INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN MECATRÓNICA Y TECNOLOGÍAS DE INFORMACIÓN

DEFINICIÓN DE UN MODELO PARA EL ALMACENAMIENTO DE DATOS SEMIESTRUCTURADOS

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADÉMICO DE:

MAESTRÍA EN CIENCIAS EN TECNOLOGÍA INFORMÁTICA

POR:

ARIOSTO GAONA PLIEGO

MONTERREY, N.L. DICIEMBRE 2008

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

DIVISIÓN DE MECATRÓNICA Y TECNOLOGÍAS DE INFORMACIÓN

PROGRAMAS DE GRADUADOS EN MECATRÓNICA Y TECNOLOGÍAS DE INFORMACIÓN

Los miembros del comité de tesis recomendamos que la presente tesis del Ing. Ariosto Gaona Pliego sea aceptada como requisito parcial para obtener el grado académico de Maestro en Ciencias en Tecnología Informática.

Comité de tesis:

_____

Lorena Guadalupe Gómez Martínez, Ph.D.
Asesor

_____

Juan Carlos Lavariega Jarquín, Ph.D.
Sinodal

_____

Pablo J. Tejeda Zerón, MC.
Sinodal

_____

Dr. Joaquín Acevedo Mascarúa
Director de Investigación y Posgrado
Escuela de Ingeniería
Diciembre 2008

DEFINICIÓN DE UN MODELO PARA EL ALMACENAMIENTO DE DATOS SEMIESTRUCTURADOS

POR:

ARIOSTO GAONA PLIEGO

TESIS

Presentada al Programa de Graduados en Mecatrónica y Tecnologías de Información.

Este trabajo es requisito parcial para obtener el grado de Maestro en Ciencias en Tecnología Informática

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

DICIEMBRE 2008

Dedicatoria

A Dios por su amor y cuidados a lo largo de mi vida;

A mi madre por sus consejos y cariño; y

A mi padre por su constante apoyo y ánimo.

Agradecimientos

Agradezco a mi familia por sus oraciones, apoyo y ánimos a lo largo de estos estudios de postgrado;

A la Dra. Lorena Guadalupe Gómez Martínez, asesora de esta tesis, por su valiosa ayuda y guía durante la realización de esta investigación;

Al Dr. Juan Carlos Lavariega Jarquín por el interés y apoyo que me brindó durante la realización de esta tesis; y

A todos mis maestros y compañeros que a lo largo de mis estudios de maestría me brindaron su apoyo.

¡Muchas Gracias!

Resumen

El flujo de información disponible a través de la Internet ha crecido de manera exponencial, sin embargo, el poder aprovechar esta información es difícil ya que se encuentra el texto en forma libre y al hacer una búsqueda se regresa un número muy grande de documentos que satisfacen los criterios requeridos, pero que no necesariamente representan lo que se busca.
Cualquier información en formato libre se puede representar de una manera semistructurada, utilizando etiquetas que permitan hacer búsquedas más precisas de acuerdo a lo que se necesita encontrar. Para el uso y manejo de información semiestructurada es necesario la ayuda de herramientas especiales, enfocadas en el manejo de este tipo de información, como lo es XML (Extensible Markup Language) [7] que es un lenguaje que permite estructurar la información y asignarle un significado semántico. Pero XML por sí solo no es suficiente, es necesario definir las reglas para transformar un documento sin organización a uno que si la tenga.

En esta investigación se desarrolló una metodología que define un conjunto de pasos para tomar información semiestructurada y organizarla, con el fin de poder almacenarla de forma fácil y eficiente con el objetivo de poder mantenerla, por ejemplo mediante actualizaciones, y sobre todo poder usarla a través de consultas, ya sea obteniendo solo una fracción de la información o incluso todo el documento sin que éste haya perdido detalles como su organización original.
Como resultado de la aplicación de la metodología propuesta se obtienen: un modelo de datos jerárquico y flexible que permite almacenar información semiestructurada, y también se obtienen, las estructuras que faciliten la representación de dicha información. A diferencia de propuestas existentes, como Object Exchange Model [14] ó Entidad Relación para datos semiestructurados [11], el modelo resultante se crea en base al análisis de las características de la información a manejar, mediante la metodología propuesta, y define también, las estructuras de representación particulares.

En esta investigación se desarrolló un modelo de datos semiestructurados aplicado a contenidos curriculares de computación e informática usando la metodología propuesta. La validación del modelo resultante se llevó a cabo mediante su implementación en un sistema computacional que interactúa con una base de datos cuyo diseño fue creado a partir del modelo de datos creado. Así miso, el sistema computacional interactúa con la información semiestructurada almacenada en la base de datos, a través de las estructuras definidas para la interpretación de la información, logrando con esto: almacenar, manipular y consultar información semiestructurada. Definir modelos de datos semiestructurados para otros campos se puede llevar a cabo gracias a la metodología propuesta la cual permite, mediante el análisis de las características de la información a manejar, crear modelos adaptados a la información a almacenar y manipular.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Ejemplos

Capítulo 1

Introducción

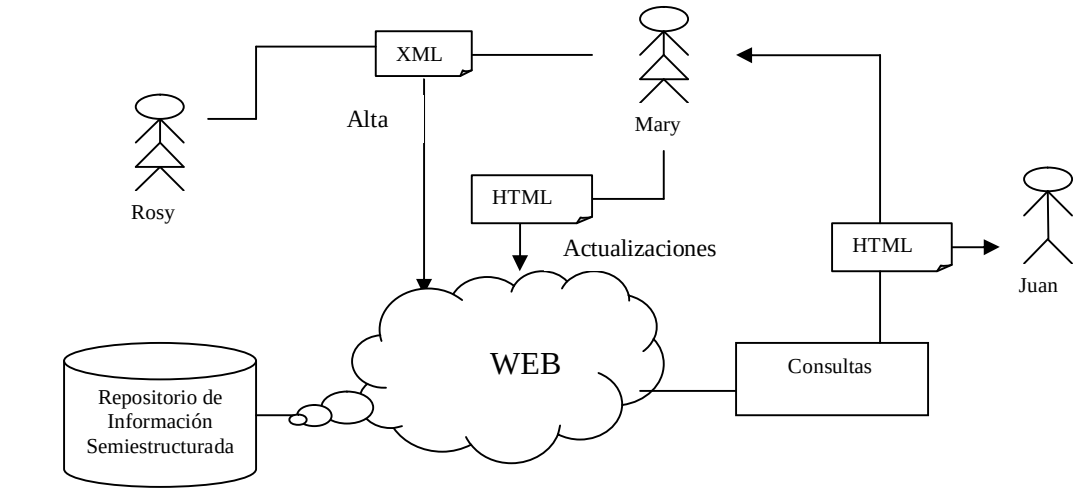
1.1 Motivación

A lo largo de la historia humana toda la información generada a partir los descubrimientos de las diferentes ramas de la ciencia era preservada, hasta mediados del siglo XX, en libros atesorados en bibliotecas. Desde la invención de la computadora, la forma de guardar información cambió radicalmente gracias a herramientas como las Bases de Datos, en las cuales se organiza la información, para almacenarla en la computadora, bajo cierta estructura definida por el diseñador de la Base de Datos, esto da como resultado mejoras en el uso y manejo de la información. También irrumpió en el escenario de la computación el Internet, que es una red mundial que conecta a muchas computadoras a través de todo el mundo y que permite acceder a la información que está almacenada en las Bases de Datos de dichas computadoras.

Actualmente una gran cantidad de datos están disponibles a través de Internet, estos datos pueden contar con una estructura definida en ellos que rija su organización, o por el contrario, no tener ninguna estructura definida, dichos datos son llamados semiestructurados [8].

Almacenar datos semiestructurados provenientes de diversas fuentes, poder realizar búsquedas eficientes sobre la información sin que ésta haya perdido detalles al momento de almacenarla, poder actualizarla al editarla y agregar o eliminar elementos sin la necesidad de dar de baja todo un documento completo, es la principal motivación para el desarrollo de esta tesis, ver figura 1.1.

El reto es como manipular tales datos semiestructurados. Trabajos previos en datos semiestructurados se han enfocado principalmente en el desarrollo de lenguajes de consulta y sistemas tales como OEM-QL [14], HOSQL [16], Tsimmis [15], Pegasus [16] y STRUDEL [17]. Como manipular datos semiestructurados provenientes de diversas fuentes ha recibido también atención, pero propuestas existentes como: OEM [14] y modelos basados en árboles como Trees with Grouping Facets [1] fallan en el adecuado manejo de información semiestructurada [4].



Operaciones:

- Dar de alta información descrita en documentos XML (Extensible Markup Language) [40]
- Realizar actualizaciones y consultas a través de Internet, sobre la información almacenada en el repositorio de información semiestructurada.

Figura 1.1: Operaciones sobre información semiestructurada

En esta tesis se planteó, a través de la definición de un modelo de datos semiestructurados, la solución para el uso eficiente de éste tipo de información, y que se cumpliera completamente con las operaciones descritas en la figura 1.1.

1.2 Motivación Práctica

El proyecto BECCA (Base Electrónica de Contenidos Curriculares para Acreditación en Educación Superior) [20], es un proyecto que propone desarrollar la infraestructura necesaria para la creación y mantenimiento de contenidos curriculares del campo de computación e informática disponibles a través de Internet. El objetivo de BECCA es crear una base educativa que ayude a la gestión de contenidos curriculares, y la cual se organizará en áreas temáticas. De ésta forma se facilitará la adecuación, extracción y uso de los datos al momento de crear los materiales educativos para éste campo.

El modelo de datos semiestructurados desarrollado en esta tesis, será el modelo que BECCA utilice para almacenar la información semiestructurada que maneje, contenidos curriculares del campo de computación e informática [20].

1.3 Objetivos de la Investigación

Los objetivos identificados para esta tesis fueron:

- Definición de estructuras capaces de representar y obtener información semiestructurada en XML.
- Un modelo de datos semiestructurado para almacenar la información del tópico de interés.
- Almacenar la información semiestructurada a partir de documentos XML que describen la información del tópico de interés.
- Realizar operaciones de consulta sobre la información almacenada, así como reconstruirla sin perder el orden que presentaban en el documento original que la describía.
- Permitir la actualización de la información semiestructurada, sin alterar elementos o secciones en la información que no se cambiaron.
- Realizar un prototipo como prueba de concepto del modelo definido aplicado al campo de contenidos curriculares.

1.4 Resultados

Como trabajo de esta tesis, se desarrolló una metodología compuesta por una serie de etapas cuyo principal fin fue definir un modelo de almacenamiento de información semiestructurada. El modelo de datos semiestructurado resultante tiene como características:

- La clasificación, de acuerdo a su estructura y características, de los elementos de la información que se puede presentar durante el almacenamiento, lo que facilita la integración de la información almacenada.
- La definición de estructuras para la representación de la información usadas al momento de extraer de la fuente y almacenarla en el modelo creado. Las estructuras permiten obtener la información lo más completa posible de la fuente y facilitan la reconstrucción de la información ya almacenada.

La aplicación de la metodología desarrollada en esta tesis permitió comprobar que el modelo de almacenamiento semiestructurado obtenido para los contenidos curriculares cumple con los objetivos definidos para esta investigación.

1.5 Organización del Documento de Tesis

El resto del documento de tesis tiene la siguiente organización:

En el Capítulo 2, ñAntecedentesò, se describen conceptos relacionados al manejo de datos semiestructurados. Los conceptos descritos en este Capítulo son: HTML (Hypertext Markup Language), XML, modelos de datos semiestructurados existentes, sus

características y ejemplos. También se incluye la descripción de los contenidos curriculares, el funcionamiento y requerimientos de BECCA y la conclusión de la necesidad de definir un nuevo modelo de datos semiestructurado.

El Capítulo 3, ñDesarrollo del Modelo de Datosò, presenta la metodología utilizada para definir el modelo de datos semiestructurados, aplicado a contenidos curriculares de computación e informática, así como los elementos obtenidos de su aplicación.

El Capítulo 4, ñImplementaciónò, describe los pasos realizados para la implementación del modelo de datos semiestructurado desarrollado en el Capítulo 3, que en esta tesis se aplicó a contenidos curriculares.

El Capítulo 5, ñConclusionesò, contiene las aportaciones de este trabajo, los principales retos que se resolvieron y posibles áreas de extensión de este trabajo de investigación.

Capítulo 2

Antecedentes

Este Capítulo presenta conceptos relacionados al manejo de información semiestructurada. La sección 2.1 presenta XML (Extensible Markup Language) [7] y su ventaja sobre HTML (Hypertext Markup Language) en el manejo de información semiestructurada. En la sección 2.2 se define el concepto de modelos de datos semiestructurados. La sección 2.3 se presentan los modelos de datos semiestructurados existentes. La sección 2.4 enlista sistemas existentes para el manejo de datos semiestructurados. En la sección 2.5 se presenta el funcionamiento y requerimientos de BECCA (Base Electrónica de Contenidos Curriculares para Acreditación en Educación Superior) [20], así como las características de los contenidos curriculares. La sección 2.6 muestra el análisis de la aplicación de los modelos existentes en el manejo de los contenidos curriculares. En la sección 2.7 se destaca la necesidad de definir un nuevo modelo para el almacenamiento de datos semiestructurados. Finalmente el Capítulo concluye con un resumen presentado en la sección 2.8

2.1 XML

El Internet es actualmente el medio preferido para el intercambio de información, su éxito radica en la aceptación de HTTP (Hypertext Transfer Protocol) como un estándar y del éxito de HTML (Hypertext Markup Language) como la herramienta para estructurar texto para presentaciones visuales [5]. Debido a que HTML fue diseñado para describir la presentación de los datos y no el contenido, se debe considerar otro lenguaje que facilite el manejo de datos semiestructurados, es decir, que se enfoque más a la organización de la información, XML (Extensible Markup Language) [7] es la herramienta que cumple con tal propósito, debido a que:

- XML provee facilidades para poder estructurar información y asociarle un significado semántico [12], ésta característica permitirá usar la información de un documento XML y procesarla como se desee.
- Además, XML es considerado actualmente como un estándar para la representación y el intercambio de datos, esto debido a los beneficios que XML brinda para el manejo de los datos, como por ejemplo, proveer de un camino natural para la estructuración de datos, basado en jerarquías o representaciones basadas en grafos [3].

Los datos semiestructurados son datos que no tienen una organización o estructura estricta pero si parcial o que puede ser irregular [2]. El uso de XML facilitará el manejo de los datos semiestructurados, ya que XML permite, por ejemplo, el uso de *tags* que pueden ser definidos a voluntad, además de que permite anidar estructuras hasta un fondo arbitrario [12].

2.2 Los Modelos de Datos Semiestructurados

La información es uno de los activos más importantes para cualquier entidad, por lo tanto su eficiente y correcto manejo es imprescindible para el buen desempeño de cualquier institución.

En la Actualidad ha habido un incremento en la cantidad de información que transita en la Web, al igual que las personas que buscan información a través de ella [6]. Los datos que circulan a través de Internet en ciertas ocasiones pueden tener una estructura, es decir, contar con una organización en su información, pero en otras ocasiones, la información no tendrá ninguna estructura definida, si bien esto es cierto, también lo es que la mayoría de la información estará en medio de éstas dos posibilidades, por lo que se consideran datos semiestructurados [8].

El definir un modelo de datos especial para el manejo de la información semiestructurada tiene como objetivo hacer mejor uso, mediante operaciones como las consultas, de información proveniente de diversas fuentes, como Internet, que no comparten una estructura única en común y que debe ser almacenada en una Base de Datos.

2.2.1 Tipos de Modelos de Datos Semiestructurados

Existen diferentes modelos para el almacenamiento de datos semiestructurados; en un extremo hay modelos complejos y en el otro extremo modelos de forma libre.

2.2.1.1 Modelos Complejos

Los modelos complejos [21] definen todos los diferentes elementos que la información semiestructurada a manejar va a contener, si información nueva se presenta, antes de poder ser manejada, debe de obtenerse su estructura para agregarla al modelo complejo, de otra forma, esta nueva información se perderá.

La figura 2.1, muestra un ejemplo gráfico del funcionamiento y desventaja del modelo complejo, su explicación es la siguiente: el *objeto 3,* dentro de la figura 2.1, representa un modelo complejo, el cual tiene definido todos los elementos que espera de la información que manejara. El *objeto 1* y *objeto 2*, representan entradas de Información hacia el modelo complejo, tanto el conjunto de información X, del *objeto 1,* como el conjunto de información Y del *objeto 2*, contienen elementos que fueron definidos en el modelo de datos complejos, por lo cual pueden ser almacenados exitosamente, pero, si un conjunto de información Z se presenta, *objeto 4*, y contiene elementos que no fueron definidos en el modelo complejo, para este caso elementos F y G, esta información, dentro del modelo complejo se perderá.

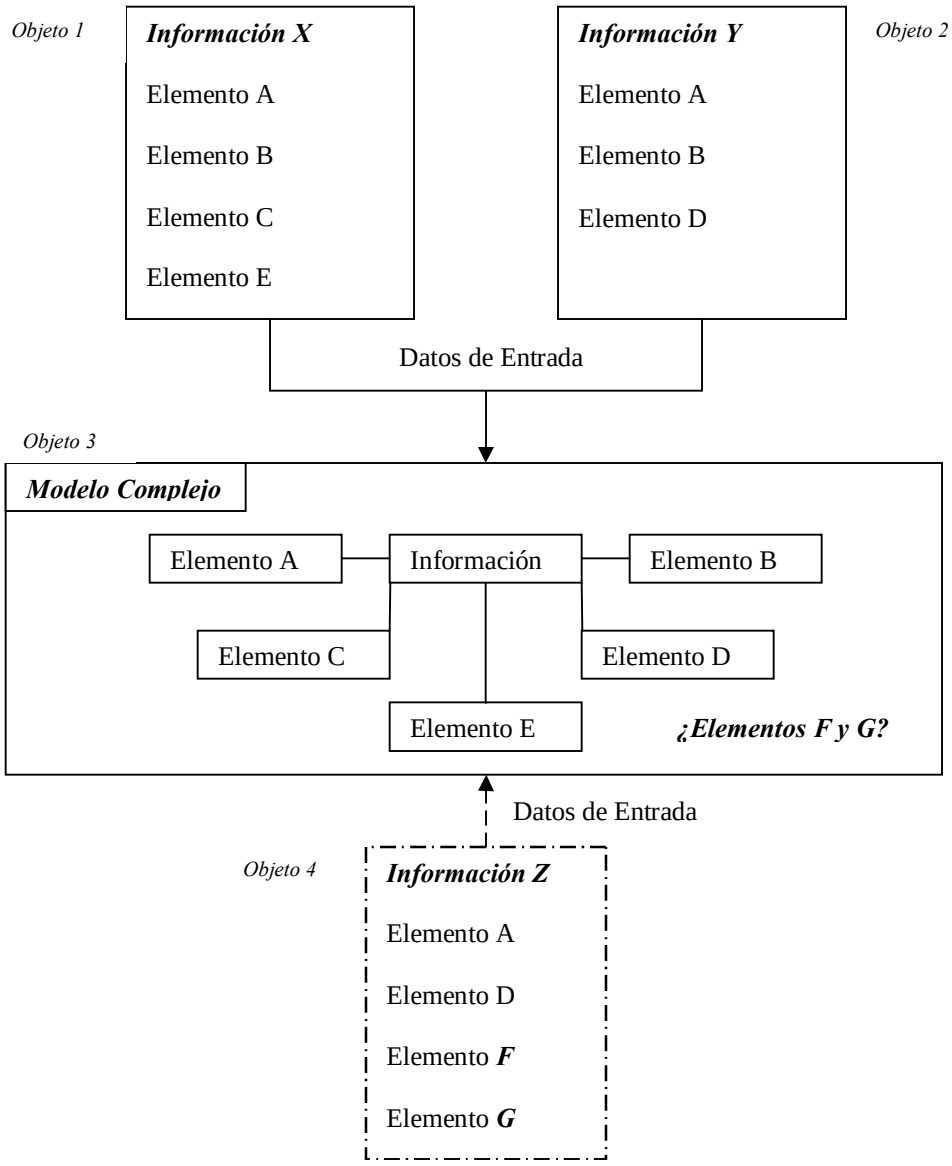


Figura 2.1: Modelo Semiestructurado Complejo

Al predeterminar los tipos de objetos y sus estructuras, un modelo complejo se vuelve complicado de manejar por la gran diversidad de elementos con los que cuenta. Por ejemplo, algunas interfaces requerirán campos adicionales o menos campos para un objeto de cierto tipo [21].

2.2.1.2 Modelos Free-Form

La estructura de los documentos, y la información semiestructurada en sí, tiene forma libre (free - form); en un modelo complejo, aunque se prevean todos los tipos de objetos que puedan llegar a presentarse, siempre existirá la posibilidad de que ninguno de los elementos predefinidos satisfaga las necesidades de algún otro objeto que se presente.

El modelo forma libre [21], en contraste con el modelo complejo, plantea la definición de una estructura que almacene cualquier tipo de información que pueda presentarse, eliminando así el problema de definir diferentes estructuras para las diferentes clases de objetos que pueden venir dentro de la información semiestructurada.

El problema, al usar un modelo forma libre es: la interpretación de la información en la estructura puede resultar en interpretaciones complejas y que no satisfacen plenamente el adecuado almacenamiento de la información, derivando en consultas que arrojan información irrelevante o errónea.

2.3 Modelos de Datos Semiestructurados Existentes

Dado que los modelos complejos se basan en un solo principio, el definir los diferentes tipos de elementos que se pueden presentar para ser agregados al modelo complejo, ver figura 2.1, en esta sección sólo se presentan modelos free ï form, ya que las diferentes opciones existentes, para este tipo de modelos, sí varían de unas a otras en sus propuestas para el manejo de información semiestructurada, en contraste con los modelos complejos que siguen un único principio.

Los ejemplos de modelos de datos semiestructurados estudiados son: Object Exchange Model [14], árbol con agrupación de facetas [1], entidad relación para datos semiestructurados [11] y el modelo de datos HyperFile [21]. Las características de cada uno de los modelos presentados varían de unos a otros como se puede observarse a continuación:

2.3.1 Modelo de datos OEM (Object Exchange Model)

El Object Exchange Model (OEM) [14], es un modelo simple tipo free form para la representación de datos semiestructurados y datos que pueden ser irregulares o incompletos. Este modelo OEM se enfoca principalmente en el problema de intercambio de información que los diferentes elementos de un sistema realizan. Para el modelo OEM, cada objeto debe cumplir con la siguiente estructura, mostrada en la Tabla 2.1:

Etiqueta	Tipo	Valor	ID Objeto

Tabla 2.1. Estructura del modelo OEM.

- Etiqueta: Que es una variable de caracteres String que describe al objeto al que representa.
- Tipo: El tipo de datos que es el objeto, por ejemplo: entero, string, etc.
- Valor: El valor para el objeto.
- ID del Objeto: Un variable de identificación única para el objeto

Ejemplo: En la figura 2.2 se muestra la representación de una sección de un contenido curricular perteneciente al campo de computación e informática, cuyo contenido se divide en área, subárea, subsubárea y en temas de estudio.

Currículo: Ingeniería en Sistemas
.
.
6. Programación e Ingeniería de Software
6.1 Algorítmica
6.1.1 Fundamentos de algorítmica
PII: Historia de la Computación.
PI2: Algorítmica Básica.
PI3: Enfoque estructurado.
PI4: Enfoque por objetos.

Figura 2.2: Sección de un contenido curricular del campo de computación e informática.

El modelo OEM, se enfoca en el intercambio de información, basándose en un objeto que debe cumplir las características mostradas en la tabla 2.1, por lo cual, para el ejemplo de una sección de contenido curricular, mostrado en la figura 2.2, el objeto para el modelo OEM quedaría como se muestra en la tabla 2.2.

Etiquete	Tipo	Valor	ID Objeto
Contenidos Curriculares	String	set{**CC1**,é ,**CCn**}	1
CC1 = Ingeniería en Sistemas	String	set{**área, subárea, subsubárea, temas**}	1.1
área	String	**Programación e Ingeniería de SW**	1.1.6
subárea	String	**Algorítmica**	1.1.6.1
subsubárea	String	**Fundamentos de Algorítmica**	1.1.6.1.1
temas de estudio	String	set{**tema1, tema2, tema3, tema4**}	1.1.6.1.1.P1
tema 1	String	**Historia de la Computación**	1.1.6.1.1.P1.1
tema 2	String	**Algorítmica Básica**	1.1.6.1.1.P1.2
tema 3	String	**Enfoque Estructurado**	1.1.6.1.1.P1.3
tema 4	String	**Enfoque por Objetos**	1.1.6.1.1.P1.4

Tabla 2.2: Representación del contenido curricular de la figura 2.2, basado en el modelo OEM mostrado en la tabla 2.1.

