-11). Otra opción es presionando el botón *Nuevo*,  ubicado en la barra de herramientas.

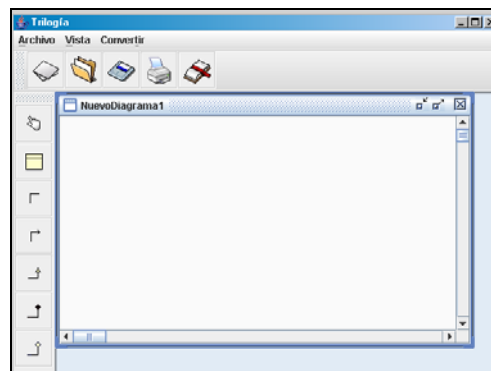



Figura 5-11. Diagrama de clases vacío.



En las siguientes secciones se explica cómo crear clases y asociaciones sobre la ventana *NuevoDiagrama1*.

### Creación, Modificación y Eliminación de una clase

Para crear una clase, se presiona el botón *Clase*,  ubicado en la barra de herramientas UML. Luego se hace clic sobre el diagrama de clases. Seguidamente debe aparecer sobre el diagrama la clase que ha sido creada (ver figura 5-12).

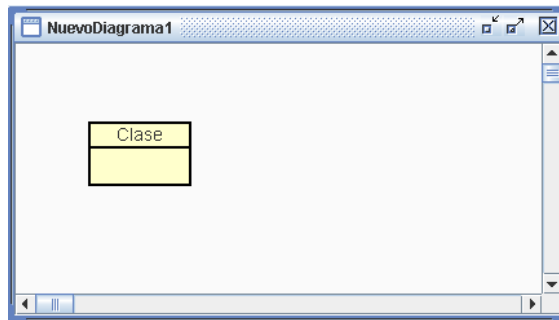


Figura 5-12. Nueva clase.

Una vez que la clase ha sido creada, se pueden modificar sus propiedades. Para esto, se hace doble clic sobre la clase y seguidamente debe aparecer el cuadro de diálogo *Propiedades* de la clase (ver figura 5-13).

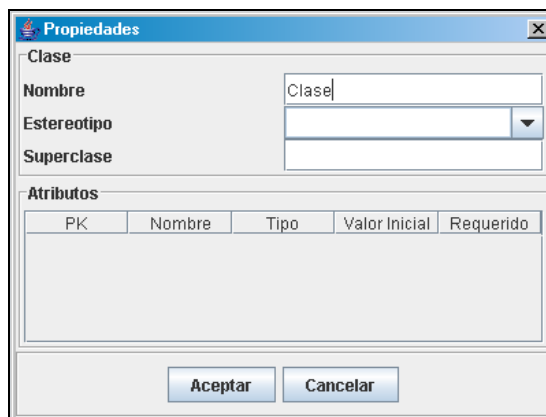


Figura 5-13. Cuadro de diálogo: *Propiedades* de la clase





Para agregar o eliminar un atributo, se hace clic derecho sobre la tabla *Atributos* y debe mostrarse un menú emergente con las opciones. *Ingresar atributo* y *Eliminar atributo*. Los atributos deben tener un nombre y un tipo de dato. Si el atributo representa un identificador o clave, se marca la opción PK. Una vez modificadas las propiedades, se hace clic en el botón *Aceptar*. El resultado de la modificación de las propiedades de la clase debe aparecer reflejada sobre el diagrama de clases (ver figura 5-14).

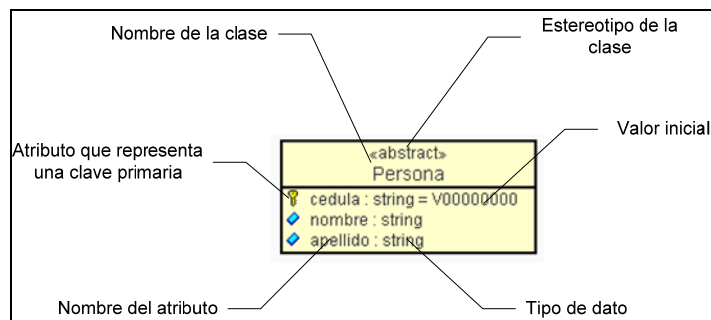


Figura 5-14. Representación gráfica de una clase..