El modelo OEM define una estructura para la obtención y almacenamiento de información semiestructurada, pero carece, de acuerdo a los fines que persigue BECCA, a través de la definición de un modelo de datos semiestructurado, de una metodología que indiquen los pasos a seguir para la representación, por medio de diagramas, de los elementos a almacenar, lo que dificulta, por ejemplo, llevar a un nivel de base de datos la información semiestructurada a manejar.

Otro aspecto que puede resultar contrario de ser aplicado el OEM en BECCA es con respecto al manejo eficiente de la información, por las siguientes razones:

- El manejo de un solo ID en OEM, como se muestra en la columna ***ID Objeto*** en la tabla 2.2, hace necesario ir concatenando el Id de un nuevo elemento hijo al Id del padre, por lo tanto y debido a que en información semiestructurada los niveles de anidamiento tienen un fondo arbitrario, un Id con estas características podría alcanzar longitudes muy grandes lo que complicaría su manejo.
- Las consultas a información usando OEM, se complicarían dado que no existe una metodología que indique como se lleva a base de datos la relación entre diversos elementos, por ejemplo: ¿Toda la información de todos los contenidos curriculares se almacena en una sola tabla?

2.3.2 El modelo de datos de árbol con agrupación de facetas (Trees with Grouping Facets)

El modelo descrito en [1], plantea organizar la información en una estructura tipo árbol, agregando una extensión llamada agrupación de facetas, con el fin de mejorar el manejo de los datos semiestructurados en éste tipo de estructuras. Éste enriquecimiento permite establecer una relación semántica explicita entre los elementos de datos en las bases de datos semiestructuradas. Algunas de las agrupaciones de facetas para el manejo de datos semiestructurados que se sugieren son las mostradas en la figura 2.3:

> *Conectores:* Para agrupar elementos se hace mediante los conectores ñandò, ñorò y ñxorò.
> *Orden:* Para especificar que un conjunto de elementos está en un orden específico o que no lo está, se hace mediante las etiquetas ñorderedò o ñunorderedò.
> *Repetición:* Para indicar si elementos del mismo tipo pueden repetirse, se usa la etiqueta ñrepetition allowedò de lo contrario se usa la etiqueta ñrepetition not allowedò.
> *Exclusión:* Para excluir ciertos elementos se indica mediante ñexcludedò.

Figura 2.3: Agrupaciones de facetas.

Debido a que las agrupaciones de facetas, descritas en la figura 2.3, no se pueden ilustrar en la figura 2.2, por que no incluye elementos de opción, repetición y exclusión, se presenta otro ejemplo de contenido curricular.

Ejemplo: Considerando un currículo de la Universidad de Munich, ilustrado en tabla 2.3, en los primeros cuatro elementos algunos cursos son opcionales mientras que otros cursos son requeridos, esto representado mediante conectores and y or, además se considera que la información mostrada ya tiene un orden definido, lo cual se puede representar con ñorderò, para indicar que los elementos del currículo no se pueden repetir se usa ñrepetition not allowedò y para eliminar un elemento se indica mediante ñexcludedò.

De acuerdo a este ejemplo, un alumno podría, en el semestre 3, cursar: CS III, Graph Theory y App. Análisis y un proyecto ya sea de: Programming o Systems o Logics pero no uno de Philosophy.

Terms	Comp. Sc.	Mathematics	Project
1	CS I	Álgebra I and Analysis I	
2	CS II and Hardware Basic	Algebra II	
3	CS III	Graph Theory And App. Analysis	Programming or Systems or Philosophy (excluded) or Logics
4	CS IV and Advanced Algorithms	Stochastics or Numerical Mathematics	

Tabla 2.3: Sección de un currículo de la Universidad de Munich [1].

Como datos semiestructurados, la sección número cuatro del contenido curricular mostrado en la Tabla 2.3, puede expresarse cómo muestra la Figura 2.4, y con agrupación de facetas como muestra la Figura 2.5:

```
<course_of_studies>
 ...

  <term>
    <number>4</number>
    <computer_sciences>
      <course>CS IV </course>
      <course>
        Advanced Algorithms  </course>
    </computer_sciences>
    <mathematics>
      <course>Stochastic</course>
      <course>
        Numerical Mathematics
      </course>
    </mathematics>
    <projects>
      <course>
        Programming   </course>
      <course>
        System Course   </course>
      <course>
        Philosophy   </course>
      <course>
        Logics   </course>
    </projects>
  </term>
</course_of_studies>
```

Figura 2.4: Código XML sin agrupación de facetas

```
<course_of_studies>
 ...
  <term>
    <ordered>
      <repetition not allowed>
        <number>4</number>
      </repetition not allowed>
        <computer_sciences>
          <AND>
            <course> CS IV </course>
            <course> Advanced Algorithms </course>
          </AND>
        </computer_sciences>
        <mathematics>
          <OR>
            <course> Stochastic </course>
            <course> Numerical Mathematics </course>
          </OR>
        </mathematics>
        <projects>
          <OR>
            <course> Programming </course>
            <course> System </course>
            <excluded>
              <course> Philosophy  </course>
            </excluded>
            <course> System </course>
          </OR>
        </projects>
    </ordered>
  </term>
</course_of_studies>
```

Figura 2.5: Código XML con agrupación de facetas

En la figura 2.4, se muestra la sección número cuatro del contenido curricular mostrado en la tabla 2.3 sin agrupación de facetas, y al estar organizada sin agrupación de

facetas hay información que se pierde; no se está expresando que cursos son opcionales y cuales son requeridos. De igual forma no se pude saber si la información del contenido curricular presentado debe llevar, forzosamente, el orden que tiene o que la información no está ordenada y no interesa. Tampoco se puede distinguir, en el contenido de la figura 2.4, si alguna sección del contenido curricular puede repetirse, por ejemplo, existir otra sección número 4.

La figura 2.5, muestra la organización de la sección cuatro del contenido curricular de la tabla 2.3, e incluye agrupación de facetas, con esta adición la sintaxis XML se mantiene y la información que se perdía en la figura 2.4 como: que materias son opcionales o requeridas, información que puede repetirse ó por ejemplo si existe un orden específico en el contenido, en la figura 2.5 se retiene sin necesidad de realizar otras operaciones para no perder dicha información.

La desventaja en éste modelo es que no todas las agrupaciones de facetas antes mencionadas se ajustan adecuadamente a las bases de datos y a lenguajes de modelación como entidad relación o diagramas de clases.

Además se imponen las siguientes restricciones para la agrupación de facetas:

- Solo una agrupación de facetas puede especificarse para un grupo de nodos.
- El grupo de facetas especificado siempre aplicará para todos los hijos inmediatos.
- El modelo de datos se limita a árboles.

Desde el punto de vista de los requerimientos de BECCA que plantea la necesidad de almacenamiento de información semiestructurada facilitando la obtención e integración para un uso y manejo eficiente de la información, el modelo de árbol con agrupación de facetas, crea problemas con las desventajas planteadas, ya que no todas las agrupaciones de facetas se adecuan a la base de datos y a lenguajes de modelación como entidad relación o diagramas de clases. Además de que no incluye facilidades para el manejo de información anidada.

2.3.3 Modelo de datos Entidad Relación para datos semiestructurados

El modelo Entidad Relación: Está basado en tres conceptos básicos: la entidad, los atributos y la relación. La *entidad* representa algo real o conceptual. Las entidades pueden ser un conjunto de objetos; con respecto a esto, está muy cerca de las *clases* dentro de los modelos orientados a objetos. Los *atributos* son las características de ese objeto. La *relación* es un objeto particular mediante el cual se unen dos o más entidades, una característica de la *relación* es que existe una cardinalidad entre la relación y una entidad, la cual se representa como uno ï muchos, muchos ï uno ó uno ï uno.

Extensión de los modelos E-R para datos semiestructurados: De acuerdo al modelo descrito en [11], para cualquier punto de vista que se tome, del modelo E-R, se nota que está pobremente equipado para representar los datos con estructuras irregulares,

por lo cual se sugieren dos pasos para dar al modelo E-R, la flexibilidad necesaria para representar en XML datos semiestructurados. Estas añadiduras tiene la ventaja adicional de resolver la ambigüedad sobre la interpretación de los atributos en el modelo E-R. Los dos pasos son:

- Marcar todos los atributos como opcionales o requeridos. En otras palabras el diseñador puede establecer una diferencia entre características inherentes o esenciales.
- Representar explícitamente un atributo *elección,* una para la cual la entidad podría tener o no tener un valor. Existe la posibilidad, en un futuro, de que en este modelo propuesto pueda marcarse la elección de una Entidad *e* como (exclusivo), es decir, solo se puede seleccionar un valor de las opciones, o viceversa, ser inclusivo, es decir, *e* puede tener el valor de ambas opciones (sólo 2 opciones pueden existir para este atributo).

Se propone que estas dos adiciones, se representen gráficamente como se muestra en la figura 2.6, la cual considera un contenido curricular dividido en área, subárea, subsubárea y los temas como atributos requeridos, y ejemplos y bibliografía sugerida como opcionales, en donde la representación de atributos *opcionales* serían asociados a la entidad correspondiente mediante una línea sólida la cual tendría dos líneas cruzándola y los atributos *requeridos* serían representados con la notación tradicional, es decir en óvalo, finalmente los atributos *elección* serían expresados mediante un triángulo, en donde las opciones estarían en los lados opuestos del triángulo.

Un contenido curricular, representa un plan de estudios que indica las materias (divididas en: áreas, subáres, etc), bibliografía, ejemplos e información extra que componen un grado académico.

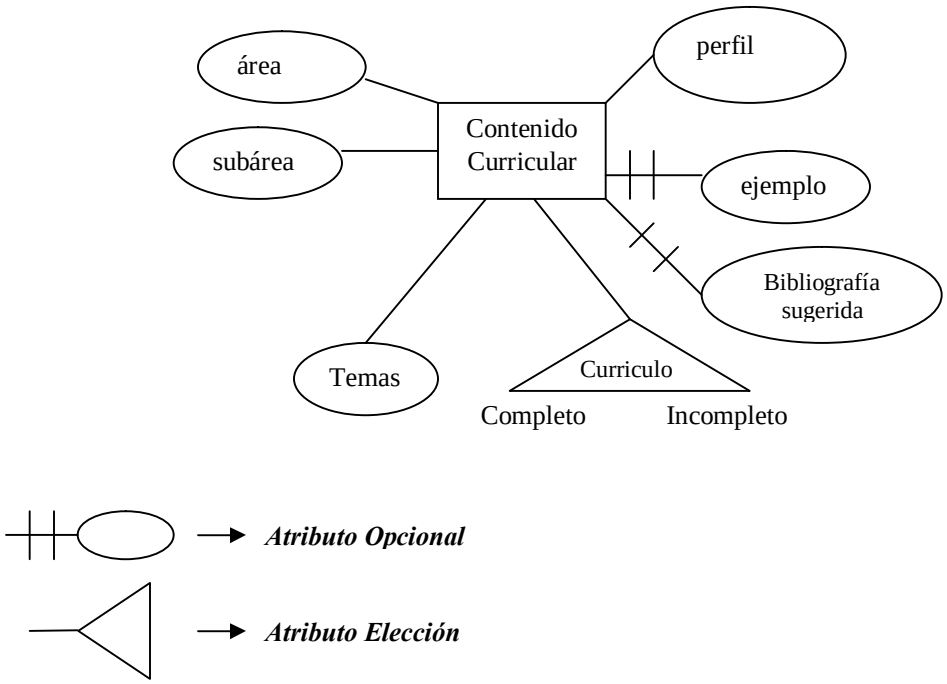


Figura 2.6. Ejemplo del uso del modelo Entidad Relación para datos semiestructurados.

El contenido de la figura 2.2, no puede ser representada en este modelo, por que no describe que pasa con el almacenamiento de información anidada hasta fondos arbitrarios y como se representa información que no fue prevista.

El modelo entidad relación para datos semiestructurados tiene como propósito diseñar la base de datos en donde se almacenarán datos semiestructurados, tomando en cuenta características que son requeridas, opcionales y aquellas en las que se pude elegir una opción. BECCA requiere un almacenamiento eficiente de información, el cual el modelo entidad relación para datos semiestructurados cumple parcialmente, debido a que también existe la posibilidad de que se presente información no prevista, de la cual no se conoce ni sus campos o el tipo de información que provee y en donde entidad relación para datos semiestructurados no plantea una solución. Además de que no facilita la obtención de información debido a que no plantea una estructura de representación.

2.3.4 Modelo de datos HyperFile

El modelo de datos HyperFile [21], emergió como un modelo para el manejo de información semiestructurada. La idea principal es ver los datos como una colección de objetos que están ligados en diferentes formas.

El modelo HyperFile, propone clases de objetos únicos, tipo freeform. Esencialmente cada objeto es modelado como un conjunto de tuplas de la forma: <*tuple_type, key, data*> Cada tupla representa una propiedad de un nodo, con *tuple_type* siendo el nombre, el

data siendo el valor y *key* siendo una propiedad corta que distingue el valor de otras tuplas. Estas tuplas pueden contener texto, datos de imagen, llaves, información bibliográfica, referencias y punteros a otros objetos.

Pointer	Contenido Curricular	set{**CC1**,é ,**CCn**}
String	**CC1 = Ingeniería en Sistemas**	set{**área, subárea, subsubárea, temas**}
String	**área**	**Programación e Ingeniería de SW**
String	**subárea**	**Algorítmica**
String	**subsubárea**	**Fundamentos de Algorítmica**
String	**temas**	set{**tema1, tema2, tema3, tema4**}
String	**tema 1**	**Historia de la Computación**
String	**tema 2**	**Algorítmica Básica**
String	**tema 3**	**Enfoque Estructurado**
String	**tema 4**	**Enfoque por Objetos**

Tabla 2.4: Representación del contenido curricular de la figura 2.2, basado en el modelo HyperFile.

En la tabla 2.4, se muestra el contenido curricular de la figura 2.2, aplicándole el modelo HyperFile. Una clara desventaja que se aprecia en el modelo es que no existe una referencia, por ejemplo un identificador, que facilite el que a partir de una tupla se pueda regresar al elemento que la contiene. Por lo tanto, se complicaría el manejo de los datos semiestructurados almacenados en la base de datos, objetivo que es importante en BECCA. Otra característica importante que se maneja dentro de BECCA es poder realizar consultas sobre los diferentes elementos dentro de la información almacenada, lo cual con Hyperfile no se facilita debido a que la información no se clasifica y toda se representa en una sola estructura.

El modelo de datos semiestructurados OEM y el modelo de datos semiestructurados HyperFile, son modelos que plantean una estructura semejante para la representación de la información, la diferencia radica en que el modelo de datos semiestructurados OEM agrega un ID a en la estructura para la representación de la información, con el propósito de tener un mayor control sobre la información almacenada.

Los modelos de datos semiestructurados, como OEM, árbol con agrupación de facetas, entidad relación para datos semiestructurados e Hyperfile, son usados en sistemas dedicados al manejo de datos semiestructurados. Un modelo de datos, como parte de un sistema, es el que rige el manejo que el sistema le da a los datos que usa.

2.4 Sistemas basados en modelos de datos semiestructurados

Algunos sistemas existentes que manejan datos semiestructurados son Tsimmis [15], Pegasus [16], Strudel [17] y DISCO [18], estos son sistemas que se crearon con el propósito de lograr un mejor uso de la información proveniente de diferentes fuentes sus características principales son:

2.4.1 Sistema Tsimmis

El sistema Tsimmis [15] provee de las herramientas necesarias para acceder, de una manera integrada, a múltiples fuentes de información con la seguridad de que la información obtenida es consistente. Algunas características del proyecto Tsimmis son:

- Tsimmis se enfoca en proveer acceso integrado a muy diversa y dinámica información. La información puede ser semiestructurada o no estructurada. Además, las fuentes disponibles, sus contenidos, y el significado de sus contenidos podrían cambiar frecuentemente.
- La integración en este ambiente requiere más participación humana. En un caso extremo, la integración es realizada manualmente por el usuario.

En suma, la meta de Tsimmis no es el realizar la integración completamente automatizada de la información, sino proveer de un marco de trabajo y herramientas para asistir al usuario [15].

Arquitectura del Sistema: La arquitectura del sistema Tsimmis, ilustrada en la figura 2.7, consta de varios elementos que muestran como se procesa una petición de información y los elementos por los que debe de pasar, desde el usuario hasta la base de datos a la que se consulta, entre los elementos que destacan en la arquitectura están:

- *Translator (*o *wrapper)*: La función de *translator* es convertir los objetos de datos a un modelo de información común. Para realizar una interpretación lógica, *translator* convierte consultas sobre información a un modelo común de petición que la fuente a la que se consulta, pueda ejecutar y de esta forma convertir también la respuesta dada por la fuente en un modelo común.
- - *Mediator*: El *Mediator* es un sistema que refina, en cierta forma, información de una o más fuentes. Un *Mediator* incrusta los conocimientos que son necesarios para procesar cierta clase de información.
- *Constraint Manager:* Se encarga de identificar los requerimientos de consistencia semántica sobre información almacenada.

Implementación Tsimmis: El sistema Tsimmis se implementó en un escenario que consiste de una gran diversidad de fuentes de información bibliográfica, que incluyen sistemas de información de bibliotecas, bases de datos relacionales conteniendo registros bibliográficos estructurados y sistemas de archivos con bibliografía no estructurada. Usando como base el modelo OEM, descrito anteriormente, se accede a sistemas tales como el Stanford University Folio System. El folio provee acceso a cuarenta repositorios, incluyendo un catálogo del contenido de bibliotecas de Stanford y de varias fuentes comerciales como INSPEC, que tiene información de ciencias computacionales y otras publicaciones y artículos [14].

La forma de obtener objetos OEM de las fuentes de información, como las antes mencionadas de Stanford, se hace mediante el lenguaje OEM-QL (Object Exchange

Model ï Query Language) [14]. La forma básica de realizar una consulta es mediante las instrucciones SELECT-FROM-WHERE [14].

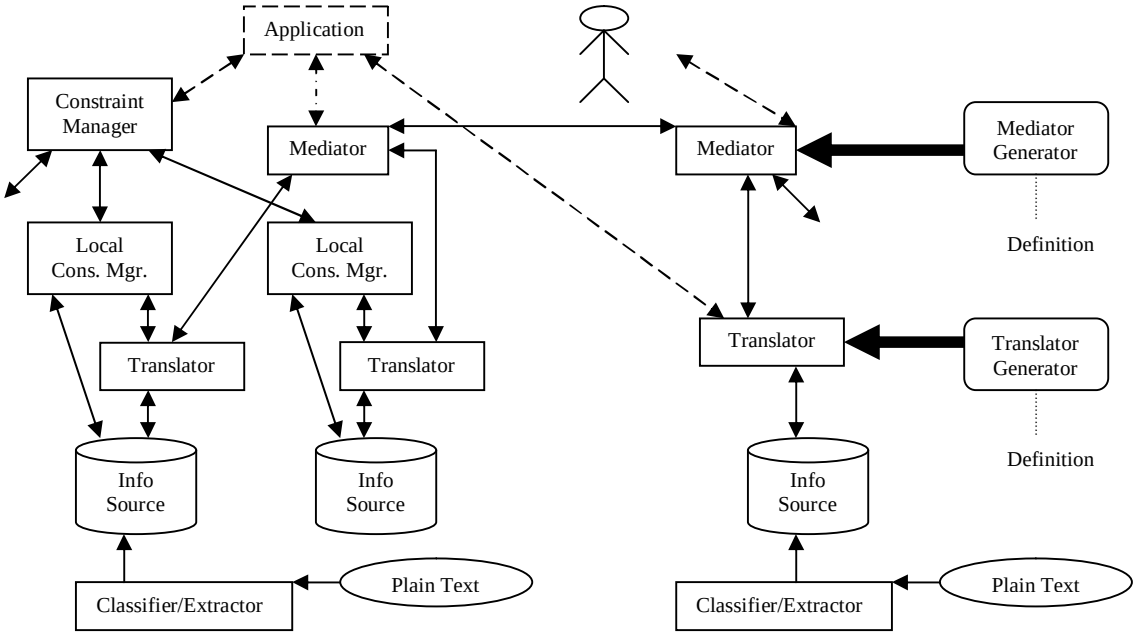


Figura 2.7: Arquitectura Tsimmis [15].

2.4.2 Sistema Pegasus

Pegasus [16], es un sistema manejador de múltiples bases de datos heterogéneas, proyecto creado por el departamento de tecnología de Base de Datos de los laboratorios de Hewlett-Packard, para responder a la necesidad del manejo y acceso a datos compartidos a través de de un amplio rango de aplicaciones. El objetivo de un sistema de múltiples bases de datos heterogéneas es soportar varios sistemas de bases de datos con diferentes modelos de base de datos, lenguajes y servicios. El propósito de Pegasus es proveer de una interfase integrada y uniforme para la obtención de información de bases de datos heterogéneas existentes.

Modelo de Datos: El modelo de datos usado en el sistema Pegasus es llamado también Pegasus [16], éste modelo orientado a objetos sirve como un marco de trabajo para uniformar la interoperación de múltiples fuentes de datos con diferentes sistemas administradores de datos, este modelo de datos contiene tres términos básicos:

- Los *Tipos* que tienen un único nombre y representan colecciones de objetos que comparten características comunes. Los tipos son organizados en grafos acíclicos dirigidos que soportan generalizaciones o especializaciones. Un tipo puede ser declarado para ser un subtipo de otro tipo. Una función definida en un tipo dado

es también definida en todos los subtipos. Objetos que son instancias de un tipo son también instancias de un supertipo.

- Los *Objetos* que son identificados por sus identificadores de objetos. Algunos objetos como los enteros, son autoidentificados. Una carácterística de los objetos es que pueden perder o ganar *tipos* dinámicamente. Por ejemplo, un objeto representando una persona dada, puede ser creado como una instancia del tipo estudiante, después éste pude perder el tipo estudiante y adquirir el tipo empleado.
- Las *Funciones* que son las manifestaciones de operaciones y proveen de mapeos entre objetos. Propiedades en los objetos son representados en términos de funciones. Los argumentos y resultados de las funciones son los tipos.

Lenguaje de Consulta: El lenguaje de manipulación de datos en Pegasus es el lenguaje llamado HOSQL (Heterogeneous Object Structured Query Language) [16], que es un lenguaje de consulta orientado a objetos que provee de declaraciones para manipular múltiples bases de datos heterogéneas.

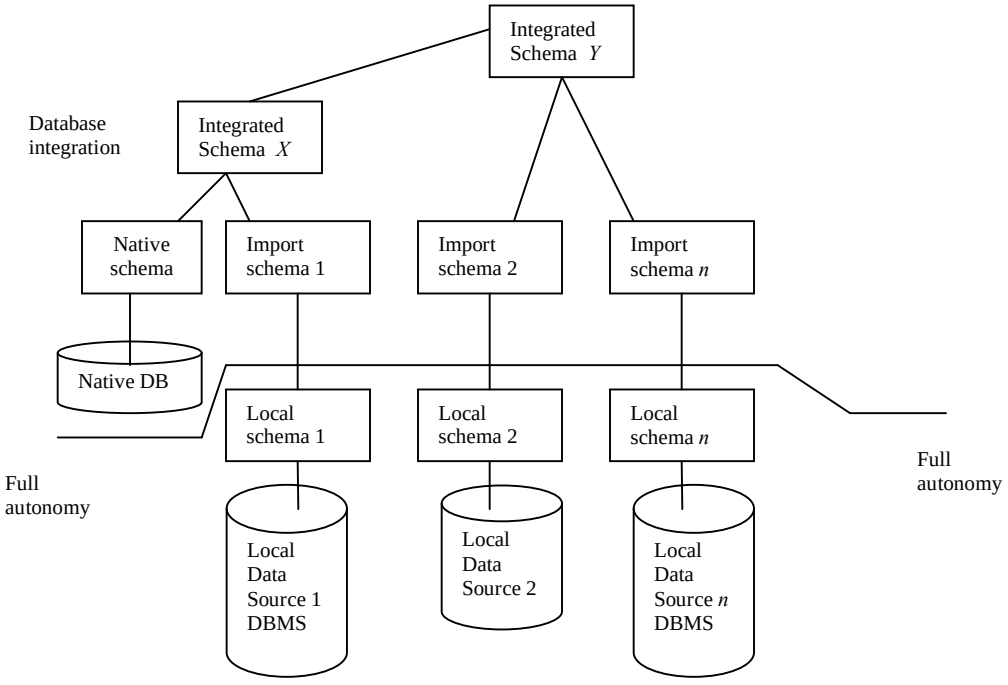


Figura 2.8: Configuración del Sistema de Base de Datos Pegasus [16].

Configuración de Pegasus: En la figura 2.8, se muestra la configuración para la integración de Bases de Datos para un Sistema Pegasus en donde: una fuente de datos es típicamente una Base de Datos, aunque esta puede ser también de otro tipo (como un sistema de archivos). Las fuentes de datos no directamente controladas por Pegasus son llamadas fuentes de datos locales. Una fuente de datos es representada en Pegasus al importar un esquema que parece como un esquema Pegasus. En una base de datos nativa de Pegasus tanto el esquema como los datos son manejados por Pegasus.

2.4.3 STRUDEL

STRUDEL [17] es un sistema manejador de sitios Web y la idea principal es la separación de la apreciación lógica de la información disponible en sitios Web, la estructura de esa información en páginas ligadas y la presentación gráfica de páginas en HTML. La construcción de un sitio Web usando STRUDEL envuelve dos pasos: El primero es que el administrador del sitio Web define los datos que estarán disponibles en el sitio. El segundo paso consiste en que el administrador del sitio Web define como estructura y presenta esos datos. La aportación de STRUDEL no es el proveer de un nuevo editor HTML sino proveer de herramientas para la integración de datos de diferentes fuentes.

Modelo de Datos: El modelo de datos grafos uniformes [17], es el modelo que STRUDEL usa para la integración de datos de diferentes fuentes. En cada nivel del sistema STRUDEL, los datos son vistos uniformemente como un grafo. Los grafos en este modelo, contiene objetos, también llamados nodos, conectados por aristas etiquetadas con el nombre del atributo [17], el modelo de grafos uniformes es muy similar al modelo OEM [14], mencionado anteriormente.

Características del sistema: El uso de STRUDEL para construir y manejar sitios Web provee de los siguientes beneficios:

- STRUDEL provee de mecanismos que permite la integración automática de múltiples fuentes de datos. El sistema no requiere que se migren todos los datos dentro del un solo repositorio.
- Cuando un sitio Web es construido como resultado de una consulta, el usuario puede construir estructuras adicionales para describir el sitio Web.
- Como STRUDEL usa un modelo de datos de grafos, sitios Web existentes pueden ser incorporados directamente al sistema STRUDEL.

Arquitectura del Sistema: La figura 2.9, muestra la Arquitectura de STRUDEL, en donde el *mediator* integra múltiples colecciones y objetos en un grafo de datos. El *Site Management* procesa consultas que computa el *Site Graph,* los cuales son vistas de *Data Graph.* Un *Site Graph* es también un grafo cuyos nodos contienen los atributos que especifican como desplegar en HTML el contenido de los nodos. Finalmente el *HTML generator* materializa un sitio gráfico como una página HTML.

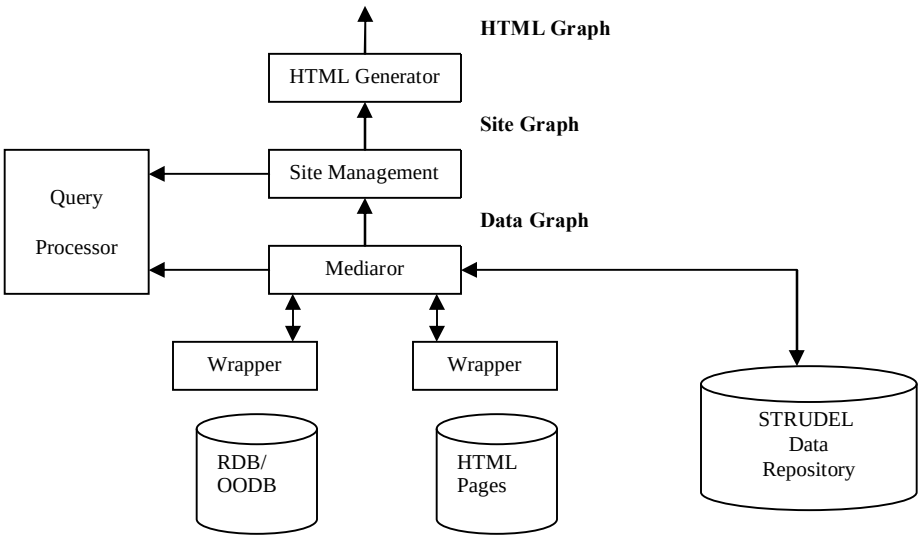


Figura 2.9: Arquitectura de STRUDEL [17].

2.4.4 Sistema DISCO (Distributed Information Search COmponent)

DISCO [18], es un sistema de base de datos distribuidas heterogéneas que pude ir incrementando la cantidad de fuentes de datos que tiene. Algunas de las ventajas de DISCO son:

- Provee de características que permite a los usuarios manejar el número de fuentes de datos que va agregando.
- DISCO ayuda al administrador de la base de datos en la adición de fuetes de datos si las nuevas fuentes de datos son similares a las fuentes de datos ya existentes.
- DISCO también apoya en la transformación de tipos cuando hay semejanza entre los tipos.

Modelo de Datos: El modelo de datos usado en este sistema es también nombrado DISCO [18], basado en el modelo ODMG 2.0 [18]. El modelo de objetos ODMG se basa de acuerdo a un sistema de tipo. Los tipos pueden ser atómicos o estructurados. Los tipos atómicos son predefinidos, como por ejemplo el entero, boléanos o strings. Los tipos estructurados son por ejemplo las listas y tablas. Las expresiones de tipos se construyen a través de aplicaciones recursivas de constructores tipos estructurados a tipos atómicos y tipos de expresión. Los tipos objeto son descritos en los modelos de datos a través de una interfase de objetos. La interfase de objetos especifica las propiedades (atributos y relaciones) y operaciones o métodos que son características de las instancias de esos tipos objeto. Una relación es un atributo de valor de referencia a un tipo objeto

Arquitectura Disco: La figura 2.10, representa la arquitectura de Disco, en donde el usuario final interactúa con la Aplicación. La Aplicación accede a la fuente de datos vía el Mediador. El Mediador exporta un esquema, que es una representación integrada de la fuente de datos. El Mediador procesa consultas sobre la representación integrada. Para

realizar la representación el administrador de la base de datos provee de información al Mediador para finalizar el proceso de consulta. Para tratar con diferentes lenguajes de consulta en cada fuente de datos, los Wrappers transforman las consultas enviadas a la fuente de datos y también transforma las respuestas enviadas por la fuente de datos.

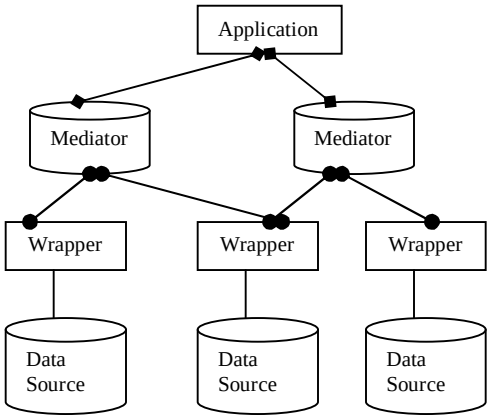


Figura 2.10: Arquitectura Disco [18].

Un problema que se presenta en un escenario de datos semiestructurados, es la integración de información.

Cabe destacar que los sistemas existentes que manejan datos semiestructurados como: Pegasus [16], Strudel [17] y DISCO [18], hacen referencia a modelos de datos semiestructurados cuyos nombres corresponden a cada sistema, y de los cuáles se buscó información para incorporarla en esta investigación, pero debido a la falta de documentación referente a cada modelo no sé agregó.

2.5 Proyecto de Base Electrónica de Contenidos Curriculares para Acreditación en Educación Superior (BECCA)

El proyecto Bymx, propone definir una ñArquitectura para la Integración de Información, Aplicaciones y Servicios en Webò, e investigar y desarrollar soluciones para la integración de información, servicios y aplicaciones, considerando a la Web como el principal proveedor de información.

Al producto que se desarrollará en la arquitectura Bymx se le denominó BECCA**,** proyecto para el cual se desarrollará el modelo de datos que aquí se menciona.

El proyecto BECCA propone desarrollar la infraestructura necesaria para la creación y mantenimiento de contenidos curriculares disponibles a través de la Internet y dispositivos móviles que ayuden a la uniformidad en la enseñanza de las carreras profesionales en el campo de informática y computación.

El objetivo final de BECCA es desarrollar una base educativa que ayude a la gestión de los contenidos curriculares, esto para el campo de computación e informática, y la cual se organizará en áreas temáticas (sub-área, temas y sub-temas). De esta forma se facilitaría la adecuación, extracción y uso de los datos al momento de crear los materiales educativos para ésta área [20].

2.5.1 Características de los Contenidos Curriculares del Campo de Computación e Informática

Los modelos curriculares del campo de computación e informática que se almacenarán en BECCA abarcarán los siguientes perfiles profesionales, de acuerdo a la clasificación presentada en la guía para el examen general de egreso de las carreras de informática y computación de carácter opcional que presenta el CENEVAL (Centro Nacional de Evaluación para la Educación Superior):

- Licenciatura en Informática.
- Licenciatura en Sistemas Computacionales.
- Licenciatura en Ciencias de la Computación.
- Ingeniería en Sistemas.

Cada perfil profesional del campo de computación e informática, considerados para manejar en BECCA, tiene identificados campos de conocimientos, por ejemplo, la Licenciatura en Informática considera los siguientes campos de conocimiento:

- Entorno Social.
- Matemáticas.
- Arquitectura de Computación.
- Redes.
- Software de base.
- Programación e ingeniería de software.
- Tratamiento de información.
- Interacción Humano-máquina.

La estructura de los contenidos curriculares del campo de computación e informática podría tener la estructura mostrada en la figura 2.11, en dónde el contenido curricular se divide en áreas, por ejemplo Entorno Social, en subáreas como por ejemplo las Organizaciones, en Subsubáreas como podría ser la Teoría de las Organizaciones y finalmente se muestran sus temas de estudio como por ejemplo Tipos y principios básicos de las organizaciones. Adicionalmente cada área, subáreas, etc. Puede almacenar imágenes, ligas de Internet, etc.

1. Entorno Social
Descripción
1.1 Las Organizaciones
Objetivo
1.1.1 Teoría de las Organizaciones
ES1: Tipos y principios básicos de las organizaciones
ES2: Procedimientos administrativos
ES3: Recursos Humanos

Figura 2.11: Sección de un Contenido Curricular del campo de computación e informática.

Las operaciones que se pretenden realizar en BECCA sobre los contenidos curriculares del campo de computación e informática son los siguientes:

- Obtención por parte de BECCA de contenidos curriculares de alguna fuente de información.
- Envío a BECCA de contenidos curriculares en algún formato de edición con información complementaría en XML.
- Registrar a cada autor de algún contenido curricular, de tal forma que exista responsabilidad por los contenidos curriculares.
- Registro de subscriptores con el propósito de informarles, dado el caso, que la información que usan ha sido modificada por el autor.
- Además crear el registro de un revisor asociado a un autor que verifique la veracidad de la información.
- Integración de contenidos seleccionados para ser suministrados en formato PDF o HTML a un subscriptor.
- Reconstruir los documentos almacenados en la forma original en que fueron obtenidos.

2.5.2 Infraestructura BECCA

Con el propósito de mostrar el funcionamiento de BECCA, en la figura 2.12 se representa una visualización general de BECCA en su primera etapa, en donde la información estará centralizada en un servidor:

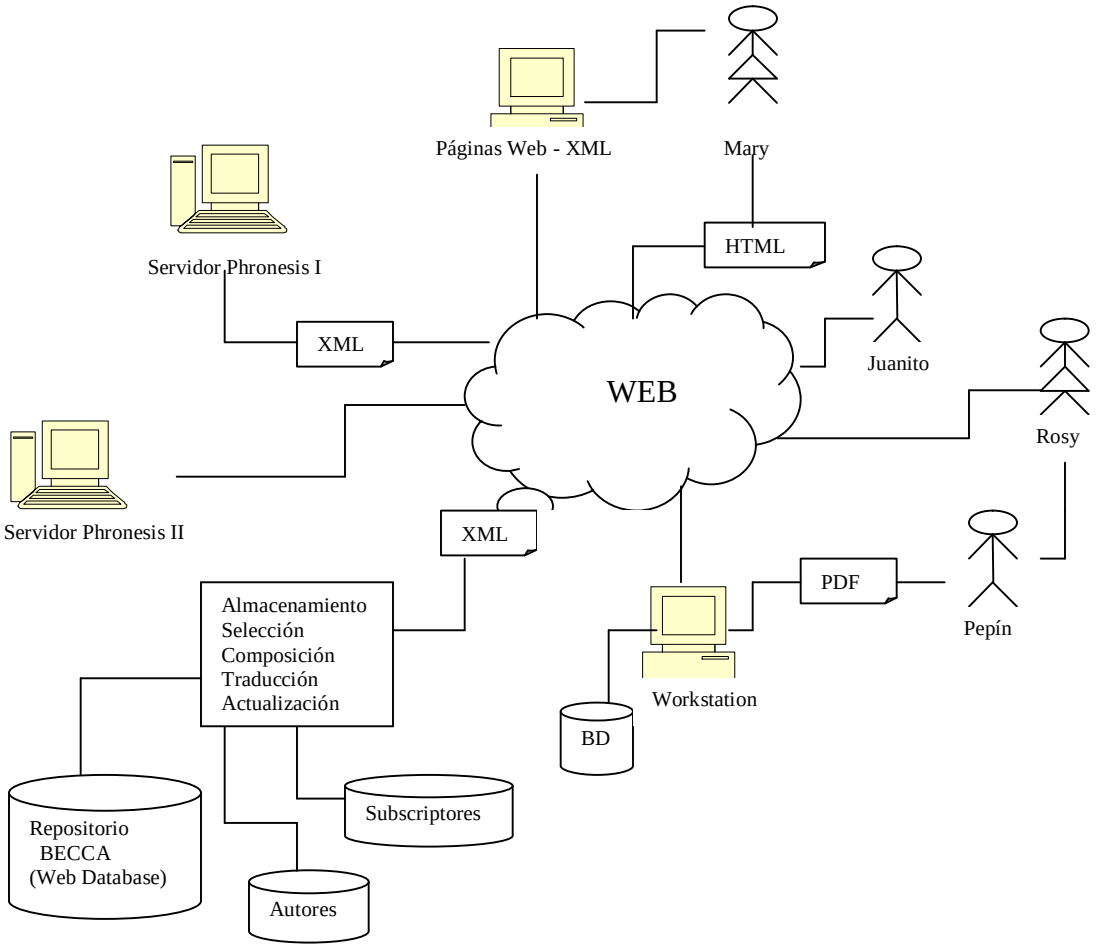


Figura 2.12: Infraestructura de BECCA [20].

Una explicación del funcionamiento de BECCA, basado en la figura 2.12, es la siguiente: Juanito y Mary son autores de contenidos curriculares que envían sus contenidos en algún formato de edición (MS Word, AbiWord, Framemaker o un editor de XML). Los contenidos curriculares se almacenarían en la base llamada BECCA, con información complementaría en formato XML. Los autores se registrarían con el propósito de que exista responsabilidad por los contenidos almacenados. Los subscriptores, por ejemplo Rosy, sería el individuo que extraería información curricular. Dentro BECCA existirá un registró de subscriptores, esto con el propósito de informarles cuando el contenido que ellos utilicen haya sido modificado por el autor. Junto a los datos del autor, deberían existir los datos de un revisor, para que la veracidad de la información almacenada pueda ser constatada [20].

De acuerdo a la infraestructura de BECCA, ver figura 2.12, el proceso de traducción de (un formato de representación a otro) y de composición (integración de los sub-temas, temas y unidades), se realiza en un procesador cercano a donde radican los contenidos. Puede observarse también, en la figura 2.12, que un subscriptor, por ejemplo Mary,

pudiera solicitar que los contenidos seleccionados sean integrados y le sean suministrados en formato PDF o HTML. El formato HTML, sería utilizado para consultarse mediante un navegador, por ejemplo Internet Explorer, Netscape, etc. En este último caso, el subscriptor será el responsable de colocar los contenidos HTML en un servidor que él pueda administrar. La tarea de BECCA terminará después de que los contenidos han sido suministrados al subscriptor [20].

Los requerimientos de BECCA con respecto al modelo de datos serán los siguientes:

- Crear un método de obtención de información de diferentes fuentes para almacenarla en la base de datos BECCA.
- Definir la forma de almacenamiento de la información en la base de datos llamada BECCA.
- Desarrollar un método para la integración de la información seleccionada de BECCA y presentarla como un solo documento.

2.6 Aplicación de los modelos anteriores en BECCA

Ninguno de los modelos para el manejo de datos semiestructurados mencionados anteriormente como: OEM, árbol con agrupación de facetas, entidad relación para datos semiestructurados e Hyperfile, cumple con todos los puntos deseables en un modelo de datos para BECCA como:

- *Almacenar datos semiestructurados:* El problema de almacenamiento se refiere al establecimiento del diseño que debe tener la base de datos para almacenar los diferentes documentos de información semiestructurada que se obtengan. El diseño debe tomar en cuenta la información que debe ser requerida, la información que puede ser opcional y además ser flexible para almacenar información que contenga más información junto con la organización del documento, esto con el propósito de no perder características que sean esenciales.

Además se busca que los modelos faciliten:

- *Obtener información:* Se refiere a como se obtendrá la información que será almacenada, lo más completa posible sin perder características como: orden y tipo de información.
- *El integrar información de diferentes documentos que hablan sobre el mismo tema:* La integración de información es un problema que se presenta en un ambiente de integración de diferentes fuentes. El problema se refiere a una decisión que se debe de tomar con respecto ha si dos objetos, por ejemplo dos tablas, se refieren a la misma entidad del mundo real, para que en dado caso de que sí se refieran a la misma entidad, la información que se obtenga extraída de ambos objetos se complemente y además se evite obtener información repetida [19]. Esta parte es importante en BECCA para poder realizar consulta de información mediante a través de diferentes parámetros de búsqueda.

Por lo tanto su implementación en BECCA no resultaría en un adecuado manejo de la información, además de que OEM, árbol con agrupación de facetas, entidad relación para datos semiestructurados e Hyperfile, no se diseñaron pensando en las características de los contenidos curriculares del campo de computación e informática.

La Tabla 2.5, muestra los modelos OEM, árbol con agrupación de facetas y entidad relación para datos semiestructurados y marca los puntos que cubren para el modelo de datos que se busca para BECCA, ninguno de ellos cubrió todos los puntos requeridos:

Modelo	**Obtención de información**	**Integración de información**	**Almacenamiento de la información semiestructurada**
OEM	X		X
Árbol con agrupación de facetas	X		X
Entidad - Relación para datos semiestructurados			X
HyperFile	X		X
Modelo Propuesto	X	X	X

Tabla 2.5: Modelos de datos semiestructurados y los aspectos que abarcan.

La tabla 2.5 indica que OEM, Árbol con Agrupación de Facetas, Entidad Relación para datos semiestructurados e Hyperfile, son modelos que plantean almacenamiento y obtención de información, sin embargo, tanto el almacenamiento y la obtención en estos modelos, presentan deficiencias, para el caso de almacenamiento, pierde elementos como orden y presentan complicaciones en el manejo de información anidada a fondos arbitrarios. Para el caso de obtención de información, no indican que realizar para manejar los diferentes tipos de elementos, con diferentes estructuras que puedan presentarse y que hacer con información inesperada. Para integración de información, los modelos existentes OEM, Árbol con Agrupación de Facetas, Entidad Relación para datos semiestructurados e Hyperfile, no plantean como organizar la información para poder realizar diversas consultas sobre la información almacenada.

2.7 Necesidad de Definir un Nuevo Modelo de Almacenamiento para Información Semiestructurada

Los modelos de datos analizados, OEM, árbol con agrupación de facetas y Entidad ï Relación para datos semiestructurados e Hyperfile no cumplen con todos los

requerimientos que BECCA necesita defina y facilite un modelo de datos, ver tabla 2.5, es por tal motivo que se necesita definir un modelo de datos que satisfaga las necesidades de BECCA, para que de este forma BECCA pueda llevar a cabo todas las operaciones que pretende realizar sobre los contenidos curriculares del campo de computación e informática.

El modelo de datos para la obtención, integración, almacenamiento que se definió para esta tesis es de un propósito más particular y específico, porque se orienta al manejo específico de los datos de los contenidos curriculares del campo de computación e informática, descritos anteriormente, y es particular porque se desarrollará para su implementación en BECCA. El modelo de datos pretende cubrir los problemas que se presentan en el manejo de los datos semiestructurados como la obtención, almacenamiento e integración de los datos de diferentes fuentes tomando en cuanta las características específicas de los contenidos curriculares de computación e informática.

2.8 Resumen

En éste Capítulo se abordó el tema de XML y su aportación en las facilidades que ofrece para el manejo de información semiestructurada, además se estudiaron los modelos de almacenamiento de información semiestructurada.

Se distinguieron entre dos tipos de modelos semiestructurados: modelos complejos y modelos free ï form, presentando ejemplos de modelos free ï form existentes: OEM, árbol con agrupación de facetas, entidad relación para datos semiestructurados e Hyperfile.

Se analizaron también las características de los contenidos curriculares orientados al campo de computación e informática, con el propósito de examinar si los modelos, vistos en este Capítulo, eran capaces de manejar la información semiestructurada de los contenidos curriculares, llegando a la conclusión de la necesidad de definir un nuevo modelo de datos semiestructurados capaz de manejar adecuadamente información semiestructurada como la contenida en contenidos curriculares, el cuál sería aplicado en BECCA, proyecto que propone desarrollar la infraestructura necesaria para la creación y mantenimiento de contenidos curriculares.

Capítulo 3

Desarrollo del Modelo de Datos

Este Capítulo presenta la propuesta para el desarrollo del modelo, para el manejo de datos semiestructurados, aplicado a contenidos curriculares del campo de computación e informática. El desarrollo comprende los pasos realizados para la creación del modelo y los elementos obtenidos para su aplicación. En la sección 3.1 se presenta el concepto de modelos de datos semiestructurados. La sección 3.2 presenta la propuesta para la definición del modelo de datos semiestructurado. La sección 3.3 presenta la metodología para el desarrollo del modelo de datos semiestructurado propuesto. En la sección 3.4 se presenta la aplicación de la metodología de la sección 3.3 a un caso práctico de contenidos curriculares. Finalmente el Capítulo concluye con un resumen presentado en la sección 3.5.

3.1 Los Modelos de Datos Semiestructurados

En el Capítulo 2 se destacó la importancia de los modelos de almacenamiento de datos semiestructurados en el manejo de este tipo de información. Así mismo se analizaron propuestas de modelos de datos semiestructurados existentes como: Object Exchange Model [14], árbol con agrupación de facetas [1], entidad relación para datos semiestructurados [11] y el modelo de datos HyperFile [21].

Como se explicó en el Capítulo 2, los modelos de datos semiestructurados existentes se distinguen por definirse entre dos opciones: en un extremo modelos complejos y en el otro extremo modelos de forma libre [21]. Los modelos complejos definen diferentes clases de objetos al predeterminar todos los tipos de objetos y sus estructuras que pueden presentarse. El modelo forma libre, en contraste con el modelo complejo, plantea la definición de una estructura que almacene cualquier tipo de información.

El problema con los tipos complejos y free ï form, y ejemplos de modelos de datos semiestructurados analizados en el Capítulo 2: Object Exchange Model, árbol con agrupación de facetas, entidad relación para datos semiestructurados y el modelo de datos HyperFile, es que no cumplen con los puntos que se buscó cubriera un modelo de datos para el almacenamiento de datos semiestructurados aplicados al manejo de contenidos curriculares, descritos en la tabla 2.5.