Para eliminar una clase que haya sido creada, se hace clic derecho sobre la clase y seguidamente se muestra un menú emergente con las opciones *Eliminar* y *Propiedades*. Si la clase no posee ninguna asociación con alguna otra clase, al presionar la opción *Eliminar*, la clase es eliminada del diagrama de clases.

### Creación, Modificación y Eliminación de asociaciones

Para crear una asociación entre clases, se debe seleccionar sobre la barra de herramientas el tipo de asociación que se desea crear. Se pueden elegir las siguientes opciones:

- Asociación bidireccional
- Asociación unidireccional
- Generalización/Especialización
- Composición
- Agregación



Una vez seleccionado el tipo de asociación a crear, se hace clic sobre la primera clase y luego sobre la segunda. Seguidamente debe aparecer una línea sobre el diagrama de clases que enlaza las clases (ver figura 5-15).

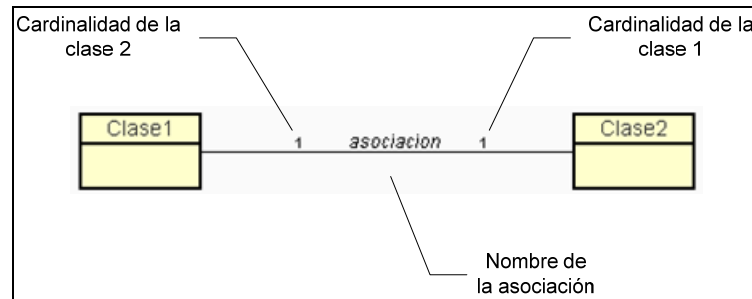


Figura 5-15. Representación gráfica de una asociación.

Existen ciertas restricciones al momento de crear asociaciones entre clases:

- Las asociaciones bidireccionales, unidireccionales, de composición y agregación, solamente pueden ser creadas entre dos clases no abstractas, es decir, que no estén identificadas con el estereotipo «abstract».
- Una relación de generalización/especialización solamente puede crearse entre una clase abstracta y una clase no abstracta.

Para modificar las propiedades de la asociación, se hace doble clic sobre la línea que enlaza las clases. Seguidamente aparece el cuadro de diálogo *Propiedades* de la asociación (ver figura 5-16).

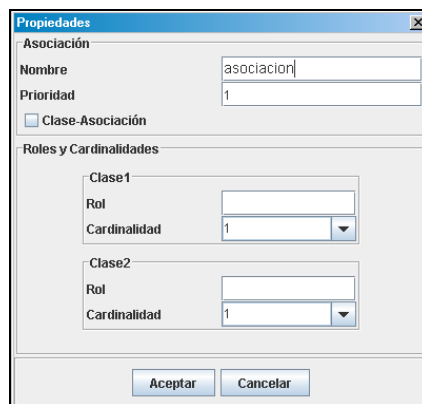


Figura 5-16. Cuadro de diálogo: Propiedades de asociación



En la figura 5-16, la propiedad `prioridad` es un número entero que indica si la asociación debe ser tomada en cuenta antes que otras asociaciones del diagrama al momento de generar la vista sobre el diagrama de clases. Asignar un valor mayor indica una menor prioridad al momento de generar los recorridos de la vista para la creación de los esquemas XML.

Finalmente, para eliminar una asociación, se hace clic derecho sobre la línea que representa la asociación. Seguidamente aparece un menú emergente en el cual se debe seleccionar la opción *Eliminar*.

### 5.4.3 Creación de esquemas XML y esquemas BDOR

Para crear un documento esquema XML a partir de un diagrama de clases existente, inicialmente se debe crear una vista sobre el diagrama de clases. Para ello, primero se selecciona una clase del diagrama UML haciendo clic con el ratón sobre dicha clase y luego se hace clic en el menú *Vista* → *Construir vista*. Seguidamente aparecen resaltadas en color azul las asociaciones del diagrama de clase UML que forman parte de la vista (ver figura 5-17).