3.2 Propuesta para la Definición del Modelo de Datos Semiestructurados

Para esta investigación, orientada al manejo de información semiestructurada, se plantea un escenario en el que sí se conocen ciertos elementos de la información a almacenar, pero se dejará también abierta la posibilidad de almacenar información que no

se previó. Además de activar mecanismos, en el diseño, que no hagan necesario que un documento con información semiestructurada, deba contener los elementos que fueron previstos.

El modelo planteado se encuentra entre un modelo complejo y un modelo free-form, la principal ventaja es que el almacenamiento de la información se realizará en forma más ordenada y sencilla y cuyas características son:

- Se sabe en donde se encuentran datos básicos que se prevén y de los cuales se conoce los campos que contienen, con lo que se logrará un mayor control y mejor uso de la información.
- Además de dejar abierta la posibilidad de almacenar de forma correcta, simple y eficiente información no prevista y con estructura no conocida.

El modelo para el manejo de datos semiestructurados que se plantea en este documento, se orienta al manejo de la información de contenidos curriculares del campo de computación e informática, tomando en cuenta las operaciones que se desean hacer sobre la información en BECCA (Base Electrónica de Contenido Curriculares para Acreditación en Educación Superior) [20].

3.3 Metodología para Definir el Modelo de Datos Semiestructurado

En esta sección se describe una metodología que tiene como propósito principal definir un modelo de datos semiestructurado y un conjunto de herramientas que faciliten el manejo de la información semiestructurada. El modelo de datos resultante tiene las características, descritas en la sección 3.2, de definir estructuras conocidas y también permitir manejar información con estructuras desconocidas.

A partir del análisis de la información semiestructurada del tópico de interés que se desea manejar, mediante la metodología propuesta en esta sección, se obtienen:

- Estructuras de representación de información.
- Un DTD (Document Type Definition) para especificar las restricciones en la estructura y sintaxis describiendo: los elementos (etiquetas que son permitidas y su contenido), estructura (orden en que van las etiquetas) y el anidamiento (qué etiquetas van dentro de otras) que un documento XML (Extensible Markup Language) [7] con información semiestructurada debe cumplir.
- un modelo de datos semiestructurados a partir del cuál se define la relación entre los diversos objetos de la información.

La metodología propuesta consta de 6 pasos, los cuales a su vez se categorizan en 3 grupos con el fin de una mejor comprensión.

El primer grupo, que abarca los pasos 1 y 2, tiene como objetivo definir las estructuras que manejan los datos básicos que se prevén y de los cuales se conoce los campos que contienen, con lo que se logra un mayor control y mejor uso de la información, ventajas que ofrece un modelo de datos semiestructurado complejo. Los pasos comprendidos en este grupo son:

1. Identificar elementos, en la información a manejar, cuya estructura es conocida y estática, es decir no cambian. Este paso busca, entre las diversas muestras de información semiestructurada a almacenar, datos básicos o bien, que están presentes en la mayoría de las muestras y selecciona aquellos datos cuya estructura no cambia entre las muestras.
2. Identificar elementos, en la información que pueden presentarse y cuya estructura no es estática. Este paso busca, entre las muestras de información semiestructurada a almacenar, datos presentes en la mayoría de las muestras, y selecciona datos cuya estructura varía entre las muestras.

El segundo grupo, que comprende los pasos 3 y 4, tiene como objetivo definir la estructura que permite manejar de forma correcta, simple y eficiente información no prevista y con estructura no conocida y que son, para este grupo, las ventajas ofrecidas un modelo free-form. Los pasos definidos para este grupo son:

3. Identificar variaciones en la información. Este paso tiene como propósito buscar en las muestras de información a almacenar, variaciones de datos que puedan manejarse para evitar crear nuevas estructuras de representación.
4. Definir la forma de almacenar información no esperada y de la que no se conoce su estructura. En este paso se define la estructura genérica que permite almacenar y representar información sin estructura definida, que tengan niveles de anidamiento indefinido y de la cual no se conoce su estructura.

El tercer grupo, que comprende los pasos 5 y 6, permiten definir las herramientas para concretar la definición del modelo de almacenamiento. Los pasos de este grupo son:

5. Modelar la estructura de almacenamiento, tomando en cuenta los pasos previos. En este paso se define el modelo de almacenamiento semiestructurado como resultado de los pasos 1 al 4.
6. Definir una estructura para representar información en un documento XML mediante un DTD. Con este paso se definen las reglas para la representación de la información semiestructurada en formato XML.

El conjunto de pasos, de la metodología descrita, permiten la creación de un modelo entre las opciones existentes de modelos semiestructurados: complejos y free-form, evitando heredar las desventajas de ambos modelos, por que:

- Con el primer grupo, que representa al modelo complejo, se evita depender completamente de una estructura genérica para el almacenamiento de todos los datos, lo que puede complicar el manejo de información, y que es la desventaja que da un modelo free form, representado por los pasos enumerados en el segundo grupo.
- El segundo grupo a su vez, permite eliminar las desventajas del primer grupo, que representa al modelo complejo, es decir, no llenar el modelo semiestructurado con estructuras creadas para cada uno de los objetos a manejar.

3.4 Aplicación de la Metodología en los Contenidos Curriculares

Mediante el análisis, basado en la aplicación de la metodología propuesta en la sección 3.3, de las características de la información semiestructurada a manejar, para este caso las características descritas en la sección 2.5.1 de Contenidos Curriculares del campo de computación e informática, se inicia el desarrollo del modelo de datos semiestructurados para este tipo de información. El análisis, a través de la metodología propuesta, comprende las siguientes etapas:

3.4.1 Identificación, Manejo y Aplicación de Elementos con Estructuras Estáticas

La primera etapa de la metodología se refiere a todos aquellos elementos, dentro de la información semiestructurada del tópico de interés, que se conocen y cuya principal característica es que el formato o estructura en que se presenta la información es estática.

Durante la etapa 1 y mediante el análisis de las diversas muestras de Contenidos Curriculares, se buscan e identifican todos aquellos elementos que se conocen, por que se presentan en la mayoría de las muestras, y cuya estructura es estática, es decir, no cambió de muestra a muestra. Los elementos resultantes de la aplicación de ésta etapa son:

- *Dirección Internet:* Sitio de dónde se obtuvo el contenido curricular.
- *Autor:* Es el responsable de la información que está en el contenido. Autor contempla datos como:
    - Nombre(s) y Apellidos: paterno y materno
    - Datos de localización tales como: E-mail, teléfono, dirección, organización y página personal en Internet.
- *Revisor:* Es la persona que avala la calidad del contenido curricular. Revisor contendrá, también, los mismos datos identificados para Autor.
- *Carrera:* Para que carrera está dirigido el contenido curricular.
- *Asignatura:* Conjunto de nombres de asignaturas que pueden existir para una carrera.

La obtención y representación de información de la que se conoce su estructura se realiza a través de un documento XML, en donde la estructura de la información en XML estará controlada por un DTD, por ejemplo, para el caso de Autor, el DTD en el ejemplo 3.1 pide que un documento XML con la información del Autor tenga: NombreA y DatosAutor, en donde NombreA esta compuesto por: NombreAE, APaternoA y AMaternoA, y DatosAutor por: EMail, OarganizacionA, TelefonoA, DireccionA (información correspondiente a la ubicación física de las oficinas del autor) y PaginaA (corresponde a información de la página en Internet del autor).

```
<!ELEMENT Autor ( NombreA, DatosAutor ) >
        <!ELEMENT NombreA ( NombreAE, APaternoA, AMaternoA ) >
        <!ELEMENT NombreAE (#PCDATA) >
        <!ELEMENT APaternoA (#PCDATA) >
        <!ELEMENT AMaternoA (#PCDATA) >
        <!ELEMENT DatosAutor (EMailA, OrganizacionA, TelefonoA, DireccionA, PaginaA) >
                <!ELEMENT EMailA (#PCDATA)>
                <!ELEMENT OrganizacionA (#PCDATA)>
                <!ELEMENT TelefonoA (#PCDATA)>
                <!ELEMENT DireccionA (#PCDATA)>
                <!ELEMENT PaginaA (#PCDATA)>
```

Ejemplo 3.1: Estructura de Autor en un DTD.

El ejemplo 3.2 muestra un documento XML que describe la información de un Autor, además el documento con la información del Autor cumple con los campos, que pide el DTD del ejemplo 3.1, existan:

```
<Autor>
   <NombreA>
      <NombreAE>  Ariosto </NombreAE>
      <APaternoA> Gaona </APaternoA>
      <AMaternoA> Pliego </AMaternoA>
   </NombreA>
   <DatosAutor>
      <EMailA> A00786140@itesm.mx </EMailA>
      <OrganizacionA> ITESM </OrganizacionA>
      <TelefonoA> 82986696 </TelefonoA>
      <DireccionA> A3 - 2 </DireccionA>
      < PaginaA > http://a786140 </ PaginaA >
   </DatosAutor>
</Autor>
```

Ejemplo 3.2: Estructura de Autor en XML.

La interpretación del ejemplo 3.1 y ejemplo 3.2 es la siguiente: el DTD del ejemplo 3.1 pide la existencia de un primer elemento, etiqueta, llamado Autor, en el documento XML que describe a un Autor, por lo tanto, debe existir una etiqueta que abre *<Autor>* y una etiqueta que cierra *</Autor>*, ver ejemplo 3.2, que indica que toda la información que se encuentre entre la etiqueta que abre y la etiqueta que cierra es la información referente a Autor.

El elemento, etiqueta, Autor en el documento DTD del ejemplo 3.1 indica que dentro de ese elemento deben existir 2 etiquetas llamadas NombreA y DatosAutor, por lo tanto en el documento XML que describe a un Autor, dentro de las etiquetas de *<Autor>* y *</Autor>* deberán existir las etiquetas *<NombreA>* y *</NombreA>* que contendrán la información referente al nombre y las etiquetas *<DatosAutor>* y *</DatosAutor>* que contendrá la información de datos autor.

El elemento NombreA en el documento DTD del ejemplo 3.1, indica que deben existir, dentro de esa etiqueta, los elementos: NombreAE, APaternoA, AMaternoA, por lo tanto dentro de las etiquetas *<NombreA>* y *</NombreA>* en el documento XML deberán existir las etiquetas *<NombreAE>* y *</NombreAE>*, *<APaternoA>* y *</APaternoA>* y *<AMaternoA>* y *</AMaternoA>*. Como NombreAE, dentro del DTD del ejemplo 3.1 indica que debe existir un *#PCDATA* (valor a almacenar) en el documento XML dentro de esas etiquetas deberá existir la información a la que se refiere, para este caso, el nombre de pila del autor, que en el ejemplo 3.2 es *Ariosto*.

Una vez obtenida la información, representada en XML, ejemplo 3.2, y validada por una estructura definida en un DTD, ejemplo 3.1, la información se almacena en una tabla con una estructura que contiene los campos que se definieron en el DTD. Para el caso de Autor, la tabla que lo almacena tiene el formato mostrado en la tabla 3.1, basada en los campos definidos en el DTD del ejemplo 3.1:

ID	**Nombre**	**APaterno**	**AMaterno**	**Email**	**Organización**	**Teléfono**	**Dirección**	**Página**
1	Ariosto	Gaona	Pliego	A00786140@itesm.mx	ITESM	82986696	A3 - 2	http://a786140

Tabla 3.1: Estructura de tabla para almacenar Autor.

A la tabla 3.1 se le agregó una columna ID como identificador de la información contenida en ese registro. El ID, para este tipo de información, no es necesario especificarlo en el DTD porque se genera a partir del número de registros existentes en la tabla de Autor.

3.4.2 Identificación de Elementos sin Estructura Estática

Durante la segunda etapa de la metodología presentada, se identifican todos aquellos elementos conocidos de la información semiestructurada que podrían presentarse y que no presentan una estructura estática. Se busca, de entre las muestras de información semiestructurada a almacenar, datos presentes en la mayoría de las muestras, y se seleccionan datos cuya estructura varía entre las muestras. A tales datos se les asigna una estructura que pueda representarlos.

En la etapa 2 de la metodología, dentro de la información semiestructurada del tópico de interés, para este caso los contenidos curriculares del campo de computación e informática, los elementos que se identificaron pueden presentarse y que no mantienen una estructura estática son:

- *Contenido:* Los datos, en contenidos, están dados por un nombre, ya sea de un área, tema, subtema, etc. Los datos podrían ser un conjunto de datos anidados hasta un fondo arbitrario y en cada nivel existir elementos como: ejemplo, descripciones, bibliografía, etc.
- *Descripción:* Cada elemento dentro del contenido puede contener una breve explicación de una sección.
- *Objetivo:* Cada elemento dentro del contenido puede contener objetivos específicos.
- *Ejemplos:* Pueden existir ejemplo de diversos tipos: imagen, texto o ligas a Internet.
- *Bibliografía:* Información en libros o en Internet que apoya el estudio del contenido.

Una vez identificados los datos cuya estructura no es estática, se les define una estructura que sea capaz de representarlos.

3.4.2.1 Definición de Estructura para ***Contenido, Descripción y Objetivo***

Se define una estructura, para representar la información de *Contenido, Descripción y Objetivo* al momento de extraerse y almacenarse.

<ID, Valor, IDPertenenciaIE >
En donde:

- ID: Valor que se asignará a un elemento, ID asignado de acuerdo a su orden dentro del documento del que se obtiene. El ID empezará a asignarse a partir de 1, para el primer elemento que se encuentre en el documento. Al siguiente elemento se le asignará el número 2, y así sucesivamente. Para un elemento hijo, o anidado, dentro de otro, se le asignará no el valor que sigue al ID del elemento padre, sino el número 1 y así sucesivamente a los elementos que se encuentren al mismo nivel que el primer hijo. Un ejemplo de la aplicación de un ID es:

  Ejemplo de una sección de contenido curricular con varios niveles de anidamiento:

  Presentar un tratamiento profundo de análisis de complejidad
  Descripción: aprender a realizar análisis de complejidad
  (a) Problemas y Algoritmos
  Objetivo: Identificar los principales problemas en Algoritmos
  Complejidad en ordenamientos

  Figura 3.1: Ejemplo de Contenido con varios niveles de anidamiento.

  Representación de la Figura 3.1, asignándole un ID de acuerdo a su orden en el documento:

1 Presentar un tratamiento profundo de análisis de complejidad
1 Descripción: aprender a realizar análisis de complejidad
2 (a) Problemas y Algoritmos
1 Objetivo: Identificar los principales problemas en Algoritmos
2 Complejidad en ordenamientos

Figura 3.2: Ejemplo de Contenido con ID de anidamiento.

- Valor: Almacenará la información correspondiente al *contenido, descripción u objetivo.*
- IDPertenenciaIE: El ID de pertenencia permitirá conocer que elemento dentro del documento curricular, es el que llama a éste valor. El ID de pertenencia estará conformado por todas las concatenaciones de los elementos que lo contienen, separados por un punto el uno del otro. Por ejemplo:

  Basado en la figura 3.2, para la sección de:

1 Objetivo: Identificar los principales problemas en Algoritmos

Figura 3.3: Sección de un Contenido Curricular.

El ID de pertenencia, para la figura 3.3, es la concatenación de los IDôs de sus padres separados por un punto:

ID de pertenencia para la figura 3.3, basada en la figura 3.2, sería: **1.2**
En donde:
El ID **1** corresponde a: Presentar un tratamiento profundo de análisis de complejidad
El ID **2** corresponde a: (a) Problemas y Algoritmos

Figura 3.4: Formación de un ID de Pertenencia.

Tanto el ID como el ID de Pertenencia, definidos para esta estructura, formarán la llave primaria que identificará de forma única a la información que la estructura contenga. Para el ejemplo anterior, formado por la figura 3.3 y 3.4, la llave primaria está formada por ID 1, de la figura 3.3, y el ID de Pertenencia 1.2, mostrado en la figura 3.4.

3.4.2.2 Manejo y Aplicación de la Estructura Definida a ***Descripción***

La información de *Descripción* que se encuentre en un documento XML, deberá estar validada mediante un DTD, en donde el DTD tiene la forma de la estructura definida para Descripción en la sección 3.4.2.1, es decir el DTD validará que la estructura de la información de *Descripción* dentro del documento XML cumpla con la estructura requerida, y que también aplica para la información semiestructurada en contenidos curriculares correspondiente a *Contenido y Objetivo.*

Para *Descripción*, un documento DTD, tiene la estructura mostrada en el ejemplo 3.3:

```
<!ELEMENT Descripcion (IDDescripcion, DescripcionDE,  IDPertenencia)>
        <!ELEMENT IDDescripcion (#PCDATA)>
        <!ELEMENT DescripcionDE (#PCDATA)>
        <!ELEMENT IDPertenencia (#PCDATA)>
```

Ejemplo 3.3: Estructura de *Descripción* en un DTD.

Aplicando el DTD del ejemplo 3.3, que impone una estructura para información referente a *Descripción*, en un documento XML que describa *Descripción* se tiene:

```
<Descripcion>
        < IDDescripcion>   1 </ IDDescripcion >
        < DescripcionDE>  Aprender a realizar análisis de complejidad</ DescripcionDE >
        < IDPertenencia >  1 </ IDPertenencia >
</Descripcion>
```

Ejemplo 3.4: Estructura de *Descripción* en XML.

Como se puede observar en el ejemplo 3.3, el DTD para controlar los elementos de la información *Descripción*, en un documento XML, ejemplo 3.4, tiene la estructura definida en la sección 3.4.2.1 correspondiente a la estructura asignada al tipo de información *Descripción, Objetivo y Contenido*, en donde los campos de *Descripción*, en el DTD del ejemplo 3.3, IDPertenencia, DescripcionDE e IDPertenencia, corresponden respectivamente a los campos ID, Valor e IDPertenecia, definidos en la sección 3.4.2.1.

3.4.2.3 Manejo y Aplicación de la Estructura Definida para ***Contenido*** y ***Descripción***

Un DTD que muestre un *Contenido* y que dentro del *Contenido* permita la existencia de una *Descripción*, y que a su vez el DTD cumpla con la estructura definida para *Contenido* y *Descripción* en la sección 3.4.2.1, tiene la forma que se muestra en el DTD del ejemplo 3.5

```
<!ELEMENT Contenido (IDDocumento, Informacion, IDPertenencia,  Descripcion ) >
        <!ELEMENT IDDocumento (#PCDATA) >
        <!ELEMENT Informacion (#PCDATA) >
        <!ELEMENT IDPertenencia (#PCDATA) >
        <!ELEMENT Descripcion (IDDescripcion, DescripcionDE)>
                <!ELEMENT IDDescripcion (#PCDATA)>
                <!ELEMENT DescripcionDE (#PCDATA)>
```

Ejemplo 3.5: Estructura de Contenido y Descripción en un DTD.

Nótese que, en el ejemplo 3.5, *Descripción* parece no cumplir con la estructura que se le definió debido a que falta la sección de IDPertenencia, sin embargo, al pertenecer *Descripción* a un *Contenido*, el IDPertenencia para *Descripción* es el IDDocumento y el IDPertenecia asignado al *Contenido* al que *Descripción* pertenece. Así mismo *Contenido*, cumple con la estructura definida para ese tipo de información, pero además se le agrega

la opción de que contenido contenga *Descripción*, de lo que se desprende que podrán agregársele a cada estructura, nuevo campos surgidos de su relación con otros elementos.

Un documento XML que contenga un *Contenido* y que dentro de *Contenido* contenga una *Descripción* y que a su vez cumpla con la estructura definida en el DTD del ejemplo 3.5, se muestra en el Ejemplo 3.6.

```
<Contenido>
    <IDDocumento>  1 </IDDocumento>
    <Informacion> Presentar un tratamiento profundo de análisis de complejidad  </Informacion>
    <IDPertenencia> </IDPertenencia>
    <Descripcion>
      <IDDescripcion> 1 </IDDescripcion>
      <DescripcionDE>
                  Aprender a realizar análisis de complejidad
      </DescripcionDE>
   </Descripcion>
</Contenido>
```

Ejemplo 3.6: Estructura de *Contenido* y *Descripción* en XML.

Como se puede apreciar en el ejemplo 3.6, el documento XML describe la información correspondiente a un *Contenido* que contiene una *Descripción* y todos los elementos en el documento XML están dados por la estructura que presenta el DTD del ejemplo 3.5.

Para representar información sin estructura estática y a la que se le define una estructura y que se encuentre en un documento XML, como el documento del ejemplo 3.6, la forma de almacenamiento se define a partir de la estructura definida para el tipo de información que almacenará.

En el caso del ejemplo 3.5, que muestra la estructura para representar información de un *Contenido* que puede contener *Descripción*, la estructura de almacenamiento para *Contenido* tiene la siguiente forma:

|← -ID -→| |← ---------- Valor ----------→| |←----- ID de Pertenencia ------→|

ID	**Información**	**IDPertenencia**
1	Presentar un tratamiento profundoé	null

Tabla 3.2: Tabla de almacenamiento para Contenido.

Para el caso de la *Descripción* presente en un *Contenido*, de acuerdo al ejemplo 3.5, la tabla tiene el siguiente formato:

|← ID →| |← ---- Valor -----→| |←---------- ID de Pertenencia ----------→|

ID	**Descripción**	**ID *(De la tabla Contenido)***	**IDPertenencia *(De la tabla Contenido)***
1	Aprender a realizaré .	1	1

Tabla 3.3: Tabla de almacenamiento para Descripción.

Como se puede observar, tanto la tabla 3.2 como la tabla 3.3, mantienen la estructura definida para el tipo de información *Contenido* y *Descripción*, definido en la sección 3.4.2.1.

La cantidad de IDôs que formen el ID de Pertenencia para un elemento, estará sujeto a su relación con otros elementos que lo contengan, como es el caso de *Descripción*, en la tabla 3.3, en donde el ID de pertenencia esta dado por la llave primaria que forman el ID e ID de pertenencia en la tabla *Contenido* en la tabla 3.2.

3.4.2.4 Definición de Estructura para ***Ejemplo***

Se define una estructura para representar la información de *Ejemplo* al momento de extraerse y representarla:

<ID, Valor, Descripción, Etiqueta, TipoInformacion, IDPertenenciaIE >
En donde:

- ID: Valor que se asignará a un elemento. ID asignado de acuerdo a su orden dentro del documento del que se obtiene. La asignación del ID se realizará igual que en el ejemplo mostrado en las figura 3.1 y figura 3.2.
- Valor: Almacenará la información referente al Ejemplo.
- Descripción: Descripción, en texto, del Ejemplo
- Etiqueta: Indica que clase de ejemplo es: código, texto, gráfica, etc.
- TipoInformación: Se declara que clase de información se almacenó.
- IDPertenenciaIE: El ID de pertenencia permitirá conocer que elemento previsto, o no esperado, dentro del documento curricular, es el que llama a éste valor. El ID de Pertenencia se asignará, para cada elemento, de acuerdo al ejemplo mostrado en la figura 3.3 y figura 3.4.

Tanto el ID como el ID de Pertenencia, definidos para esta estructura, formarán la llave primaria que identificará de forma única a la información que la estructura contenga.

La estructura para *Ejemplos* se aplicará de forma similar a los elementos *Contenido y Descripción*, mostrados en la sección 3.4.2.3, para la obtención y almacenamiento de información *Ejemplos*.

3.4.2.5 Definición de Estructura para ***Bibliografía***

Se define una estructura para representar la información de *Bibliografía* al momento de extraerse y al momento de representarse, el formato de la estructura que almacena *Bibliografía* es:

<IDBibliografia, Título, Editorial, Edicion, Paginas, Fecha, Pais, OtrosDatos>
<IDAutorBibliografia, Nombre, APaterno, AMaterno, IDBibliografia>

Los datos *Bibliografías* se obtienen y almacenan usando las estructuras definidas para ese tipo de información. El manejo y aplicación de las estructuras para Bibliografía, se realizará al igual que los ejemplos mostrados para *Contenido y Descripción,* mostrados en la sección 3.4.2 descrita en este Capítulo.

3.4.3 Identificar Variaciones

La etapa 3 de la metodología se refiere a todas aquellas variaciones, específicamente en nombres e información tipo texto, que puedan presentase en la información semiestructurada a almacenar. La identificación de las variaciones existentes permite tener un mayor control sobre el manejo de la información semiestructurada que se maneje.

Al aplicar la etapa 3 en los contenidos curriculares se identifican las variaciones que pueden presentarse al momento de manejar la información en los contenidos, dichos datos son:

- *Sinónimos: Los nombres pueden variar.* Nombres asignados a un tema, por ejemplo, pueden nombrarse de diferente forma, pero referirse a lo mismo.
- *Materias previas:* Las materias previas que se debe cubrir para un contenido dirigido a una asignatura pueden variar de una a otras, además pueden presentarse en forma de código o con su nombre.

Tomando en cuenta que *Los nombres pueden variar,* se puede crear una lista que contenga sinónimos para una misma información, aplicada a materias en contenidos curriculares se tiene, por ejemplo:

Materia: Análisis de Algoritmos	**Sinónimos:** Algoritmos Análisis de Algorítmica Algorítmica

Tabla 3.4: Lista de posibles sinónimos Aplicados a una Materia.

En la tabla 3.4 el definir una lista de sinónimos para una materia, *Análisis de Algoritmos*, permite asociar y manejar diferentes nombres para esa materia, por lo tanto, durante el manejo de la información se puede hacer referencia a una materia con diferentes nombres sabiendo que se refiere a lo mismo. De esta forma se arregla el problema de variaciones en nombres.

La lista de sinónimos se tomó en cuenta al momento de diseñar el modelo de datos semiestructurado.

3.4.4 Estructura para Información no Prevista

En la etapa 4 de la sección 3.3 se hace referencia a la posibilidad de que exista información no prevista y de la cuál no se conoce la estructura que pueda tener. Para este tipo de información es para la cual se plantea una estructura que sea capaz de representarla.

La información que no fue prevista y no es básica, y de la cuál no se conoce ni su estructura ni los campos que contiene, se almacenará usando un formato que pueda anidar información, brindando información referente a cada valor que almacene la estructura. El formato propuesto, que el que se usará para información no prevista en contenidos curriculares, es el siguiente:

<ID, Tipo de Información, Valor, EtiquetaContenido, IDPertenenciaIE >
En donde:

- ID: Valor que se asignará a un elemento. ID asignado de acuerdo a su orden dentro del documento del que se obtiene. Un ejemplo de la asignación del ID se muestra en el ejemplo 3.1 y el ejemplo 3.2.
- Tipo Información: Indicará si la información es tipo texto, imagen, etc.
- Valor: Almacenará la información no prevista.
- EtiquetaContenido: La etiqueta contenido indicará la a que se refiere la información almacenada.
- IDPertenenciaIE: El ID de pertenencia permitirá conocer que elemento previsto, o no esperado, dentro del documento curricular, es el que llama a éste valor. Un ejemplo de la creación del IDPertenecia se muestra en el ejemplo 3.3 y ejemplo 3.4.

La estructura para *Información Extra* se maneja y aplica de forma similar a los elementos *Contenido* y *Descripción* para la obtención y almacenamiento de *Información Extra*, mostrados en la sección 3.4.2 de este Capítulo. Además se deja abierta la posibilidad de almacenar en este formato a los elementos básicos que no cumplen con la estructura y los campos definidos, como lo son: ejemplo, descripción, objetivos.

Tanto el ID como el ID de Pertenencia, definidos para esta estructura, formarán la llave primaria que identificará de forma única a la información que la estructura contenga.

3.4.5 Definición del Modelo de Almacenamiento

La etapa 5 de la metodología implica crear el modelo que indica la forma en que la información semiestructurada se relaciona y almacena. Para el desarrollo de esta etapa se requiere conocer y clasificar la información que puede presentarse, así como las operaciones que se desean hacer sobre la información a manejar, para lo cual es necesario desarrollas las etapas 1 a 4 de la sección 3.3 de este Capítulo.

Aplicado a Contenidos Curriculares, tomando en cuenta las etapas 1 a 2, se debe de modelar la base de datos, la etapa 3 indicará que consideraciones se deben de realizar sobre los datos que pueden presentarse en los dos primeros puntos y la estructura definida en la etapa 4 se deberá ubicar en el diseño de tal forma que cualquier elemento pueda tener acceso a el.

El desarrollo del diagrama que muestre cómo se organiza y almacena la información de los contenidos curriculares, en base a los pasos previos se pude realizar mediante el uso de herramientas como el diagrama de clases en UML (Unified Modeling Language) o Entidad-Relación.

3.4.5.1 Construcción del Modelo Usando Diagramas de Clases en UML

Para definir las relaciones entre las diferentes entidades, se utiliza la notación del diagrama de clases en UML. El diagrama de relaciones presentado en la figura 3.5, contiene las siguientes clases:

- Persona (Autor y Revisor)
- Carrera
- Asignatura
- Datos Asignatura
- Bibliografía
- Autor Bibliografía
- BibliografiaC (clase derivada de la relación Bibliografía y Contenido, ésta clase contiene las páginas a las que hace referencia una bibliografía en el contenido)
- DatosContenidos
- Contenidos, Ejemplos, Descripción, Objetivo e Información Extra

Las descripciones de las clases identificadas en la figura 3.5 para contenidos curriculares, identificando y describiendo los elementos que las componen son:

Para las clases:

Persona: Esta clase es solamente un molde que tiene el nombre, apellidos paterno y materno de una persona así como información personal. Persona puede clasificarse como Autor o Revisor.

Una Persona tiene los siguientes atributos:

- Nombre: Se refiere al nombre de pila de una persona.
- APaterno: Se refiere al apellido paterno de la persona.
- AMaterno: Se refiere al apellido materno de una persona.
- EMail: Dirección de correo electrónica personal o de trabajo.
- Organización: En que universidad, empresa u organización trabaja.
- Teléfono: Teléfono de su oficina de trabajo o lugar en que se puede contactar.

- Dirección: La dirección de trabajo, de la universidad y/u oficina dentro de la universidad.
- PáginaInternet: Página personal o de trabajo en Internet.

Autor: Se refiere al tipo de personas que estarán autorizadas a enviar información curricular a la base de datos para ser almacenada.

Un Autor tiene los siguientes atributos:
- Los heredados por la clase abstracta Persona.
- IDAutor: Identificador único que se asigna a un Autor.
- Tipo: El elemento Tipo hace diferencia entre la clase de autor, de entre aquellos que sólo son autores y aquellos que aparta de ser autores también tienen la categoría de revisores

Revisor: La clase Revisor hereda de la clase abstracta *Persona, y* se refiere a todos los individuos autorizados a certificar la calidad de los contenidos curriculares, que se conocerán como revisores.

Un Revisor tiene los siguientes atributos:
- Los heredados por la clase abstracta Persona.
- IDRevisor: Es un identificador único que identifica a una persona en específico.

Carrera: La carrera profesional indica la orientación del contenido curricular o una asignatura, esta clase permitirá tener un mayor control y una búsqueda más rápida de los contenidos orientados a ciertas asignaturas.

Una Carrera tiene los siguientes atributos:
- IDCarrera: El identificado es el número único asignado a cada Carrera específica.
- Carrera: Nombre asignado a una carrera.

Asignatura: En un contenido curricular dedicado a un carrera en específico, existen diversas asignaturas que se abordan, por lo tanto, en esta clase se pretende representar todas las asignaturas existentes en el campo de computación e informática, de esta forma cada contenido se relacionará a alguna de las asignatura existente.

Una Asignatura tiene los siguientes atributos:
- IDAsignatrua: Identificador único que refiere a una asignatura.
- Asignatura: En asignatura se indicará el nombre que tiene asignado.
- Sinónimos: Relaciona una asignatura con otros nombres que se le asignan a esa misma asignatura.

DatosAsignatura: La clase de DatosAsignatura, contendrá datos presentes en contenidos curriculares y que corresponden a datos de la asignatura a la que pertenecen, tales como: Semestre de Impartición, Materias requeridas previas a la asignatura a la que pertenece el contenido curricular a almacenar, Créditos, Número de horas prácticas y teóricas, etc.

Un DatosAsignatura tiene los siguientes atributos:

- IDDatosAsignatura: Es un identificador único asociado a un dato sobre la asignatura a la que pertenece un contenido curricular.
- Valor: Contiene el valor al que refiere una característica de la asignatura a la que pertenece un contenido curricular.
- EtiquetaAsignatura: Indica el significado del valor almacenado.

Bibliografía: La clase Bibliografía se refiere a la bibliografía que se almacenará en la base de datos y que está asociada a una asignatura o a varias asignaturas. Existe la posibilidad de que: un contenido curricular, o sus temas, o sus subtemas, etc., hagan referencia a la cita bibliográfica de un libro o a las citas bibliográficas de un grupo de libros.

Una Bibliografia tiene los siguientes atributos:

- IDBibliografia: Identificará de forma única a una bibliografía.
- Título: Refiere el título principal al que se refiere la bibliografía.
- Editorial: El nombre de la casa que imprime el libro a quien se refiere la bibliografía.
- Edición: Se refiere a la edición del libro.
- Fecha: Contiene la fecha en que se realizó el libro.
- País: El país en que fue realizado el libro.
- OtrosDatos: Cualquier otro elemento que contenga una bibliografía y que no fue previsto se registrará en el elemento OtrosDatos.

AutorBibliografía: La clase de agregación AutorBibliografía es una parte de la clase Bibliografía y también hereda de los elementos la clase abstracta Persona. La clase AutorBibliografía cotiene el nombre del autor o autores de un libro o documento que se cite en una bibliografía.

Un AutorBibliografia tiene los siguientes atributos:

- Hereda los elementos que existen en la clase abstracta Persona.
- IDAutorBibliografia: Permitirá identificar de forma única a un autor

BibliografíaC: Bibliografía que se encuentra dentro de un tema o subtema, o subsubtema, etc.

Una BibliografíaC tiene los siguientes atributos:

- IDBibDoc: Refiere a un identificador que se obtiene de acuerdo al orden en el documento y su pertenencia a otro elemento que lo anide.
- Páginas: Comprende las páginas principales a las cuales se hace referencia dentro de un libro.

DatosContenidos: La clase DatosContenidos se creó con el propósito de tener una clave que asocie a los datos de un contenido, como la asignatura, autor, revisor, etc, con la información propia del contenido, como lo son los temas, subtemas, etc.

Un DatosContenidos tiene los siguientes atributos:

- IDDatosContenido: La clave IDDatosContenido identificará de forma única a un documento enviado.
- DirInternet: Liga en Internet en la que se puede obtener el documento curricular original que se almacenó en la base de datos.

Contenidos: La información referente a los temas, subtemas, etc., que se especifican para una asignatura se almacenarán en la entidad Contenidos. En esta entidad existirá la posibilidad de almacenar información anidada hasta un fondo arbitrario.

Un Contenidos tiene los siguientes atributos:

- IDDocumento: El IDDocumento es un identificador que registra el orden de un tema dentro del contenido.
- Información: En la información se almacenará el nombre del tema, o subtema, o subsubtema, etc.

Ejemplo, Descripción, Objetivo: Cada una de las clases: Ejemplo, Descripción y Objetivo contendrán los elementos que indican su nombre. Además, al ser clases de agregación también forman parte de la clase Contenidos. Estos datos, Ejemplo o Descripción u Objetivo, podrán existir en cualquier sección de un contenido curricular enviado para ser almacenado en la base de datos.

Un Ejemplo, Descrición y Objetivo tienen los siguientes atributos:

- IDEntidad: El identificador se obtiene de acuerdo a su orden en el documento y su pertenencia a un elemento que lo anide.
- Valor*:* El *valor,* que llevará el nombre de la clase, contendrá la información que almacenará dicha clase.
- Descripción*:* Solo la clase Ejemplo contendrá este elemento, que permite almacenar una descripción o explicación del ejemplo almacenado.
- EtiquetaContenido*:* Solamente la clase Ejemplo contendrá el elemento etiqueta contenido, que permitirá definir a que se refiere el valor almacenado: código, gráfica, etc.
- TipoInformacion*:* Solamente la clase Ejemplo contendrá este atributo, que permitirá definir que tipo de ejemplo se almacenó como por ejemplo: texto, imagen, etc.

Información Extra: El elemento información extra, tiene la estructura descrita en la etapa 4, y permite almacenar toda aquella información que no se previo, de acuerdo a las intenciones del modelo planteado en este documento. El almacenamiento de un objeto se realizará como un conjunto de registros.

3.4.5.2 Lectura de las Relaciones en el Modelo de Datos en Diagramas de Clase en UML

De acuerdo a la figura 3.5, la naturaleza de las relaciones se leerán de la siguiente forma:

Persona - Autor

Un Autor *hereda* todos los atributos de la clase Persona
Una Persona *hereda* todos los atributos a una clase Autor

Persona - Revisor

Un Revisor va a *heredar* todos los atributos de la clase Persona
Una clase Persona *hereda* todos sus atributos a una clase Revisor

Autor – DatosContenidos

Un Autor *crea* ninguno, uno o más de un DatosContenidos, el cuál contiene un Contenido Curricular
Un DatosContenidos *es creado* por al menos un Autor

Revisor - DatosContenido

Un Revisor pudo haber *revisado* ninguno, uno o más de un DatosContenidos, en donde DatosContenidos tiene sólo un contenido curricular asociado a el.
Un DatosContenido, que se relaciona con un contenido curricular, debe *ser verificado* por al menos un Revisor.

Carrera – Asignatura:

En una Carrera *se imparte* al menos una Asignatura
Una Asignatura *debe ser impartida* en el menos una Carrera

Asignatura – DatosContenidos

Una Asignatura *es descrita* en ningúno, uno o más de un DatosContenidos, en donde DatosContenidos, tiene asociado un Contenido.
Un DatosContenidos, relacionado con un contenido curricular *describe* al menos una Asignatura.

Asignatura - Bibliografía

Una Asignatura *puede tener* ninguna, una o más de una Bibliografía
Una Bibliografía *es usada* por al menos una Asignatura

DatosAsignatura - DatosContenido

Un DatosAsignatura *existe* en al menos un DatosContenido
En un DatosContenido *se usan* ninguna, una o más de un DatosAsignatura

DatosContenido – Carrera

En un DatosContenido *se asocian* ninguna, una o más de una Carrera
Una Carrera *puede ser asociada* en ninguna, una o más de un DatosContenidos.

DatosContenido – Bibliografía

En un DatosContenido *se cita* ninguna, una o más de una Bibliografía
Una Bibliografía *puede ser citada* por ninguna, una o más de un DatosContenidos.

DatosContenido - Contenidos

En un DatosContenido *debe existir* un Contenidos, donde contenidos representa la información existente en un contenido curricular
Un Contenidos *esta asociado* a solamente un DatosContenidos

Bibliografía – Contenidos

Una Bibliografía *es referenciada* por ninguno, uno o más de un Contenidos
Un Contenido *cita* de ninguna, una o más de una Bibliografía

Contenidos – Contenidos

En un Contenido *se anida* de ninguno, uno o más de un Contenidos
Un contenido anidado *pertenece* a un Contenido que lo anida.

Contenidos – (Ejemplo, Descripción, Objetivo e Información Extra)

En un Contenido *pueden existir* ninguno, uno o más de un (Ejemplos, Descripción, Objetivo e Información Extra)
Un (Ejemplos, Descripción, Objetivo e Información Extra) *es parte de* un Contenido

Persona - AutorBibliografía

Un AutorBibliografia *hereda* los atributos existentes en la clase Persona
Una Persona *hereda* sus atributos a un AutorBibliografia

AutorBibliografía - Bibliografia

En una Bibliografía *existen* ninguno, uno o más de un AutorBibliografia
Un AutorBibliografia *es referenciado* en una Bibliografía

InformaciónExtra – InformaciónExtra

InformaciónExtra puede *anidar ninguna, una o más de un* InformaciónExtra
Una InformaciónExtra es *anidada* por una InformacionExtra

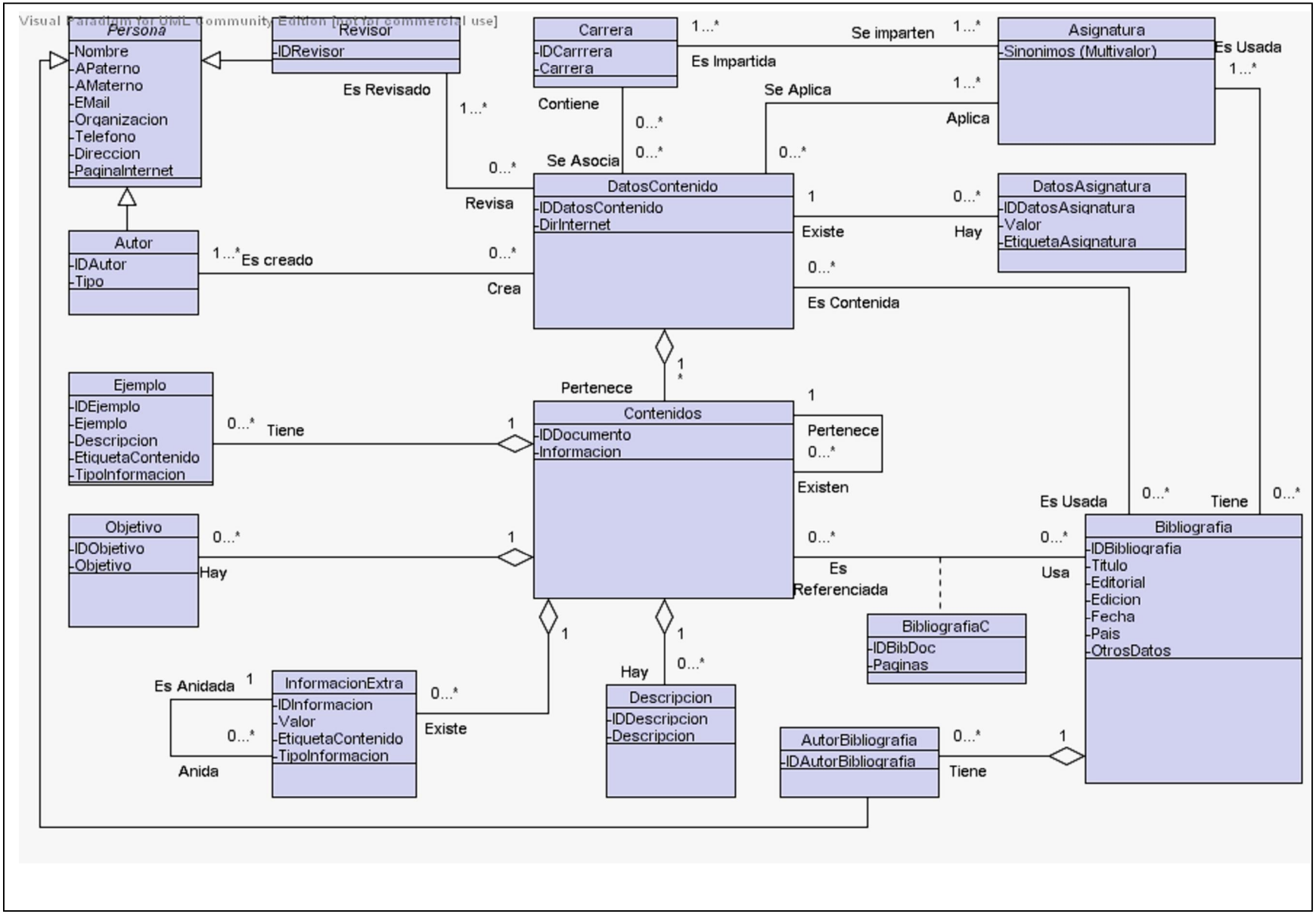


Figura 3.5: Diagrama de clases en UML

3.4.5.3 Construcción del Modelo Usando Entidad - Relación

Otra forma de modelar la estructura de almacenamiento, aparte del diagramas de clases en UML, es usando Entidad-Relación. La figura 3.6 muestra el diagrama de almacenamiento basado en Entidad Relación.

Los elementos, sus atributos, así como su relación con otros elementos en la figura 3.6 de Entidad ï Relación son similares a los elementos, atributos y relaciones identificados para el diagrama de clases en UML en la figura 3.6.

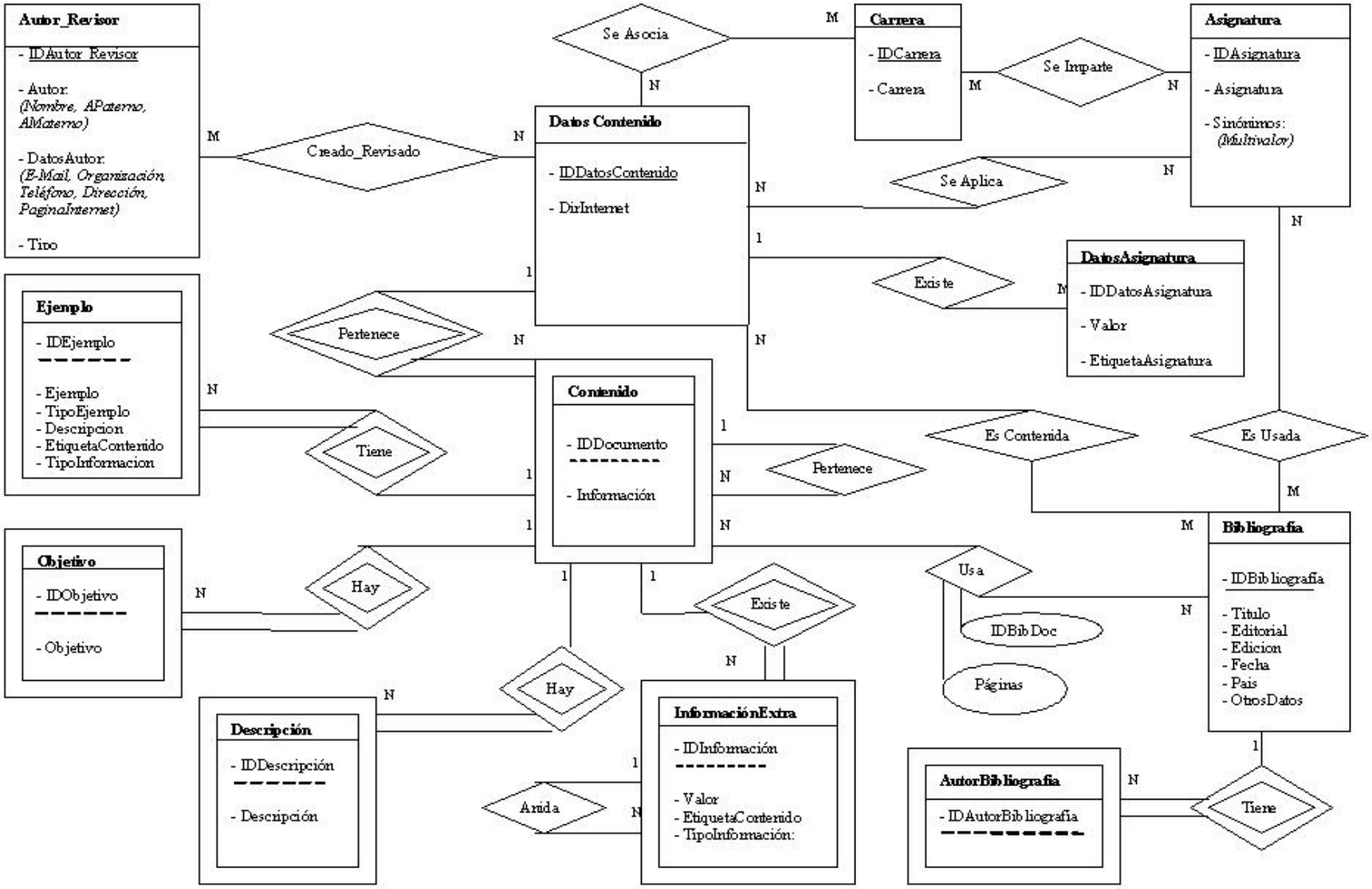


Figura 3.6: Diagrama Entidad Relación

3.4.5.4 Construcción de las Estructuras de Almacenamiento

La creación de las estructuras de almacenamiento se realiza a partir del modelo de datos semiestructurado definido con la ayuda de las herramientas UML o Entidad ï Relación definidos en este Capítulo.