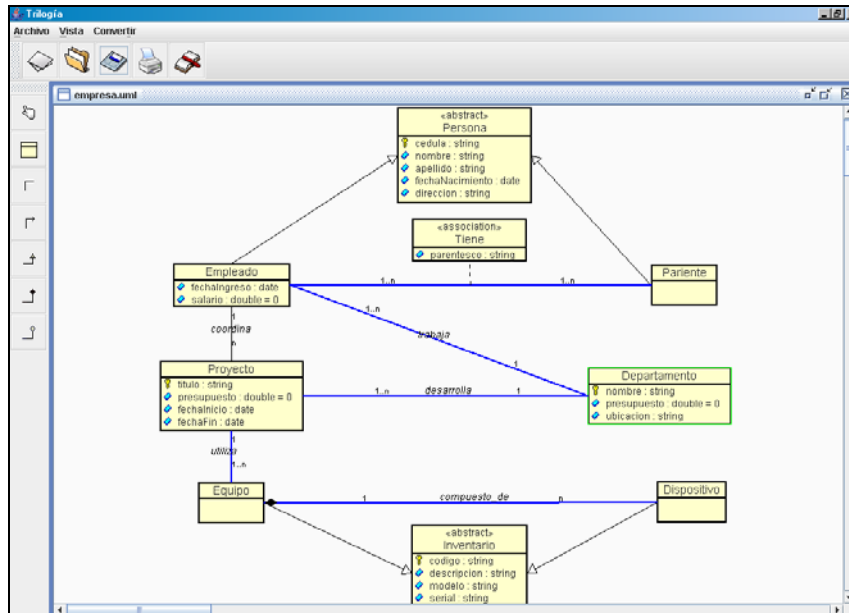
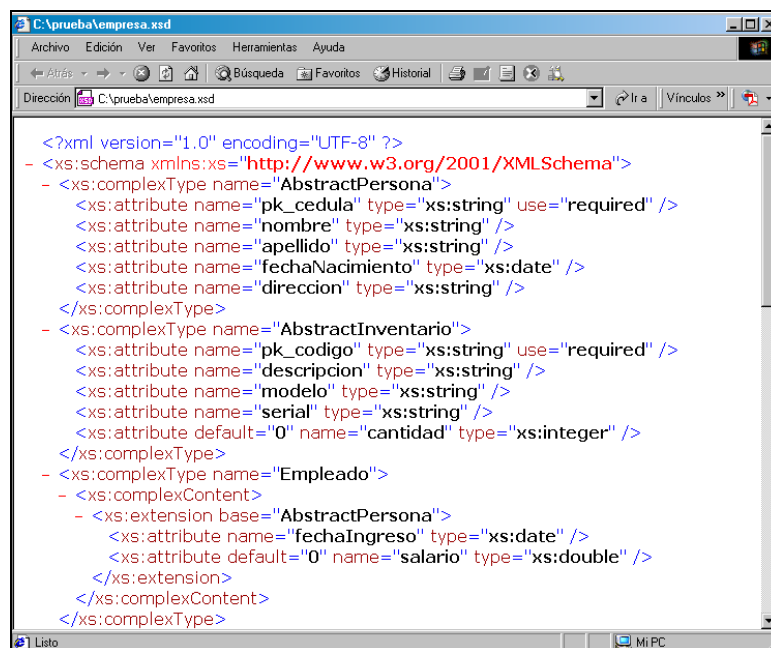


Figura 5-17. Representación de una vista sobre el diagrama de clases.



Si se desea modificar la vista existente, primero se hace clic en el menú *Vista* → *Destruir vista*. Luego se modifican los pesos de las asociaciones, asignando menor peso a las asociaciones que formarán parte de la vista y que serán tomadas en cuenta en la construcción del esquema XML. Finalmente, se crea la vista del mismo modo explicado anteriormente.

Para construir un documento esquema XML a partir del diagrama de clases UML que posee la vista, se hace clic en el menú *Convertir* → *Crear esquema XML desde diagrama UML*. Seguidamente aparece el cuadro de diálogo *Guardar*, en el cual se debe seleccionar la ubicación y asignar el nombre con el que se desea guardar el documento esquema XML. Una vez creado el documento esquema XML, el archivo puede ser abrirse desde cualquier navegador (*browser*) o editor de texto (ver figura 5-18).



```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
- <xs:complexType name="AbstractPersona">
  <xs:attribute name="pk_cedula" type="xs:string" use="required" />
  <xs:attribute name="nombre" type="xs:string" />
  <xs:attribute name="apellido" type="xs:string" />
  <xs:attribute name="fechaNacimiento" type="xs:date" />
  <xs:attribute name="direccion" type="xs:string" />
</xs:complexType>
- <xs:complexType name="AbstractInventario">
  <xs:attribute name="pk_codigo" type="xs:string" use="required" />
  <xs:attribute name="descripcion" type="xs:string" />
  <xs:attribute name="modelo" type="xs:string" />
  <xs:attribute name="serial" type="xs:string" />
  <xs:attribute default="0" name="cantidad" type="xs:integer" />
</xs:complexType>
- <xs:complexType name="Empleado">
- <xs:complexContent>
- <xs:extension base="AbstractPersona">
  <xs:attribute name="fechaIngreso" type="xs:date" />
  <xs:attribute default="0" name="salario" type="xs:double" />
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

Figura 5-18. Documento esquema XML creado por la aplicación.

Para generar un esquema BDOR a partir del esquema XML, se hace clic en el menú *Convertir* → *Crear esquema BDOR desde esquema XML*. Seguidamente se muestra el cuadro de diálogo mostrado en la figura 5-19.

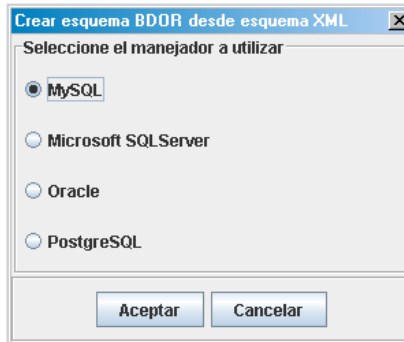


Figura 5-19. Cuadro de diálogo: Crear esquema BDOR desde esquema XML.

En el cuadro de diálogo *Crear esquema BDOR desde esquema XML*, se selecciona el manejador de BDOR a utilizar. Se hace clic en el botón *Aceptar* y en el cuadro de diálogo *Abrir*, se debe seleccionar el archivo esquema XML a convertir. Finalmente aparece la ventana *EsquemaBDOR*, la cual muestra el esquema BDOR que ha sido creado (ver figura 5-20).

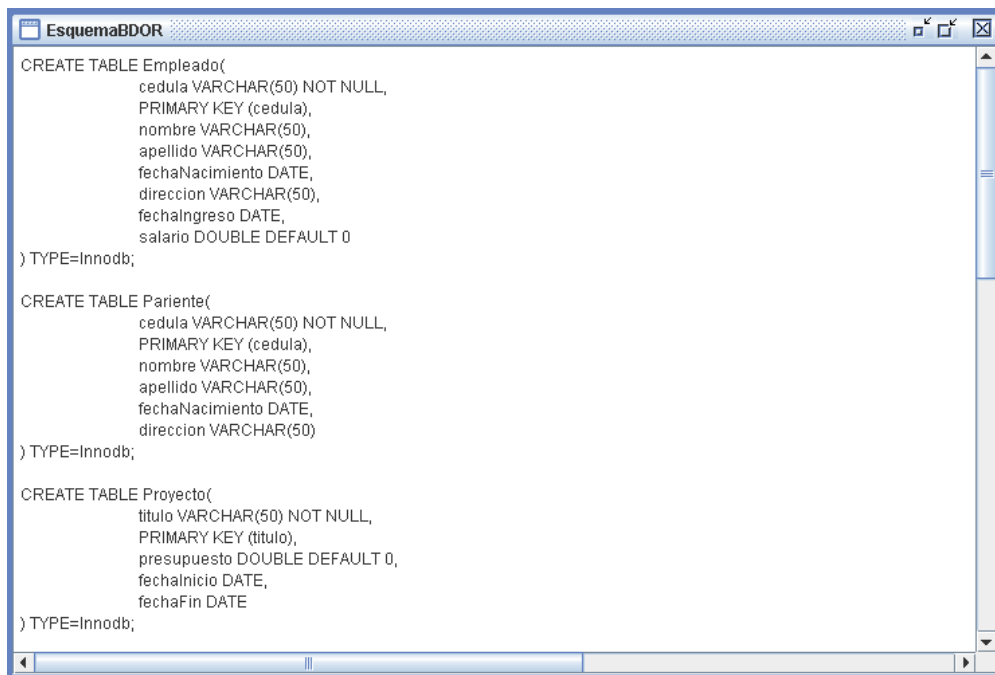


Figura 5-20. Documento esquema BDOR creado por la aplicación.



### 5.4.4 Creación de un diagrama de clases a partir del catálogo de una BDOR

Finalmente, es posible la creación de un diagrama de clases a partir del catálogo que contiene los metadatos de una BDOR. Para efectuar esta operación, se hace clic en el menú *Convertir* → *Crear diagrama UML desde catálogo BDOR*. Seguidamente aparece el cuadro de diálogo mostrado en la figura 5-21.



Figura 5-21. Cuadro de diálogo: Crear diagrama UML desde catálogo BDOR.

En el cuadro de diálogo *Crear diagrama UML desde catálogo UML* se selecciona el sistema de gestión de BDOR a utilizar, se asigna el nombre del servidor donde se encuentra ubicada la base de datos, el nombre de usuario (*login*) y por último la contraseña (*password*), con los cuales se va a acceder a la base de datos. Finalmente aparece la ventana *NuevoDiagrama1* en la cual se muestra el diagrama generado (ver figura 5-22).

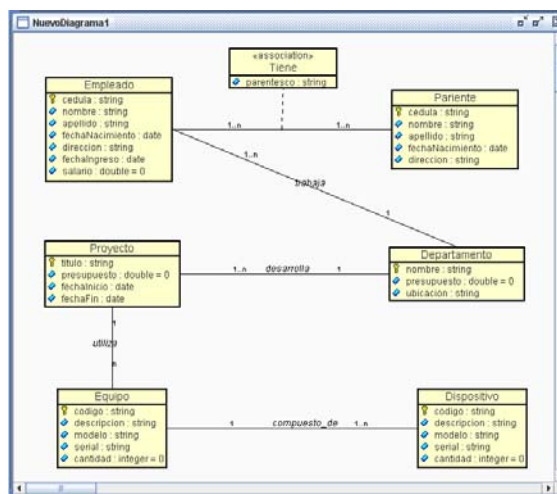