Considerando el modelo y diseño realizado para almacenamiento de contenidos curriculares, figura 3.5 o figura 3.6, se obtienen las siguientes estructuras de almacenamiento:

Autor_Revisor
IDAutor_Revisor, Nombre, APaterno AMaterno, EMail, Organización, Teléfono, Dirección, PáginaInternet, IDTipo (FK), Usuario, Clave

Carrera
IDCarrera, Carrera

Asignatura
IDAsignatura, Asignatura

DatosAsignatura
IDDatosAsignatura, Valor, IDEtiquetaAsignatura (FK), IDDatosContenido(FK)

DatosContenido
IDDatosContenido, DirInternet

Contenidos
IDDatosContenido (FK), IDDocumento, Información, IDPertenencia (FK)

Ejemplo
IDEjemplo, Ejemplo, Descripción, IDEtiquetaContenido (FK), IDTipoInformacion (FK), IDDatosContenido (FK), IDDocumento (FK), IDPertenencia (FK)

Descripción
IDDescripcion, Descripcion, IDDatosContenido (FK), IDDocumento (FK), IDPertenencia (FK)

Objetivo
IDObjetivo, Objetivo, IDDatosContenido (FK), IDDocumento (FK), IDPertenencia (FK)

Bibliografía
IDBibliografia, Título, Editorial, Edicion, Fecha, Pais, OtrosDatos

AutorBibliografia
IDAutorBibliografia, Nombre, APaterno, AMaterno, IDBibliografia (FK)

InformaciónExtra
IDInformacion, Valor, IDEtiquetaContenido (FK), IDTipoInformacion (FK), IDPertenenciaIE (FK), IDDatosContenido (FK), IDDocumento (FK), IDPertenencia (FK)

TipoAutor_Revisor
IDTipo, Tipo

SinonimosAsignatura
IDAsignatura (FK), Sinonimo

TipoInformación
IDTipoInformación, TipoInformación

EtiquetaContenido
IDEtiquetaContenido, EtiquetaContenido

CreadoPor
IDAutor_Revisor (FK), IDDatosContenido (FK)

RevisadoPor
IDAutor_Revisor (FK), IDDatosContenido (FK)

Imparte
IDCarrera (FK), IDAsignatura(FK)

Aplica
IDAsignatura (FK), IDDatosContenido (FK),

EsContenida
IDDatosContenido (FK), IDBibliografia (FK)

EtiquetaAsignatura
IDEtiquetaAsignatura, EtiquetaAsignatura

EsUsada
IDAsignatura (FK), IDBibliografía (FK)

BibiografiaC
IDDatosContenido (FK), IDDocumento (FK), IDBibliografia (FK), IDPertenencia (FK), IDBibDoc, Paginas

Se Asocia
IDDatosContenido (FK), IDCarrera (FK)

Los campos subrayados indican que son una llave primaria en esa tabla, los campos con la indicación (FK) expresan la existencia de ese ID como llave primaria en alguna otra tabla y los campos subrayados con la indicación (FK) indican que son una llave primaria en esa tabla pero el ID proviene de otra tabla de donde son llaves primarias.

Con el propósito de identificar claramente la relaciones existentes entre las diferentes tablas, se muestra en la figura 3.7 sólo los campos de llaves foráneas, presentes en cada tabla, ligadas a la tabla original de donde el ID foráneo proviene, mediante un número que apunta a una tabla.

Si una tabla no contiene llaves foráneas, pero su ID es llamado por otra tabla, entonces a tal tabla sin llaves foráneas sólo se le representará con su nombre, sin campos y en negrillas, junto a un número que identifica a esa tabla, como es el caso de *Carrera.*

Además, para facilitar la identificación de las llaves foráneas dentro de las tablas en donde son llaves primarias, el nombre es el mismo. Por ejemplo: La tabla DatosAsignatura, dentro de la figura 3.7, marcada con el número 4, llama una llave foránea IDEtiquetaAsignatura, que es llave primaria en la tabla EtiquetaAsignatura marcada con el número 22, por lo tanto el IDEtiqeutaAsignatura llamado en la tabla DatosAsignatura como llave foránea, tiene el mismo nombre que en la tabla EtiquetaAsignatura.

1	**Autor_Revisor**
	IDTipo (13)

2	**Carrera**

3	**Asignatura**

4	**DatosAsignatura**	
	IDEtiquetaAsignatura (22)	IDDatosContenido (5)

5	**DatosContenido**

6	**Contenidos**	
	IDDatosContenido (5)	IDPertenencia (6)

7	**Ejemplo**				
	IDEtiquetaContenido (16)	IDTipoInformacion (15)	IDDatosContenido (5)	IDDocumento (6)	IDPertenencia (6)

8	**Descripción**		
	IDDatosContenido (5)	IDDocumento (6)	IDPertenencia (6)

9	**Objetivo**		
	IDDatosContenido (5)	IDDocumento (6)	IDPertenencia (6)

10	**Bibliografía**

11	**AutorBibliografía**
	IDBibliografia (10)

12	**InformacionExtra**					
	IDEtiquetaContenido (16)	IDTipoInformacion (15)	IDPertenenciaIE (12)	IDDatosContenido (5)	IDDocumento (6)	IDPertenencia (6)

13	**TipoAutor_Revisor**

14	**SinónimosAsignatura**
	IDAsignatura (3)

15	**TipoInformación**

16	**EtiquetaContenido**

17	**CreadoPor**	
	IDAutor_Revisor (1)	IDDatosContenido (5)

18	RevisadoPor	
	IDAutor_Revisor (1)	IDDatosContenido (5)

19	Imparte	
	IDCarrera (2)	IDAsignatura (3)

20	Aplica	
	IDAsignatura ()	IDDatosContenido (5)

21	EsContenida	
	IDDatosContenido (5)	IDBibliografia (10)

22	EtiquetaAsignatura

23	EsUsado	
	IDAsignatura (3)	IDBibliografia (10)

24	BibliografiaC			
	IDDatosContenido (5)	IDDocumento (6)	IDPertenencia (6)	IDBibliografia (10)

25	SeAsocia	
	IDDatosContenido (5)	IDCarrera (2)

Figura 3.7: Relación de llaves Foráneas en la Base de Datos

3.4.6 Definición de un DTD para la Obtención de la Información

La etapa 6 de la metodología, se refiere a la construcción de una DTD que controle la estructura de la información semiestructurada a extraer descrita en un documento XML. El DTD se realizará en base a los elementos definidos y relacionados en el modelo de almacenamiento definido en la etapa 5.

Aplicando la etapa 6 al manejo de la información de los contenidos curriculares, se desarrolla el DTD en base al modelo desarrollado en la etapa 5, el cual se muestra en la figura 3.5 o figura 3.6. Debe notarse que los diferentes elementos identificados en los modelos de la figura 3.5 y 3.6 cuentan con las estructuras definidas en etapas previas de acuerdo al tipo de información que representen.

```
<!ELEMENT Main ( DatosBasicos, Contenido* ) >

<!ELEMENT DatosBasicos (Autor+, Revisor+, (Carrera  | DirInternet | Asignatura | DatosAsignatura | Bibliografia)* ) >


 <!ELEMENT Autor ( NombreA, DatosAutor ) >
        <!ELEMENT NombreA ( NombreAE, APaternoA, AMaternoA ) >
                <!ELEMENT NombreAE (#PCDATA) >
                <!ELEMENT APaternoA (#PCDATA) >
                <!ELEMENT AMaternoA (#PCDATA) >
        <!ELEMENT DatosAutor (EMailA, OrganizacionA, TelefonoA, DireccionA, PaginaA ) >
                <!ELEMENT EMailA (#PCDATA)>
                <!ELEMENT OrganizacionA (#PCDATA)>
                <!ELEMENT TelefonoA (#PCDATA)>
                <!ELEMENT DireccionA (#PCDATA)>
                <!ELEMENT PaginaA (#PCDATA)>

<!ELEMENT Revisor ( NombreR, DatosRevisor ) >
        <!ELEMENT NombreR ( NombreRE, APaternoR, AMaternoR ) >
                <!ELEMENT NombreRE (#PCDATA) >
```

```
                <!ELEMENT APaternoR (#PCDATA) >
                <!ELEMENT AMaternoR (#PCDATA) >
        <!ELEMENT DatosRevisor (EMailR, OrganizacionR, TelefonoR, DireccionR, PaginaR ) >
                <!ELEMENT EMailR (#PCDATA)>
                <!ELEMENT OrganizacionR (#PCDATA)>
                <!ELEMENT TelefonoR (#PCDATA)>
                <!ELEMENT DireccionR (#PCDATA)>
                <!ELEMENT PaginaR (#PCDATA)>


  <!ELEMENT Carrera (#PCDATA)>
  <!ELEMENT DirInternet (#PCDATA)>
  <!ELEMENT Asignatura (#PCDATA) >
  <!ELEMENT DatosAsignatura (ValorDA, EtiquetaAsignaturaDA) >
        <!ELEMENT ValorDA (#PCDATA)>
        <!ELEMENT EtiquetaAsignaturaDA (#PCDATA)>

 <!ELEMENT Bibliografia (AutorBibB*, TituloB?, EditorialB?, EdicionB?, FechaB?, PaisB?, OtrosDatosB?)>
        <!ELEMENT AutorBibB ( NombreBE?, APaternoB?, AMaternoB?) >
                <!ELEMENT NombreBE (#PCDATA) >
                <!ELEMENT APaternoB (#PCDATA) >
                <!ELEMENT AMaternoB (#PCDATA) >
        <!ELEMENT TituloB (#PCDATA)>
        <!ELEMENT EditorialB (#PCDATA)>
        <!ELEMENT EdicionB (#PCDATA)>
        <!ELEMENT FechaB (#PCDATA)>
        <!ELEMENT PaisB (#PCDATA)>
         <!ELEMENT OtrosDatosB (#PCDATA)>

<!ELEMENT Contenido (IDDocumento, Informacion?, IDPertenencia?, (Ejemplo | Descripcion | Objetivo |
BibliografiaC | InformacionExtra)* ) >

  <!ELEMENT IDDocumento (#PCDATA) >
  <!ELEMENT Informacion (#PCDATA) >
  <!ELEMENT IDPertenencia (#PCDATA) >

  <!ELEMENT Ejemplo (IDEjemplo, EjemploE, DescripcionE?, EtiquetaContenidoE?, TipoInformacionE?)>
        <!ELEMENT IDEjemplo (#PCDATA)>
        <!ELEMENT EjemploE (#PCDATA)>
        <!ELEMENT DescripcionE (#PCDATA)>
        <!ELEMENT EtiquetaContenidoE (#PCDATA)>
        <!ELEMENT TipoInformacionE (#PCDATA)>

  <!ELEMENT Descripcion (IDDescripcion, DescripcionDE )>
        <!ELEMENT IDDescripcion (#PCDATA)>
        <!ELEMENT DescripcionDE (#PCDATA)>

  <!ELEMENT Objetivo (IDObjetivo, ObjetivoE )>
        <!ELEMENT IDObjetivo (#PCDATA)>
        <!ELEMENT ObjetivoE (#PCDATA)>

 <!ELEMENT BibliografiaC (IDBibDoc, AutorBib*, Titulo?, Editorial?, Fecha?, Edicion? , Pais?, Paginas?, OtrosDatos?)>
        <!ELEMENT IDBibDoc (#PCDATA)>
        <!ELEMENT AutorBib ( NombreE?, APaterno?, AMaterno? ) >
                <!ELEMENT NombreE (#PCDATA) >
                <!ELEMENT APaterno (#PCDATA) >
                <!ELEMENT AMaterno (#PCDATA) >
         <!ELEMENT Titulo (#PCDATA)>
        <!ELEMENT Editorial (#PCDATA)>
         <!ELEMENT Edicion (#PCDATA)>
        <!ELEMENT Fecha (#PCDATA)>
        <!ELEMENT Pais (#PCDATA)>
```

```
        <!ELEMENT Paginas (#PCDATA)>
        <!ELEMENT OtrosDatos (#PCDATA)>

<!ELEMENT InformacionExtra (IDInformacion, ValorIE, EtiquetaContenidoIE?, TipoInformacionIE, IDPertenenciaIE?)>
        <!ELEMENT IDInformacion (#PCDATA)>
        <!ELEMENT ValorIE (#PCDATA)>
         <!ELEMENT EtiquetaContenidoIE (#PCDATA)>
        <!ELEMENT TipoInformacionIE (#PCDATA)>
        <!ELEMENT IDPertenenciaIE (#PCDATA)>
```

Figura 3.8: DTD para el manejo de Contenidos Curriculares

El DTD de la figura 3.8 describe cuales son las características y estructuras, que un documento XML que describe información semiestructurada de contenidos curriculares debe, cumplir. Un ejemplo para la lectura de un DTD se realiza en la sección 3.4.1.1 de este Capítulo. Cabe destacar que un DTD permite enriqueces la semántica de un documento XML a través del uso de símbolos como:

- ***** que permite la existencia de cero o más elementos
- **?** que permite que el elemento que lo uso exista en ninguna o una ocasión.
- **,** indica el orden en que deben presentarse la información
- | símbolo que no implica la existencia obligatoria de todos los elementos que se enlistan.
- **( )** es para indicar un conjunto de elementos.

3.5 Resumen

El definir un modelo para el almacenamiento de datos semiestructurados, permite mejorar el uso que a la información se le da, siempre y cuando el modelo pueda representar y almacenar adecuadamente la información semiestructurada que almacene.

Para lograr un adecuado manejo y almacenamiento de la información, se debe iniciar con el estudio de la información semiestructurada a manejar; identificando los datos que se puede presentar y clasificándolos para definir, a cada posible tipo de datos que se presente, una estructura para su obtención y almacenamiento, siempre tomando en cuenta que podrá existir información que no se esperaba y para la cual se le deberá definir una estructura específica que pueda ser capaz de obtenerla y almacenarla.

A lo largo de este Capítulo se describió la propuesta para el estudio y análisis de la información semiestructurada, que representan los contenidos curriculares, y que finalizó en la definición de un modelo que permita un adecuado manejo y almacenamiento de este tipo de información semiestructurada.

Capítulo 4

Implementación

Este Capítulo presenta los pasos realizados para la implementación del modelo de datos semiestructurado, desarrollado en el Capítulo 3 de esta investigación y aplicado al almacenamiento de los contenidos curriculares del campo de computación e informática. La sección 4.1 muestra la descripción del modelo de datos semiestructurado ha implementar. La sección 4.2 muestra los pasos desarrollados para la implementación del modelo de datos semiestructurado desarrollado en el Capítulo 3. Finalmente el Capítulo concluye con un resumen en la sección 4.3.

4.1 El Modelo ha Implementar

El modelo que se implementó en este Capítulo (figura 3.5 ó figura 3.6), fue desarrollado siguiendo la metodología propuesta en el Capítulo 3, y está orientado al manejo de la información semiestructurada de los contenidos curriculares del campo de computación e informática, tomando en cuenta las operaciones que se desean hacer sobre la información en BECCA (Base Electrónica de Contenido Curriculares para Acreditación en Educación Superior) [20].

Como se explicó en el Capítulo 3, el modelo de datos semiestructurado resultante, de la metodología planteada en esta tesis, se encuentra entre las opciones de modelos existentes: complejo y free-form [21]. La principal ventaja es que el almacenamiento de la información se realizará en forma más ordenada y sencilla, sabiendo en donde se encuentran datos básicos que se prevén y de los cuales se conoce los campos que contienen, con lo que se logrará un mayor control y mejor uso de la información. Además de dejar abierta la posibilidad de almacenar información, que no se previó y de la cuál no se conoce su estructura, de forma correcta, simple y eficiente.

4.2 Implementación

Los elementos resultantes de la aplicación de la metodología definida en el Capítulo 3 para el manejo de la información semiestructurada son:

- Estructuras definidas para los diferentes elementos que pueden presentar durante el manejo de la información semiestructurada, incluyendo una estructura para almacenar información de la que no se conoce su estructura y no se esperaba.
- Un diagrama UML (Unified Modeling Language), figura 3.5, o Entidad ï Relación, figura 3.6, que define el modelo de almacenamiento que se usará para guardar la información semiestructurada, y cuyos campos para cada elemento están basados en las estructuras definidas para cada tipo de información.

- A partir del diagrama de almacenamiento definido se crean las tablas, ver sección 3.4.5.4, y sus diferentes relaciones con otras tablas, figura 3.7, las cuales se crean en una Base de Datos.
- Un documento DTD (Documento Type Definition) que permite definir los parámetros y estructura válida para un documento XML (Extensible Markup Language) [7] que describe un contenido curricular.

4.2.1 Herramientas para Elaboración del Prototipo

Para la implementación de los elementos obtenidos de la aplicación del método definido en el Capítulo tres, para la creación del modelo para el almacenamiento de información semiestructurada, se enlistan herramientas que se sugieren usar:

- Java como lenguaje de programación [22].
- HTML como lenguaje de presentación en la creación de interfaces.
- Uso de JSPôs para relacionar java con html.
- XML para describir la información que se almacenará validándola mediante un DTD.
- MySQL, para crear la base de datos en donde se almacenará la información [23].
- MySQL Control Center para crear e interactuar con MySQL de forma gráfica [23].
- TomCat como servidor local para la ejecución de páginas jspôs [13].
- Visual Paradigm Community Edition para diseñar el diagrama UML que definirá el modelo de almacenamiento de la información semiestructurada [9].

Todas las herramientas anteriores se eligieron por estar disponibles de forma gratuita, además de que facilitan la creación de sistemas que corren en Internet.

Para el caso de MySQL y MySQL Control Center debe verificarse que ambas herramientas corran sobre la misma versión. Es decir, si se instala una versión 5 de MySQL y se instala la versión 4.5 de MySQL Control Center, al momento de ejecutar instrucciones sobre el editor proporcionado por MySQL Control Center, podría no permitir ejecutar instrucciones tales como: ñnot inò, y sin embargo ser instrucciones válidas para ejecutar sobre la versión 5 de MySQL.

Otro aspecto que debe tomarse en cuenta, si es que no se instalan las últimas versiones de las herramientas enlistadas en esta sección, tiene que ver con la capacidad, de las herramientas, para soportarse las unas a las otras, por ejemplo: en el caso de antiguas versiones de el servidor TomCat, no se soporta la interacción con XML.

4.2.2 **Etapas de la Implementación**

La descripción del desarrollo de la implementación tiene como propósito indicar los pasos a seguir para realizar el sistema de almacenamiento de información semiestructurada, así como mostrar ejemplos del uso de las herramientas que faciliten su manejo indicando problemas que se pueden presentar y como solucionarlos. Si se desea una guía completa en el manejo de alguna de las herramientas se recomienda usar los manuales liberados por los proveedores de la herramienta en cuestión y los cuales se encuentran disponibles de forma gratuita en sus páginas en Internet.

El resultado de la aplicación de las etapas sugeridas en esta sección, es la creación de un sistema capaz de leer información semiestructurada para almacenarla en una base de datos. El sistema permitirá, por su diseño, manejar la información eficientemente de acuerdo a las necesidades expresadas durante el diseño.

Una vez que han sido instaladas las herramientas antes mencionadas para la realización de la implementación, de las cuales se recomienda usar las últimas versiones disponibles, se procede a realizar las siguientes etapas:

1. Formación de la Base de Datos
2. Definición de las Clases en Java
3. Creación de las Interfaces.
4. Aplicación del DTD en XML.

El desarrollo de cada una de las etapas identificadas para la creación de la implementación es:

4.2.2.1 **Desarrollo de la Base de Datos**

La aplicación de la etapa 1 de la sección 4.2.2, que implica la creación de la base de datos que almacenará la información semiestructurada es:

Mediante MySQL Control Center, el cual se conecta automáticamente a MySQL, se crea la base de datos de forma gráfica. El primer paso consiste en asignar un nombre a la base de datos, por ejemplo DBContenidos. El siguiente paso consiste en crear las distintas tablas con sus respectivos campos, relaciones y restricciones.

Las tablas y los campos ha crear junto con sus relaciones y restricciones, se basarán en el diagrama UML, figura 3.5, o Entidad-Relación, figura 3.6, del modelo de almacenamiento de información semiestructurada. Las tablas identificadas basadas en el modelo de datos definido para el almacenamiento de contenidos curriculares se presentan en la figura 3.7.

Al iniciar la creación de tablas, es importante empezar con aquellas que no llaman llaves foráneas, de lo contrario, si aún no es creada una tabla cuya llave primaria es

llamada como llave foránea en una tabla que se está creando, el editor marcará un error al momento de ejecutar el comando. Por lo tanto, es recomendable crear al final aquellas tablas que tengan varias llaves foráneas.

Para el caso de las tablas definidas en la sección 3.4.5.4, a partir del modelo de almacenamiento de contenidos curriculares, se sugiere crearlas en el siguiente orden:

Grupo 1:

- Carrera
- Asignatura
- DatosContenido
- Bibliografía
- TipoAutor_Revisor
- TipoInformación
- EtiquetaContenido
- EtiquetaAsignatura

Grupo 2:

- Autor_Revisor
- DatosAsignatura
- Contenidos
- AutorBibliografia
- SinonimosAsignatura
- Imparte
- Aplica
- EsContenida
- EsUsada
- SeAsocia

Grupo 3:

- Ejemplo
- Descripción
- Objetivo
- InformaciónExtra
- CreadoPor
- RevisadoPor
- BibliografíaC

Un ejemplo de creación de tablas con campos, relaciones y restricciones en MySQL usando el editor provisto por MySQL Control Center, aplicado a la tabla Contenidos definida en la sección 3.4.5.2, es:

```
#El símbolo # representa un comentario
#Declaración para la creación y asignación de nombre a una tabla
CREATE TABLE `TBL_Contenidos`(

#Declaración de los campos,  o atributos,  de la tabla
 `strIDDatosContenido`    varchar(50)      NOT NULL,
 `strIDDocumento`         varchar(50)      NOT NULL,
 `strIDPertenencia`       varchar(50)      default '',
 `strInformacion`         varchar(255)     default '',

#Declaración de llave primaria
PRIMARY KEY  (`strIDDatosContenido`, `strIDDocumento` , `strIDPertenencia` ),

#Declaración de llaves foráneas
INDEX(`strIDDatosContenido`),
FOREIGN KEY (`strIDDatosContenido`)
        REFERENCES `TBL_DatosContenido`(`strIDDatosContenido`)
        ON UPDATE CASCADE ON DELETE CASCADE

#Declaración del tipo de tabla que se crea, para este caso tipo InnoDB
) TYPE=InnoDB COMMENT='Contenidos dentro del Sistema';
```

Ejemplo 4.1: Creación de tabla en MySQL.

El formato en la declaración de tablas puede variar de acuerdo a la versión de MySQL que se use, la instrucción anterior se probó para MySQL 5.0.

4.2.2.1.1 Formato de Tablas para Información con Estructura

Cabe destacar que las tablas que se crean en la Base de Datos tienen el formato que se distinguió para cada tipo de información. En los contenidos curriculares, para el caso de información de la cuál se conoce, como Autor y Revisor, se distinguen los atributos:

Nombre	APaterno	AMaterno	EMail	Organización	Teléfono	Dirección

IDAutor_Revisor	PáginaInternet	IDTipo (FK)	Usuario	Clave

Tabla 4.1: Tabla Autor_Revisor

En dónde: Nombre, APaterno, AMaterno, EMail, Organización, Teléfono, Dirección y PáginaInternet, son los campos que se consideraron, en la sección 3.4.1 punto 1, existirían para Autor y Revisor, y los campos IDAutor_Revisor, IDTipo (FK), Usuario y Clave resultaron de consideraciones y relaciones hechas a partir de la relación de estos elementos, Autor y Revisor, con otros elementos que los usan dentro del modelo de almacenamiento, figura 3.5 o figura 3.6. De ahí se distingue la diferencia, entre la tabla 3.1 y la tabla 4.1, en donde la tabla 3.1 muestra los campos considerados para Autor sin considerar las relaciones que existirán con otros elementos, y en donde la tabla 4.1, es el resultado de los campos considerados para Autor y Revisor, junto a otros campos resultado de consideraciones hechas a partir de la relación de esta entidad con otros elementos dentro del modelo de almacenamiento en la figura 3.5 o figura 3.6

4.2.2.1.2 **Formato de Tablas para Información con Estructura Asignada**

Para los elementos dentro de los documentos curriculares que se prevén y de los cuales se asume una estructura y los campos que pueden contener, por ejemplo la tabla Contenidos, deberán estar sujetas a un formato, mencionado en la sección 3.4.2, con los siguientes campos como mínimo:

<ID, Valor, IDPertenenciaIE >
En donde:

- ID: Indica el valor que se asignará a un elemento. ID asignado de acuerdo a su orden dentro del documento del que se obtiene, ver Ejemplo 3.1 y Ejemplo 32.
- Valor: Almacenará la información en sí.
- IDPertenenciaIE: El ID de pertenencia permitirá conocer que elemento previsto, o no esperado, dentro del documento curricular, es el que llama a éste valor, ver Ejemplo 3.3 y Ejemplo 3.4.