Figura 5-22. Diagrama UML generado a partir del catálogo de la BDOR.





# Capítulo 6. Conclusiones y Recomendaciones

## 6.1 Conclusiones

El rápido crecimiento y evolución de las tecnologías XML han hecho que XML se haya convertido en un lapso muy corto de tiempo en el lenguaje universal por preferencia para el intercambio de datos. La mayoría de los grupos de investigación están enfocando sus esfuerzos en tratar de incorporar XML a las tecnologías de bases de datos objeto-relacionales existentes. Los estudios iniciales solamente estaban orientados a establecer una manera de almacenar una gran cantidad de documentos existentes en repositorios de datos XML, los cuales intentaban suplir las funciones de las bases de datos relacionales. Pero estos repositorios no lograron tener la eficiencia que poseían los sistemas de gestión de bases de datos, los cuales eran sistemas que poseían una gran madurez.

En el capítulo 1 se pudo observar cómo la mayoría de los trabajos estaban orientados en tratar de utilizar las bases de datos relacionales (BDR) para almacenar los datos contenidos en los documentos XML, para posteriormente consultar estos datos y generar nuevamente los documentos XML que se habían almacenado inicialmente. El primer problema surge, ya que al usar las definiciones de tipo de documento (DTDs), éstas no soportaban definiciones de tipos de datos compatibles con las bases de datos. El segundo gran problema aparece cuando se dieron cuenta que tratar de establecer un intercambio bidireccional de datos entre los documentos XML y las BDOR no era posible si no se establecían reglas bien definidas.





Luego que la recomendación Esquema XML sustituye a la antigua DTD, los problemas de incompatibilidad de tipos de datos quedan resueltos. Por esta razón, esta tesis tenía como objetivo principal establecer las bases para lograr un intercambio de datos entre documentos XML y bases de datos objeto-relacionales (BDOR).

Un documento XML válido debe cumplir con las reglas impuestas en el documento esquema XML asociado. La creación de un documento XML puede hacerse de maneras muy distintas, pero si los documentos XML son creados en base a un documento esquema XML que es construido en base a reglas bien definidas, el problema del intercambio de datos entre los documentos XML y las BDOR queda resuelto.

En el capítulo 2 presenta las APIs DOM Nivel 3 usadas para la manipulación de documentos XML. El prototipo final, fue implementado haciendo uso de las implementaciones DOM ofrecidas por el lenguaje Java en su última versión. Solo fueron necesarias un mínimo conjunto de estas interfaces para lograr la construcción de los esquemas XML generados por la aplicación, lo que permite concluir que DOM ofrece a los programadores una de las APIs más importantes para la manipulación y construcción de documentos XML.

La construcción de un esquema XML no es posible si no se estudia con detalle la recomendación Esquema XML, en la cual se especifican los tipos de datos y la estructura de componentes de un esquema XML. En el capítulo 2 se hizo un estudio detallado de esta recomendación, la cual se tomó en cuenta para la generar una estructura válida en los esquemas XML que fueron generados por la aplicación.

Igualmente, los esquemas BDOR que se generaron con la aplicación siguen una sintaxis acorde al estándar SQL-92 presentado al final del capítulo 2. Aunque la mayoría de los sistemas de gestión de bases de datos objeto relacionales (SGBDOR) son compatibles con el estándar SQL-92, fue necesario hacer un estudio comparativo del lenguaje SQL soportado por los manejadores de bases de datos MySQL, SQLServer, PostgreSQL y ORACLE. Este estudio de características fue utilizado para que la aplicación pudiera generar esquemas BDOR compatibles con estos manejadores.

En el capítulo 3 se establecen las normas para la construcción de esquemas XML modelados a partir de diagramas de clases UML. Se define el concepto de vista, que



permite definir la estructura de componentes que tendrá el documento esquema XML. Luego, se establecieron similitudes entre las clases y asociaciones de un diagrama UML y las definiciones de tipos de datos y declaraciones de elementos contenidas en los esquemas XML. Los documentos esquema XML obtenidos, definen una estructura general para los documentos instancia XML que serán utilizados posteriormente para realizar un intercambio bidireccional.

En la segunda parte del capítulo 3, se establecieron comparaciones entre los documentos esquema XML obtenidos de la primera transformación y los esquemas de BDOR, permitiendo de esta manera, la definición de un conjunto de reglas que permiten la transformación de los componentes de un esquema XML en un conjunto de sentencias de creación de tablas en el esquema BDOR. En esta etapa se alcanzó obtener un esquema BDOR derivado de un esquema XML con el cual se puede generar la base de datos para almacenar los datos de los documentos instancia XML.

La parte final del capítulo 3 presenta el conjunto de reglas que permiten crear un diagrama de clases a partir de un esquema BDOR obtenido en la segunda transformación. El conjunto de sentencias de creación de tablas que conforman el esquema BDOR se dividió en dos conjuntos. El primer conjunto, correspondiente a las definiciones de tablas que poseían claves primarias se consiguió transformar en las clases del diagrama UML. El segundo conjunto de sentencias, correspondientes a las definiciones de tablas que poseían claves foráneas fueron transformadas en las asociaciones del diagrama UML. El diagrama resultante no poseía clases abstractas ya que el concepto de herencia no estaba definido en el estándar SQL-92.

Finalmente, los capítulos 4 y 5 se presentaron los resultados obtenidos de aplicar el modelo de procesos Watch para llevar a cabo la implementación del sistema. Cabe destacar que el modelo de procesos Watch solamente sugiere algunos de los diagramas UML que se pueden realizar en cada fase del modelo y por ende, se trató de presentar los diagramas más comunes utilizados en los procesos de desarrollo de aplicaciones. El resultado final fue la implementación de la herramienta llamada Trilogía. Los resultados obtenidos cumplieron con los objetivos propuestos en el capítulo 1, lo que demostró que las reglas presentadas en



el capítulo 3 pueden ser utilizadas como referencias para solventar el problema de intercambio bidireccional de datos entre XML y las BDOR.