Usando el ejemplo de la tabla contenidos, aplicándole el formato mínimo requerido se tiene:

Formato mínimo requerido:		
<ID,	Valor,	IDPertenenciaIE >
Formato de la tabla Contenidos:		
< IDDocumento	Información,	IDDatosContenido(FK), IDPertenencia (FK)>

Tabla 4.2: Formato de la tabla Contenido

En la tabla 4.2, puede observarse como el ID de Pertenencia para la tabla contenidos, esta formado por dos IDôs, los cuáles resultaron de su relación con otros elementos dentro del modelo de almacenamiento de información semiestructurada, aplicada a contenidos curriculares, desarrollado en el Capítulo 3, y mostrados en la figura 3.5 y figura 3.6. Así mismo se distingue en la creación de la tabla Contenidos, en la Ejemplo 4.1, que tanto el ID (strIDDocumento) como el ID de Pertenencia (strIDDatosContenido y strIDPertenencia) forman la llave primaria para ese tipo de Información llamada Contenidos.

4.2.2.1.3 **Formato de Tablas para Información sin Estructura**

Para la información que no fue prevista y no es básica, y de la cuál no se conoce ni su estructura ni los campos que contiene, se almacenará usando el siguiente formato, descrito también en la sección 3.4.4:

<ID, Tipo de Información, Valor, EtiquetaContenido, IDPertenenciaIE >
En donde:

- ID: Valor que se asignará a un elemento. ID Tomado a partir del orden en que se encuentra en el documento del que se obtiene, ver Ejemplo 3.1 y Ejemplo 3.2.
- Tipo Información: Indicará si la información es tipo texto, imagen, etc.
- Valor: Almacenará la información en sí.
- EtiquetaContenido: La etiqueta contenido indicará la a que se refiere la información almacenada.
- IDPertenenciaIE: El ID de pertenencia permitirá conocer que elemento previsto, o no esperado, dentro del documento curricular, es el que llama a éste valor,ver Ejemplo 3.3 y Ejemplo 3.4.

Aplicado el modelo anterior a la tabla información extra se tiene:

Formato mínimo requerido:				
<ID,	Tipo de Información,	Valor,	EtiquetaContenido,	IDPertenenciaIE >
Formato de la tabla InformaciónExtra:				
<IDInformacion,	IDTipoInformacion (FK),	Valor,	IDEtiquetaContenido (FK),	IDPertenenciaIE (FK), IDDatosContenido (FK), IDDocumento (FK), IDPertenencia (FK)

Tabla 4.3: Formato de la tabla Información Extra

Como se puede distinguir en los ejemplos anteriores, podrán crearse los IDôs y Campos extras que sean necesarios de acuerdo a su relación con otros elementos dentro del modelo de almacenamiento semiestructurado definido, para contenidos curriculares ver Figura 3.5 o Figura 3.6.

4.2.2.2 **Desarrollo de Clases**

La aplicación de la etapa 2 de la sección 4.2.2 implica la creación de las clases, en java, que se usarán para el manejo de la información al momento de almacenar, reconstruir y consultar la información semiestructurada que se encuentre en la Base de Datos.

En esta sección, las clases en java a crear, son de cuatro tipos:

1. Para almacenamiento de información.
2. Para lectura de documentos XML.
3. Para la representación de la información XML en clases.
4. De Interacción con la Base de Datos.

El desarrollo de cada tipo de clases es el siguiente:

4.2.2.2.1 Clases de Almacenamiento de Información

Las clases de almacenamiento se refieren a todos aquellos tipos de datos que se crearán a partir de las clases y atributos distinguidos en el modelo de almacenamiento de datos semiestructurado diseñado mediante la aplicación de la metodología de la sección 3.3.

Aplicando esta sección al manejo de contenidos curriculares, la creación de las clases de almacenamiento de información se realizará a partir del modelo de datos semiestructurados definido, mediante la metodología de la sección 3.3, para el manejo de contenidos curriculares y el cual es mostrado en la figura 3.5.

Un ejemplo para la creación de Clases de Almacenamiento, basado en el diagrama UML en la figura 3.5, aplicado a Autor es:

```
public class Autor
{
   //Atributos de Autor
   String strIDAutor;
   String strNombre;
   String strAPaterno;
   String strAMaterno;
   String strEMail;
   String strOrganizacion;
   String strTelefono;
   String strDireccion;
   String strPaginaInternet;
   String strTipo;

   //Constructor  que inicializa los atributos de la clase
   public Autor(){
      strIDAutor = "";
      strNombre = "";
      strAPaterno = "";
      strAMaterno = "";
      strEMail = "";
      strOrganizacion = "";
      strTelefono = "";
      strDireccion = "";
      strPaginaInternet = "";
      strTipo = "";
   }

   //Metodos Set que asignan un valor a cada uno de los atributos del objeto
   public void setIDAutor (String IDAutor){
      strIDAutor = IDAutor;
   }
   public void setNombre(String Nombre){
      strNombre = Nombre;
   }
   public void setAPaterno(String APaterno){
      strAPaterno = APaterno;
   }
```

```
    public void setAMaterno(String AMaterno){
        strAMaterno = AMaterno;
    }
    public void setEMail(String EMail){
        strEMail = EMail;
    }
    public void setOrganizacion(String Organizacion){
        strOrganizacion = Organizacion;
    }
    public void setTelefono(String Telefono){
        strTelefono = Telefono;
    }
    public void setDireccion(String Direccion){
        strDireccion = Direccion;
    }
    public void setPaginaInternet(String PaginaInternet){
        strPaginaInternet = PaginaInternet;
    }
    public void setTipo (String Tipo){
        strTipo = Tipo;
    }

    //Metodos Get que regresan el valor de cada uno de los atributos del objeto
    public String getIDAutor (){
        return strIDAutor;
    }
    public String getNombre(){
        return strNombre;
    }
    public String getAPaterno(){
        return strAPaterno;
    }
    public String getAMaterno(){
        return strAMaterno;
    }
    public String getEMail(){
        return strEMail;
    }
    public String getOrganizacion(){
        return strOrganizacion;
    }
    public String getTelefono(){
        return strTelefono;
    }
    public String getDireccion(){
        return strDireccion;
    }
    public String getPaginaInternet(){
        return strPaginaInternet;
    }
    public String getTipo(){
        return strTipo;
    }
}
```

Ejemplo 4.2: Clase en Java de Autor

Como puede observarse en el ejemplo 4.2 se muestra una clase java aplicada para el caso de Autor, en donde Autor es una clase existente en el modelo de almacenamiento de la figura 3.5. Como la clase de Autor, en el diagrama UML de la figura 3.5, hereda los atributos de la clase Persona, entonces, a los atributos de Autor se le agregan los atributos definidos para la clase Persona. Cabe destacar que es posible crear la clase que maneje la información de Autor en de otras formas, por ejemplo:

- Crear una clase en java llamada Persona, que contenga solo los atributos marcados en la figura 3.5 para Persona. Después, proceder a crear una clase en java llamada Autor que contenga sólo los atributos marcados para Autor en la figura 3.5. Una vez teniendo estas dos clases se puede heredar de Persona a Autor, mediante la instrucción *extends,* con lo cual, todos los atributos que tiene Persona los hereda Autor, esto significa que, Autor agregaría a sus atributos y métodos todos los atributos y métodos definidos en Persona.
- Otra forma en que se podría definir el manejo de Autor es creando una sola clase genérica para manejar a todas aquellas clases distinguidas como Personas, tales son los casos, de acuerdo a la figura 3.5, de Autor, Revisor y AutorBibliografia. La clase genérica podría ser creada bajo el nombre de Persona y sus atributos estarían dados por los atributos manejados por Persona, Autor, Reviso y AutorBibliografia.

Es importante notar que, al momento de crear las clases deben tomarse en cuenta la adición de atributos subyacentes de la relación de una clase con otras clases, es decir, tomando como ejemplo la clase Contenidos, de la figura 3.5, la clase en java para Contenido tiene el siguiente formato:

```
public class Contenido
{

   //Atributos de Contenido
   String strIDDatosContenido;
   String strIDDocumento;
   String strInformacion;
   String strIDPertenencia;
   Vector vObjetivo;
   Vector vDescripcion;
   Vector vEjemplo;
   Vector vBibliografia;
   Vector vInformacionExtra;

   //Constructor  que inicializa los atributos de la clase
   public Contenido(){
      strIDDatosContenido = "";
      strIDDocumento = "";
      strInformacion = "";
      strIDPertenencia = "";
      vObjetivo = new Vector();
      vDescripcion = new Vector();
      vEjemplo = new Vector();
      vBibliografia = new Vector();
```

```
    vInformacionExtra = new Vector();
}

//Metodos Set que asignan valores a los atributos de la clase
public void setIDDatosContenido(String IDDatosContenido){
    strIDDatosContenido = IDDatosContenido;
}
public void setIDDocumento(String IDDocumento){
    strIDDocumento = IDDocumento;
}
public void setInformacion(String Informacion){
    strInformacion = Informacion;
}
public void setIDPertenencia(String IDPertenencia){
    strIDPertenencia = IDPertenencia;
}
public void addObjetivo(Objetivo oObjetivo){
    vObjetivo.add(oObjetivo);
}
public void addDescripcion(Descripcion dDescripcion){
    vDescripcion.add(dDescripcion);
}
public void addEjemplo(Ejemplo eEjemplo){
    vEjemplo.add(eEjemplo);
}
public void addBibliografia(Bibliografia bBibliografia){
    vBibliografia.add(bBibliografia);
}
public void addInformacionExtra(InformacionExtra ieInformacionExtra){
    vInformacionExtra.add(ieInformacionExtra);
}

//Metodos Get que regresan los valores de la clase
public String getIDDatosContenido(){
    return strIDDatosContenido;
}
public String getIDDocumento(){
    return strIDDocumento;
}
public String getInformacion(){
    return strInformacion;
}
public String getIDPertenencia(){
    return strIDPertenencia;
}
public Vector getObjetivo(){
    return vObjetivo;
}
public Vector getDescripcion(){
    return vDescripcion;
}
public Vector getEjemplo(){
    return vEjemplo;
}
public Vector getBibliografia(){
    return vBibliografia;
```

```
    }
    public Vector getInformacionExtra(){
        return vInformacionExtra;
    }
}
```

Ejemplo 4.3: Clase en Java de Contenidos

Como puede observarse en el ejemplo 4.3 los atributos que la clase muestra son más que los definidos en la clase Contenidos en la figura 3.5, esto obedece a la relación que tiene la clase Contenido, en la figura 3.5, con otras clases como: DatosContenido, Ejemplo, Objetivo, InformacionExtra, Descripción y Bibliografía.

Cabe señalar que debido a que Contenidos puede tener varios Ejemplos, Objetivos, InformaciónExtra, Descripciones y Bibliografías, de acuerdo a la figura 3.5, los tipos de datos que almacenaran esa información, en el ejemplo 4.3, no serán directamente las clases creadas para manejar cada uno de esos tipos de información sino, vectores que contendrán objetos creados para manejar cada tipo de información, es decir, vBibliografia, en el ejemplo 4.3, es un tipo de datos Vector, el cual permite almacenar colecciones de objetos, y el método que agrega Bibliografía a la clase Contenido, addBibliografia(Bibliografia bBibliografia), espera un tipo de datos Bibliografía, con lo cuál, cada vez que se invoque el método addBibliografia, se agregará al vector vBibliografia una nueva Bibliografía.

4.2.2.2.2 Clases para Leer Documentos XML

Las clases para la lectura de documentos XML corresponden a todas aquellas clases que permiten leer y estructurar la información de un documento XML para después poder almacenar y manejar la información.

Para llevar acabo esta sección es necesario realizar la etapa previa, sección 4.2.2.2.1, debido a que las clases resultantes de la etapa previa son usadas para esta sección.

En la lectura del documento XML, la primera operación a realizar es convertir a tipo de dato *Element* toda la información en el documento. El tipo de dato *Element* permitirá distinguir un tag, ó etiqueta, y asociarle la información que contenga. Pese a que existen muchas instrucciones en XML que ayudan a estructurar y controlar aún más la información de un documento XML, para esta tesis sólo se usa Element, con el propósito de mantener los documentos simples, es decir, sin muchas restricciones e instrucciones.

La clase *Element* tiene el siguiente formato:

```
public class XMLElement
{

  //Atributos
   String uri;
```

```
   String localName;
   String qName;
   String value;

  //Constructor que inicializa los atributos de la clase
   public XMLelement(String uri, String localName, String qName)
   {
      value = "";
      this.uri = uri;
      this.localName = localName;
      this.qName = qName;
   }

   //Metodos get  que regresan los valores de cada atributo de la clase
   public String getUri()
   {
      return uri;
   }
   public String getLocalName()
   {
      return localName;
   }
   public String getQname()
   {
      return qName;
   }
   public String getValue()
   {
      return value;
   }

  //Método Set  que asigna un valor al Element
   public void setValue(String value)
   {
      this.value = value;
   }
}
```

Ejemplo 4.4: Clase en Java de Element.

El ejemplo 4.4, muestra el código de la clase en Java para Elemento (Element) llamada XMLElement. La clase almacena el nombre de un tag en la variable qName, el valor que contenga el tag, en la variable value. La variable URI (Universal Resource Identification), es un identificador único en el documento y la variable localName almacena el nombre local asignado al tag.

Una vez que se ha definido el tipo de datos Element, se crea la clase que hace del Documento XML, un conjunto de datos Element. Para este proceso existen dos opciones comunes en XML, DOM y SAX [7].

DOM (Document Object Model) genera un árbol jerárquico, en memoria, del documento o información en XML, cada elemento es considerado un nodo dentro del árbol. Este árbol jerárquico de información en memoria permite sea manipulada la

información. Por su parte, SAX (Simple API for XML) procesa el documento o información en XML de una manera muy diferente a DOM, SAX procesa la información por eventos. A diferencia de DOM que genera un árbol jerárquico en memoria, SAX procesa la información en XML conforme esta sea presentada, manipulando cada elemento en un determinado tiempo, sin incurrir en uso excesivo de memoria.

Para el manejo de los documentos XML con la información semiestructurada, en esta tesis se hace uso de SAX, debido a las ventajas que presenta al no incurrir en exceso uso de memoria. La clase SAX que lee el documento XML para convertirlo en un conjunto de Elementos tiene el siguiente formato:

```
public class SAXParserBean extends DefaultHandler implements ErrorHandler
{
   //Atributos
   private String text = "";
   private Vector vector;
   private XMLelement current;

   //Constructor que inicializa los atributos de la clase
   public SAXParserBean()
   {
      vector = new Vector();
      current = null;
   }
   //Regresa el texto leido
   public String getText()
   {
      return text;
   }
  //Regresa un Vector con la información del Documento XML, en base a Elementos
   public Vector parse(String filename)
   {

      try{
           SAXParserFactory spf = SAXParserFactory.newInstance();
           spf.setValidating(true);
           SAXParser saxParser = spf.newSAXParser();
           XMLReader reader = saxParser.getXMLReader();
           XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser");
           reader.setContentHandler(this);
           FileReader file = new FileReader(filename);
           reader.parse(new InputSource(file));
            //Regresa el vector con la información en base a Elementos
           return vector;
      }
      catch (ParserConfigurationException pce) {
         System.out.println("Error en parsing configuration: " + pce.getMessage());
         pce.printStackTrace();
      }
      //Si existe un error se regresa un vector vacio
      vector.clear();
      return vector;
   }
```

```
   //Se identifica el inicio de un Elemento
  public void startElement(String uri, String name, String qName)
   {
      current = new XMLelement(uri, name, qName);
      vector.addElement(current);
      text = new String();
   }
   //Se identifica el fin de un Elemento
   public void endElement(String uri, String name, String qName)
   {
      if(current != null && text != "")
         current.setValue(text.trim());
      current = null;
   }
   //Almacena el valor encontrado
   public void characters(char ch[], int start, int length)
   {
      if(current != null && text != "")
      {
         String value = new String(ch, start, length);
         text += value;
      }
   }
}
```

Ejemplo 4.5: Clase en Java que lee un Documento XML

La clase del ejemplo 4.5, llamada SAXParserBean, lee un documento XML y lo regresa como un conjunto de Elementos en una colección, mediante un Vector.

El código con la aplicación del ejemplo 4.4 y ejemplo 4.5 es:

```
Vector vector = new Vector();     //Vector que contendrá el documento XML en termino de Elementos
SAXParserBean saxPB = new SAXParserBean();    //Crea objeto que lee el documento XML
vector = saxPB.parse("C://Docto.XML");   //Se llama al método parse, de SAXParserBean,
                                          //que convierte  el archivo XML a un vector de Elementos.

//Si el vector no está vacío significa que no hubo error durante la lectura del documento XML
if(! vector.isEmpty()){

        //Ciclo para buscar todos los elementos en el vector con información semiestructurada
        for (int iCont = 0; iCont < vector.size(); iCont++){
                XMLelement element = (XMLelement) vector.get(iCont);   //Regresa un Elemento
                String strNombre = element.getQname();   //Regresa nombre del Tag ó Etiqueta.
                String strValor = element.getValue();     //Regresa el valor del Tag o Etiqeuta.
                System.out.println(strNombre + ñ = ò + strValor); //Imprime los valores del Elemento
        }
}


//Si el vector está vacío significa que hubo un error durante la lectura del documento XML
else{
…
}
```

Ejemplo 4.6: Código que aplica el ejemplo 4.4 y ejemplo 4.5

El ejemplo 4.6 muestra el código que lee un documento XML, en términos de Elementos e imprime el nombre de cada etiqueta y el valor que contiene.

4.2.2.2.3 Representación de Información XML en Clases

Habiendo creado las clases de almacenamiento, definidas en la sección 4.2.2.2.1, las clases para leer el documento XML, en la sección 4.2.2.2.2, y obtenido un Vector con la información del documento XML en términos de Elementos, se procede a analizar la información en el vector para almacenarla en las clases de almacenamiento.

El análisis del Vector XML, se propone se realice en base a la estructura del DTD definido para el control de la información semiestructurada en XML. Para el caso del DTD de la figura 3.8, definido para el control de la información semiestructurada de contenidos curriculares, el análisis de la información se puede dividir en 2: La información de *DatosBasicos* y la información de *Contenidos*, la división se hace para evitar la creación de una clase con muchas líneas de código que lea toda la información en el vector XML.

La clase que almacene los DatosBasicos en un documento XML que cumple con la estructura marcada por el DTD de la figura 3.8, tiene el siguiente formato:

```
public class DatosBasicos
{
   //Atributos de DatosBasicos
   String strIDDatosContenido;
   Vector vAutor;
   Vector vRevisor;
   Vector vCarrera;
   String strDirInternet;
   Vector vAsignatura;
   Vector vDatosAsignatura;
   Vector vBibliografia;

   //Constructor  que inicializa los atributos de una clase
   public DatosBasicos(){
      strIDDatosContenido = "";
      vAutor = new Vector();
      vRevisor = new Vector();
      vCarrera = new Vector();
      strDirInternet = "";
      vAsignatura = new Vector();
      vDatosAsignatura = new Vector();
      vBibliografia = new Vector();
   }

   //Metodos Set que regresa los valores de los atributos de la clase
   public void setIDDatosContenido(String IDDatosContenido){
      strIDDatosContenido = IDDatosContenido;
   }
   public void addAutor(Persona Autor){
      vAutor.add(Autor);
```

```
    }
    public void addRevisor(Persona Revisor){
        vRevisor.add(Revisor);
    }
    public void addCarrera(Carrera cCarrera){
        vCarrera.add(cCarrera);
    }
    public void setDirInternet(String DirInternet){
        strDirInternet = DirInternet;
    }
    public void addAsignatura(Asignatura aAsignatura){
        vAsignatura.add(aAsignatura);
    }
    public void addDatosAsignatura(DatosAsignatura DAsignatura){
        vDatosAsignatura.add(DAsignatura);
    }
    public void addBibliografia(Bibliografia Bib){
        vBibliografia.add(Bib);
    }

     //Metodos Get que regresan los valores de los atributos de la clase
     public String getIDDatosContenido(){
         return strIDDatosContenido;
    }
    public Vector getAutor(){
        return vAutor;
    }
    public Vector getRevisor(){
        return vRevisor;
    }
    public Vector getCarrera(){
        return vCarrera;
    }
    public String getDirInternet(){
        return strDirInternet;
    }
    public Vector getAsignatura(){
        return vAsignatura;
    }
    public Vector getDatosAsignatura(){
        return vDatosAsignatura;
    }
    public Vector getBibliografia(){
        return vBibliografia;
    }
}
```

Ejemplo 4.7: Código Java que Almacena DatosBasicos.

La clase que llenará a la clase DatosBasicos, a partir del análisis del vector con la información XML, lo hará mediante ciclos y condiciones, en base a la información que puede presentarse de acuerdo al DTD la figura 3.8, se tiene:

```
//Clase que estructura la informacion DatosBasicos
public class CreaDatosBasicos {
…
   DatosBasicos dbDBasicos;  //Tipo de Datos DatosBasicos, ver ejemplo4.7
   boolean bOk;  //Variable para detectar si existió algún error durante la ejecución

   //Metodo que Llena DatosBasicos y regresa un tipo de datos DatosBasicos
   public DatosBasicos CreaDBasicos(Vector vector){

         //Obtiene el vector con la información XML a analizar
         Vector vectorXML = vector;

         //Variables de Control
         String strNombre = ñò;
         String strValor = ñò;
         int        iCont = 0;
         boolean bBandera = false;

         try{

            //Se declara tipo de dato elemento
            XMLelement element;

            //Busca dentro del vector el inicio de DatosBasicos
            while(bBandera == false && iCont < vectorXML.size()){
                  element = (XMLelement) vectorXML.get(iCont);
                        strNombre = element.getQname();

                        if(strNombre.equals("DatosBasicos"))
                              bBandera = true;
                        else
                              iCont = iCont + 1;
                  }

            //Incrementa el contador para apuntar a DatosBasicos
            iCont = iCont + 1;

            //Ciclo que busca y extrae la información
            do{

                  //Bandera para controlar las etiquetas encontradas
                  bBandera = false;

                  //Extrae los siguientes elementos del contenido para DatosBasicos
                  element = (XMLelement) vectorXML.get(iCont);
                  strNombre = element.getQname();

                  //Analisa si el siguiente elemento en el vector es Autor
                  if(strNombre.equals("Autor")){
```

```
            //Se crea una Persona tipo Autor para almacenar los datos
            Persona pAutor = new Persona();

            //Extrae y Analiza el siguiente elemento dentro de Autor
            System.out.println("Autor");
            iCont = iCont + 1;
            element = (XMLelement) vectorXML.get(iCont);
            strNombre = element.getQname();

            //Obtiene el NombreA de Autor
            if(strNombre.equals("NombreA")){

                  iCont = iCont + 1;
                  element = (XMLelement) vectorXML.get(iCont);
                  strNombre = element.getQname();

                  //Analiza la Informacion de NombreAE dentro de Autor
                  if(strNombre.equals("NombreAE")){
                        //Se obtiene NombreAE
                         System.out.println("NombreAE");
                         strValor = element.getValue();

                         //Almaceno el nombre de pila del Autor
                         pAutor.setNombre(strValor);

                     //Incremento el contador para seguir buscando más informacion
                         iCont = iCont + 1;
                         element = (XMLelement) vectorXML.get(iCont);
                          strNombre = element.getQname();
                  }

                  //Lee APaternoA, AMaternoA
                  …
            //Llave que cierra el if NombreA
            }

            //Obtiene DatosAutor de Autor
            if(strNombre.equals("DatosAutor")){
                  //Lee OrganizacionA, TelefonoA, DireccionA, PaginaA)
                  …
            }

            //Almacena Autor dentro de DatosBasicos
            dbDBasicos.addAutor(pAutor);

            //Indica que entro a Autor y no hubo error
            bBandera = true;

//Llave que cierra el if de Autor
}

//Si el tag tiene como nombre Revisor, entra a la condición y revisa sus datos
else if(strNombre.equals("Revisor")){
            //Todos los elementos y valores existentes en revisor
            …
```

```
                }

                //Analiza Carrera, DirInternet, Asignatura, DatosAsignatura y Bibliografía
                …


            //Condicion para dejar de buscar en el vector con la información de DatosBasicos
            }while(bBandera == true  && iCont < vectorXML.size());

        //llave que cierra try
        }

        catch (Exception e){
                bOk = false;
                System.out.println(e.getMessage());
        }

        //Regresa DatosBasicos
        return dbDBasicos;

    //Llave que cierra el método CreaDBasicos
    }

//Llave que cierra la clase CreaDatosBasicos
}
```

Ejemplo 4.8: Código Java que Analiza DatosBasicos.