## 6.2 Recomendaciones

Las reglas propuestas en esta tesis deben ir siendo ajustadas a los nuevos estándares. El estándar SQL-3 utilizado por ORACLE parece ser completamente compatible con la recomendación Esquema XML. Se recomienda hacer un estudio profundo de la última publicación del estándar SQL-3, el cual permite la creación de tipos que son equivalentes a las definiciones de tipos complejos de los documentos esquema XML. En la figura 6-1 se proponen ejemplos de cómo realizar estas transformaciones.

Tabla 6-1. Ejemplo de transformación de definición de tipo complejo XML en una sentencia de creación de tipo usando el estándar SQL-3 de ORACLE.

Definición de tipo complejo XML	Sentencia SQL
<pre>&lt;xs:complexType name="Departamento"&gt;   &lt;xs:attribute name="pk_nombre" type="xs:string" use="required"/&gt;   &lt;xs:attribute name="presupuesto" type="xs:double" default="0"/&gt;   &lt;xs:attribute name="ubicacion" type="xs:string"/&gt; &lt;/xs:complexType&gt;</pre>	<pre><b>create type</b> Departamento(   nombre <b>varchar</b>(255) <b>not null</b>,   <b>primary key</b>(nombre),   presupuesto <b>double default</b> 0,   ubicación <b>varchar</b>(255) )</pre>
<pre>&lt;xs:complexType name="AbstractPersona"&gt;   &lt;xs:attribute name="pk_cedula" type="xs:string" use="required"/&gt;   &lt;xs:attribute name="nombre" type="xs:string"/&gt;   &lt;xs:attribute name="apellido" type="xs:string"/&gt;   &lt;xs:attribute name="fechaNacimiento" type="xs:date"/&gt;   &lt;xs:attribute name="direccion" type="xs:string"/&gt; &lt;/xs:complexType&gt;</pre>	<pre><b>create type</b> AbstractPersona(   cedula <b>varchar</b>(255) <b>not null</b>,   <b>primary key</b>(cedula),   nombre <b>varchar</b>(255),   apellido <b>varchar</b>(255),   fechaNacimiento <b>date</b>,   direccion <b>varchar</b>(255), )</pre>



<pre>&lt;xs:complexType name="Empleado"&gt;   &lt;xs:complexContent&gt;     &lt;xs:extension base="AbstractPersona"&gt;       &lt;xs:attribute name="fechaIngreso" type="xs:date"/&gt;       &lt;xs:attribute name="salario" type="xs:double" default="0"/&gt;     &lt;/xs:extension&gt;   &lt;/xs:complexContent&gt; &lt;/xs:complexType&gt;</pre>	<pre><b>create type</b> Empleado <b>under</b> AbstractPersona(   fechaIngreso <b>date</b>,   salario <b>double default 0</b>, )</pre>
---	---

También se propone tratar se resolver el problema de representar los tipos estructurados, ya que la recomendación Esquema XML solo permite declarar atributos de tipo simple. Se propone el uso de IDREFS que permitan hacer referencia al elemento que representa el tipo estructurado.

Finalmente, con respecto a la aplicación Trilogía se recomienda lo siguiente:

- Permitir al usuario ver el contenido de los documentos esquema XML. Para esto es necesario codificar un método que permita organizar la estructura de los elementos con sus respectivas tabulaciones.
- Implementar las características de Edición, que permita a los usuarios realizar las operaciones de seleccionar, copiar y pegar los distintos elementos que aparecen en los diagramas de clases.
- Permitir a los usuarios generar los documentos instancias, ya sea partir del diagrama de clases o bien de los datos contenidos en la base de datos.
- Incorporar la función que permita realizar un intercambio de datos entre los documentos instancia XML creados y las BDOR.
- Soporte para crear las hojas de estilo de los documentos instancia XML.





## Bibliografía

[**Arciniegas 2001**] F. Arciniegas, «Programación Avanzada con XML», Copyright © 2001 by The McGraw-Gill Companies, Inc.

[**Bergholz 2000**] A. Bergholz, «Extending your markup: An XML tutorial», Julio – Agosto, 2000.

[**Binstock et al. 2002**] C. Binstock, D. Peterson y otros, «XML Schema Complete Reference», Addison Wesley, 2002.

[**Bourret 1999**] R. Bourret, «Mapping DTDs to Databases», Mayo 09, 2001.

[**Champion 2001**] M. Champion, «Storing XML in DataBases», 2001.

[**Gicqueau**] A. Gicqueau, «Importing XML documents to Relational Databases using Java».

[**Groff y Weinberg 1999**] J. Groff, P. Weinberg, «SQL: The Complete Reference», Osborne/McGraw Hill, 1999.

[**Guardalben 2002**] G. Guardalben, «Integrating XML and Relational Database Technologies: A Position Paper», 2002.



[**Houlette 2003**] F. Houlette, «Fundamentos de SQL», McGraw Hill, 2003.

[**Montilva et al. 2000**] J. Montilva, K. Hamzan, M. Gharawi, «The Watch Model for Developing Business Software in Small and Midsize Organizations», Proceedings of the IV World Multiconference on Systemics, Cybernetics and Informatics. SCI'2000. Orlando, USA. Jul, 2000.

[**Muller 1997**] Pierre-Alain Muller, «Modelado de Objetos con UML», Primera Edición, Copyright © 1997 por Editions Enrolles.

[**Silberschatz et al. 2002**] A. Silberschatz, H. F. Korth, S. Sudarshan, «Fundamentos de Bases de Datos», Tercera edición, Copyright © 1997 by McGraw-Hill Companies.

[**Turau 1999**] V. Turau, «Making Legacy data accesible for XML applications», Mayo 1999.

[**Van der Vlist 2002**] E. Van der Vlist, «XML Schema». O'Reilly 2002.

[**Williams et al. 2000**] K. Williams, M. Brundage, P. Dengler, J. Gabriel, A. Hoskinson, M. Kay, T. Maxwell, M. Ochoa, J. Papa, M. Vanmane, «Professional XML DataBases», Copyright © 2000 by Wrox Press Ltd.

### Referencias electrónicas

[1] <http://www.w3.org/XML/>

[2] <http://www.w3.org/TR/REC-xml>

[3] <http://www.rpbouret.com/xml/XMLAndDatabases.htm>



- [4] <http://www-106.ibm.com/developerworks/xml/library/x-struct/>
  
- [5] <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407>
  
- [6] <http://www.w3.org/2003/06/Process-20030618/activities.html>
  
- [7] <http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407>
  
- [8] <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>
  
- [9] <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>
  
- [10] <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>
  
- [11] <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407>
  
- [12] <http://www.w3.org/2003/06/Process-20030618/activities.html>
  
- [13] <http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407>
  
- [14] [http://www.sei.cmu.edu/str/descriptions/twotier\\_body.html](http://www.sei.cmu.edu/str/descriptions/twotier_body.html)







## A. Correspondencia entre los diagramas ERE y los diagramas de clases UML

Las características de los diagramas de clases UML y sus correspondencias con los diagramas ERE se muestran en la figura A-1.

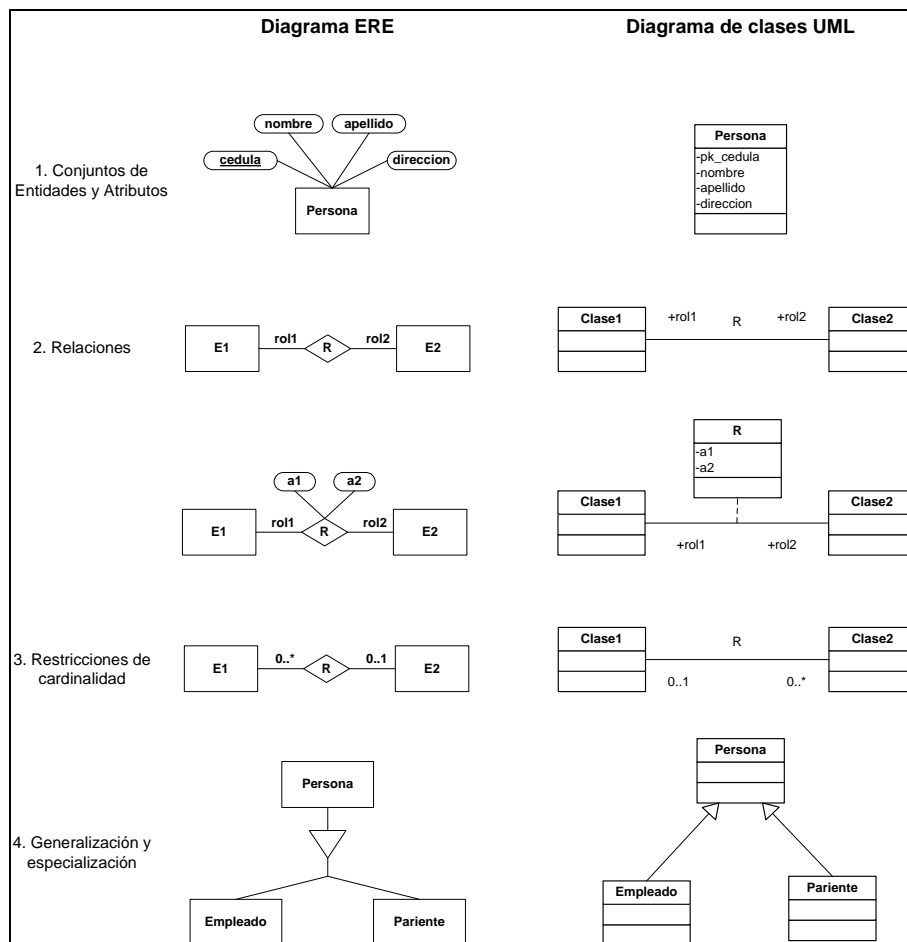


Figura A-1. Analogías entre los diagramas ERE y los diagramas de clase UML.



---

A. *Correspondencia entre los diagramas ERE y los diagramas de clases UML-----A-1*

*Figura A-1. Analogías entre los diagramas ERE y los diagramas de clase UML. ----- A-1*