Cómo se puede observar en el ejemplo 4.8, el análisis de datos básicos puede realizarse mediante la ayuda de ciclos y condiciones que indiquen que elemento se esta leyendo en el vector XML, y conforme se vayan creando los objetos como: Autor, Revisor, Carrera. DirInternet, Asignatura, DatosAsignatura y Bibliografía, estos se almacenan en un tipo de datos DatosBasicos. Al final de la ejecución del código del ejemplo 4.8, se obtiene una instancia de DatosBasicos con la información referente a los DatosBasicos de un documento XML.

El mismo procedimiento para analizar datos básicos se realiza para contenidos, la búsqueda de información referente a Contenidos en un vector con información XML, se puede realizar con una serie de ciclos y condiciones que ayuden a identificar que información se esta leyendo y de esta forma ir formando los diferentes objetos considerados para Contenidos, como: Ejemplos, Objetivos, Información Extra, Descripción, Bibliografía. La única diferencia radica en que DatosBasicos sólo se presenta en una sola ocasión en un documento XML con datos de contenidos curriculares, en Cambio Contenido, puede presentarse en varias ocasiones, por lo tanto, la clase que creará Contenidos no regresaría un tipo de datos Contenido, ejemplo 4.3 , sino, un Vector con Contenidos.

4.2.2.2.4 Clases de Interacción con la Base de Datos.

Para desarrollar las clases que interactúan con la Base de Datos, es necesario haber realizado las clases que almacenan la información, definidas en la sección 4.2.2.2.1, las clases que leen los documentos XML, definidas en la sección 4.2.2.2.2, y las cuales generan un vector con la información XML en términos de Elementos y por último las clases que analizan el vector con la información XML, y la estructuran en clases, definidas en la sección 4.2.2.2.3.

Existen muchas operaciones sobre el manejo de la información en relación a las Bases de Datos. Las operaciones definidas en esta sección, son operaciones básicas a partir de las cuales se pueden realizar nuevas operaciones, de acuerdo a las necesidades particulares en el manejo de la información semiestructurada que se presenten.

Las operaciones básicas que se describen para el manejo de la información semiestructurada en la Base de Datos son:

1. Alta de Información
2. Baja de Información
3. Consultas de Información
4. Modificación de Información

El desarrollo de las operaciones es el siguiente:

4.2.2.2.4.1 Alta de Información

El alta de información semiestructurada en Base de Datos, se realizará a partir de las clases que analizan la información y la representan en términos de clases, etapa realizada en la sección 4.2.2.2.3.

Para el caso del alta de información semiestructurada de contenidos curriculares del campo de computación e informática, el alta de información se realiza de la siguiente forma:

```
public class AltaXML extends CConexion{

…

//Se ejecutan las clases que devuelven las partes del documento a analizar: DatosBasicos y
Contenidos
   public void Alta(Vector vectorXML){

        //Para el caso de CreaDatosBasicos, se crea una instancia de la clase del ejemplo 4.8.
        CreaDatosBasicos cdbCreaDBasicos = new CreaDatosBasicos();

        //CreaDatosBasicos regresará un tipo de datos DatosBasicos, para lo cual se crea una
        //instacia de la clase del ejemplo 4.7, DatosBasicos.
        DatosBasicos dbDBasicos = new DatosBasicos();
```

```
//Al  invocar el método CreaDBasicos de la clase CreaDatosBasicos, pasándole como
//paramento el vector con la información del documento XML en términos de Elementos, se
//obtiene la información correspondiente a los datos básicos del documento que se leyó.
dbDBasicos = cdbCreaDBasicos.CreaDBasicos(vectorXML);

//Variables que obtiene la informacion de DatosBasicos
Vector vAutor     =          dbDBasicos.getAutor();
Vector vRevisor   =          dbDBasicos.getRevisor();
Vector vCarrera   =          dbDBasicos.getCarrera();
String strDirInternet =      dbDBasicos.getDirInternet();
Vector vAsignatura =         dbDBasicos.getAsignatura();
Vector vDatosAsignatura =    dbDBasicos.getDatosAsignatura();
Vector vBibliografia =       dbDBasicos.getBibliografia();

//Crea la Conexión a la Base de Datos, en donde Conexion(), es un método definido en
// la clase CConexion, que hereda esta clase
Connection Con = Conexion();

Con.setAutoCommit(false);   //Permite controlar la ejecución de commits en la BD

//Se llenan las diversas tablas en la base de datos, definida en la sección 3.4.5.4
//con la información que se tiene.
//Por ejemplo: Se almacena en la Tabla TBL_CreadoPor, los autores que participan en la
//creación de un contenido
for(iCont = 0; iCont < vAutor.size(); iCont++){

    Persona pAutor = (Persona) vAutor.get(iCont);
    strInstruccion="INSERT INTO tbl_creadopor (strIDAutor_Revisor, strIDDatosContenido) ";
    strInstruccion= strInstruccion+ "strIDDatosContenido) " + "values ('" + pAutor.getIDPersona();
    strInstruccion= strInstruccion  + "','" + strIDDatosContenido + "')";

    Statement stmt = Con.createStatement();
    stmt.executeUpdate(strInstruccion); //Se ejecuta la instrucción
    stmt.close();

}

//Igual que para la tabla TBL_CreadoPor, se van llenando el resto de las tablas, de
//acuerdo a la información con la que se cuenta
…

//Después de obtener y almacenar la información referente a DatosBasicos,  se
//obtiene y almacena la información de Contenidos
CreaContenidos ccContenidos = new CreaContenidos();
Vector vContenidos = new Vector();
vContenidos = ccContenidos.CContenidos(vectorXML);

for(iCont = 0; iCont < vContenidos.size(); iCont++){

    //Se obtiene un contenido
    Contenido cContenido = (Contenido) vContenidos.get(iCont);

    //Variables que obtiene la información de un Contenido
    String strIDDocumento = cContenido.getIDDocumento();
    String strInformacion = cContenido.getInformacion();
```

```
            String strIDPertenencia = cContenido.getIDPertenencia();
            Vector vObjetivo = cContenido.getObjetivo();
            Vector vDescripcion = cContenido.getDescripcion();
            Vector vEjemplo = cContenido.getEjemplo();
            Vector vBibliografiaC = cContenido.getBibliografia();
            Vector vInformacionExtra = cContenido.getInformacionExtra();

             //Se llenan las tablas relacionadas con Contenidos, de acuerdo a la información
             //con la que se cuente

        //Llave que cierra el for que lee el vector con todos los contenidos
        }

        Con.commit(); //Al final se realiza un commit para asegurar los cambios en la base de datos
        Con.close();     //Finalmente se cierra la conexión establecida a la base de Datos
        …
    //Llave que cierra el método Alta
    }
//Llave que cierra la clase AltaXML
}
```

Ejemplo 4.9: Código Java para dar de alta información en la Base de Datos.

En el ejemplo 4.9, se muestra el código con el cual se puede dar de alta, en la Base de Datos diseñada para tal propósito en la sección 3.4.5.4, la información correspondiente a los contenidos curriculares obtenidos a partir de un documento XML.

Cabe destacar que la clase de ejemplo 4.9, hace referencia a una herencia mediante la instrucción *extend*s a la clase CConexion, a partir de la cual se puede establecer la conexión a una base de datos. La clase CConexion tiene el siguiente formato:

```
public class CConexion {

   //Método que realiza la conexión a la Base de Datos
   public static java.sql.Connection Conexion() throws Exception {

        Class.forName("org.gjt.mm.mysql.Driver").newInstance();
         return DriverManager.getConnection("jdbc:mysql://localhost:3306/DBContenidos", "root", "");

        }
}
```

Ejemplo 4.10: Código Java para Conectarse a una Base de Datos

El ejemplo 4.10 muestra una clase que se conecta a una Base de Datos creada con MySQL, la cuál lleva por nombre DBContenidos, siendo root, la clave para el acceso a la Base de Datos, es importante señalar que root es por default, la clave de acceso universal y sin restricciones de operaciones: insert, select, delete, etc., a todas las bases de datos creadas con MySql.

4.2.2.2.4.2 Baja de Información

En esta sección se explica el proceso para eliminar información semiestructurada. Se tomará como base la eliminación completa de un documento semiestructurado dado de alta en la base de datos que almacena la información semiestructurada.

Para el caso de un documento, con información semiestructurada de un contenido curricular, de acuerdo al modelo de datos semiestructurado creado para contenidos curriculares y mostrado en la figura 3.5 o figura 3.6, la eliminación de la información puede realizarse al borrar de la tabla DatosContenido el registro del documento curricular, y esto sólo será posible si:

- Al crear la tabla DatosContenido, se identificó al IDDatosContenido, como llave primaria mediante la instrucción: PRIMARY KEY(IDDatosContenido), y
- Si al ser usada IDDatosContenido en las tablas con las que se relacionan, se le declara como llave foránea mediante la instrucción:

  FOREIGN KEY(`IDDatosContenido`)
  REFERENCES `TBL_DatosContenido`(` IDDatosContenido`)
  ON UPDATE CASCADE ON DELETE CASCADE

- En donde ON UPDATE CASCADE ON DELETE CASCADE, indica que al ser eliminada en la tabla origen, en este caso la llave IDDatosContenido de la tabla DatosContenido, todos los registros de las tablas que declararon IDDatosContenido como FOREIGN KEY serán eliminados automáticamente.

Si las tablas en la base de datos fueron creadas con las características descritas en el párrafo anterior, la eliminación de un contenido se realizaría de la siguiente forma:

```
DELETE FROM TBL_DatosContenido WHERE IDDatosContenido = 1
```

Ejemplo 4.11: Instrucción SQL para Eliminar un Registro.

El ejemplo 4.11 muestra una instrucción SQL que elimina de la tabla TBL_DatosContenido, el registro cuyo IDDatosContenido tiene el valor de 1, con lo cual, todos aquellos registros que llamen el IDDatosContenido con valor 1, en cualquier tabla de la base de datos, serán eliminados.

De no haber sido creadas las tablas de la base de datos usando las instrucciones descritas en esta sección, se tendrá que aplicar el ejemplo 4.1 para cada tabla que almacene en la base de datos la información de un contenido curricular específico.

4.2.2.2.4.3 Consultas

Las consultas sobre la información semiestructurada almacenada en la base de datos, creada para tal propósito, pueden ser definidas a partir del modelo de datos semiestructurados que muestra como la información está organizada y relacionada.

Algunas de las consultas que pueden distinguirse a partir del modelo de datos semiestructurados, creado para el manejo de la información semiestructurada de los contenidos curriculares del campo de computación e informática y mostrado en la figura 3.5 o figura 3.6, son:

- Búsqueda de contenidos por cada entidad o clase descrita en el modelo: Autor, Revisor, Carrera, Asignatura, DatosAsignatura, Ejemplo, InformaciónExtra, Objetivo, Descripción, Bibliografía, AutorBibliografia, Contenidos y DatosContenido.
- Búsqueda e Integración de contenidos a partir de las relaciones existentes entre las entidades, como las descritas en las identificadas en la sección 3.4.5.2.

Un ejemplo de reconstrucción de un documento curricular almacenado en la base de datos, presentando cada elemento del contenido en el mismo orden que mostraban en el documento XML original del que se obtuvo es:

```
//Clase que reconstruye los DatosBasicos, ver ejemplo 4.7, donde,
//DatosBasicos refiere a la información: Autor, Revisor, Carrera, DirreccionInternet, Asignatura,
//DatosAsignatura y Bibliografía.
public class ConsultaDBasicos  extends CConexion{
…
   //Método que reconstruye los datos basicos de acuerdo al IDDatosContenido ingresado
   public DatosBasicos ConsultaDBxIDContenido(String IDDatosContenido){

        //Se crea un tipo de datos DatosBasicos que contendrá la información de los datos básicos
        //Correspondientes al contenido cuyo id es IDDatosContenido
        DatosBasicos dbDatosBasicos = new DatosBasicos();
        dbDatosBasicos.setIDDatosContenido(IDDatosContenido);

        Con = Conexion();          //Crea la conexión a BD
        Con.setAutoCommit(false);          //Permite controlar la ejecución de commits en la BD

        //Lee todos los autores que crearon un contenido
        String strInstruccion = "";
        strInstruccion = "Select strIDAutor_Revisor, strNombre, strAPaterno, strAMaterno,
                          strEMail, strOrganizacion, strTelefono, strDireccion, strPaginaInternet
                          from tbl_Autor_Revisor where strIDAutor_Revisor in
                           (Select strIDAutor_Revisor From TBL_CreadoPor where
                          strIDDatosContenido = ?) order by length(strIDAutor_Revisor),
                          strIDAutor_Revisor asc  ";
        pstmt = Con.prepareStatement(strInstruccion);
        pstmt.setString(1, dbDatosBasicos.getIDDatosContenido());
        rset = pstmt.executeQuery();
```

```
        //Ciclo que lee todos los autores que participan en un contenido
        while( rset.next() ){

                Persona pAutor =  new Persona();
                pAutor.setIDPersona(rset.getString("strIDAutor_Revisor"));
                pAutor.setNombre(rset.getString("strNombre"));
                pAutor.setAPaterno(rset.getString("strAPaterno"));
                pAutor.setAMaterno(rset.getString("strAMaterno"));
                pAutor.setEMail(rset.getString("strEMail"));
                pAutor.setOrganizacion(rset.getString("strOrganizacion"));
                pAutor.setTelefono(rset.getString("strTelefono"));
                pAutor.setDireccion(rset.getString("strDireccion"));
                pAutor.setPaginaInternet(rset.getString("strPaginaInternet"));

                //Almacena el autor del contenido
                dbDatosBasicos.addAutor(pAutor);
        }


        pstmt.close();
        rset.close();

        //Lee todos los revisores, Carrera, DirreccionInternet, Asignatura, DatosAsignatura y
        //Bibliografía que crearon un contenido y se almacenan en la instancia del tipo de datos
        //DatosBasicos, que en esta clase es llamada dbDatosBasicos.
        …

        Con.commit(); //Al final se realiza un commit para asegurar los cambios en la base de datos
        Con.close();    //Finalmente se cierra la conexión establecida a la base de Datos

        //Al final se devuelve la instancia de la clase DatosBasicos con la información del contenido
        // cuyo IDDatosContenido se ingresó como parámetro de búsqueda
        return dbDatosBasicos;

    //Llave que cierra el método ConsultaDBxIDContenido
    }
//Llave que cierra la clase ConsultaDBasicos
}
```

Ejemplo 4.12: Código Java que obtiene los DatosBasicos de un Contenido.

El ejemplo 4.12 muestra la clase que reconstruye la información definida como parte de DatosBasicos, de un contenido curricular. La sección de los Contenidos del documento curricular se reconstruiría de forma similar a DatosBasicos, destacando que en Contenidos los Identificadores: IDDocumento e IDPertenencia en la entidad Contenidos, IDEjemplo en la entidad Ejemplo, IDObjetivo en la entidad Objetivo, IDInformación en InformacionExtra, IDDescripcion en Descripcion e IDBibDoc con respecto a Bibliografía, indican el orden de los elementos en el documento curricular. Con lo cuál al momento de extrar y presentar la información almacenada en la base de datos, mediante el análisis de los ID´s, se determina la posición de cada elemento dentro del doumento.

4.2.2.2.4.4 Modificaciones

Los tipos de modificaciones a la información semiestructurada almacenada en la base de datos pueden distinguirse como:

1. Cambios en los valores de los datos: Los cambios en los valores de los datos refieren a modificar o actualizar, la información que ya está en un registro de la base de datos. Por ejemplo: En la tabla TBL_Autor_Revisor, en la columna PáginaInternet, el cambio de valor contempla modificar la información contenida en ese campo.
2. Cambios en la estructura de la información: Los cambios de la estructura implica agregar Elementos, como por ejemplo: Comontenidos, Ejemplos, Descriciones, etc, para el caso de contenidos curriculares del campo de computación e informática.

Para el caso del punto 2 de esta sección. Los cambios en la estructuctura pueden realizarse como sigue:

- Eliminar el documento almacenado en la base de datos a modificar y agregar los nuevos elementos al documento original, creando un nuedo documento XML, a partir del nuevo documento original, para leerlo y almacenarlo nuevamente en la base de datos.
- Otra opción es, agregar los nuevos elementos al final del nivel que se quiera modificar, es decir, si en la base de datos se quiere agregar un ejemplo a un contenido, el ejemplo estaría colocado al final de todos los elementos presentes en ese contenido. Esto con la finalidad de no modificar los IDôs e IDôs de Pertenencia de los elementos ya existentes, ya que ingresar elemento entre los elementos ya presentes implicaría un complicado proceso de modificación de IDôs y movimientos en la Base de Datos.

4.2.2.3 Creación de Interfaces

Para crear las interfaces que sirvan de enlace entre la información semiestructurada almacenada en la base de datos y el usuario de la información, se puede realizar usando herramientas como JSP (Java Server Pages) [5], que es un conjunto de tecnologías que permite la generación dinámica de páginas Web combinando código java con lenguaje de marcas como HTML para generar el contenido de la página.

Como parte de la familia de la tecnología Java, con JSP se pueden desarrollar aplicaciones Web independientes de la plataforma. Una característica importante es que permite separar la interfaz del usuario de la generación del contenido dinámico dando lugar a procesos de desarrollo más rápidos y eficientes. Adicionalmente, puede acceder directamente a componentes Java Beans, instanciándolos y estableciendo sus propiedades e invocando sus métodos directamente desde la página JSP. Esto permite desarrollar

aplicaciones n-capas donde se separan los datos, la lógica del los datos y la lógica de presentación, encapsulando mediante Beans el acceso a los datos [5].

El desarrollo de una página JSP que invoca clases Java mediante Beans, para generar una página que presente los datos existentes en la base de datos, implican los siguientes pasos:

1. Desarrollo de la clase que regresa de la base de datos, la información que se desplegará en la página.
2. Desarrollo de la página JSP que despliega la información.

El desarrollo de cada paso enlistado es:

4.2.2.3.1 Desarrollo de la Clase que Regresa la Información

Un ejemplo de clase que regresa información, clase de tipo consulta como las definidas en la sección 4.2.2.2.4.3, aplicada a la información existente en la base de datos definida en la sección 3.4.5.4, para el caso de Autores existentes es:

```
//Clase que realiza consulta de DatosBasicos (Autor, Revisor, Carrera, DirreccionInternet,
//Asignatura, DatosAsignatura y Bibliografía.)

package opexml;              //La clase se empaqueta en la carpeta opexml

public class ConsultaDBasicos  extends CConexion{
…
//Método que extrae los autores existentes en la BD
   public Vector ConsultaAutores(){

        Con = Conexion();           //Crea la conexión a BD
        Con.setAutoCommit(false);   //Permite controlar la ejecución de commits en la BD

        String strInstruccion = "Select * From TBL_Autor_Revisor Where strIDTipo = ? ";
        pstmt = Con.prepareStatement(strInstruccion);
        pstmt.setString(1, "1");    //El valor 1, para este caso, indica que solo las personas en
                                    //la tabla  TBL_Autor_Revisor, cuyo IDTipo es 1, son los autores.

        rset = pstmt.executeQuery();    //Ejecuta la consulta strInstruccion

        while( rset.next() ){   //Ciclo que regresa todos los autores existentes en la base de datos

                //La información se agrupara, almacenará, en tipos de datos Persona,
                // cuya estructura es igual a la definida en el ejemplo 4.2
                Persona pAutor =  new Persona();
                pAutor.setIDPersona(rset.getString("strIDAutor_Revisor"));
                pAutor.setNombre(rset.getString("strNombre"));
                pAutor.setAPaterno(rset.getString("strAPaterno"));
                pAutor.setAMaterno(rset.getString("strAMaterno"));
                pAutor.setEMail(rset.getString("strEMail"));
                pAutor.setOrganizacion(rset.getString("strOrganizacion"));
                pAutor.setTelefono(rset.getString("strTelefono"));
                pAutor.setDireccion(rset.getString("strDireccion"));
```

```
                pAutor.setPaginaInternet(rset.getString("strPaginaInternet"));

                //Almacena, en un Vector, a cada persona
                // identificada como autor en la base de datos
                vAutor.add(pAutor);
        }

        //Al final, el método envia el vector con  la iformación de todos los autores
        //existentes en la base de datos.
        return vAutor;

        …
    //Llave que cierra la clase que consulta el método ConsultaAutores
    }
//Llave que cierra la clase que consulta la clase ConsultaDBasicos
}
```

Ejemplo 4.13: Código Java que obtiene Autores.

El ejemplo 4.13 consulta la base de datos definida en la sección 3.4.5.4, y regresa todos los autores existentes en la tabla TBL_Autor_Revisor. La clase regresa a los autores mediante un Vector, el cuál contiene la información de los autores, en términos del tipo de dato Persona, y la cual tiene una estructura similar a la definida en el ejemplo 4.2.

4.2.2.3.2 Desarrollo de Páginas JSP

La construcción de un JSP, debe entenderse como la mezcla de código java con HTML, en donde el código java se escribirá entre las marcas <% *código java* %>. El código JSP que invoca una clase Java mediant un Bean para desplegar información proveniente de la base de datos es:

```
//Llama las librerias de los tipos de datos que se usan
//Para el caso opexml.* se refiere a la carpeta en donde estan los tipos de datos creados,
//como es el caso de Persona.
<%@ page import="opexml.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.util.Vector" %>
<%@ page import="java.lang.*" %>
<%@ page import="java.lang.String" %>
<%@ page import="java.lang.Integer" %>

<html>

<head>
<title> Registros de Autores</title>
</head>

<body>
<form name="frmRegistrosAutor" method="post" >

//La información se mostrará usando una tabla
//Las columnas que se mostrarán son:
```

```
<table>
 <tr>
  <th>  Nombre  </th>
  <th>  Organizacion  </th>
  <th>  Email  </th>
  <th>  Teléfono </th>
  <th>  Dirección </th>
  <th>  Página Internet </th>
 </tr>

 //Se declara un Bean que llama la clase ConsultaDBasicos, ver ejemplo 4.13, la cual
 //esta en la carpeta opexml, y se le asigna un ID local llamado, para este caso, ConDBA
 <jsp:useBean id="ConDBA" class="opexml.ConsultaDBasicos" />
 <jsp:setProperty name="ConDBA" property="*" />

 //Código java
 <%
        //Se ejecuta el método que regresa los autores existentes, en la base de datos, mediante la
        //la clase ConsultaDBasicos con el alias ConDBA
        Vector vAutor = ConDBA.ConsultaAutores();

        //Se leen todos los Autores que se regresaron en un Vector,
        //al ejecutar el método ConsultaAutores
        for(int iCont = 0; iCont < vAutor.size(); iCont++)
        {
            //Cada obejeto existente en el Vector se parsea a un tipo de datos Persona
            Persona pAutor = (Persona) vAutor.get(iCont);
 %>
  //Para cada autor obtenido del vector, se adhiere a la tabla, que despliega la información,
  // un renglon, en donde pra cada columna del renglón, el valor es:
  <tr>
   //La instrucción PRE permite mostras la información sin eliminar los espacios en blanco que
   //pueda tener la información en su forma original.
   <td><pre>
        <%=pAutor.getNombre() + " " + pAutor.getAPaterno() + " " + pAutor.getAMaterno()%>
   </pre></td>
   <td> <pre> <%=pAutor.getOrganizacion()%>     </pre> </td>
   <td> <pre> <%=pAutor.getEMail()%>            </pre> </td>
   <td> <pre> <%=pAutor.getTelefono()%>         </pre> </td>
   <td> <pre> <%=pAutor.getDireccion()%>        </pre> </td>
   <td> </pre> <%=pAutor.getPaginaInternet()%>  </pre> </td>
  </tr>
 //Cierra el cliclo que lee todos los autores obtenidos de la base
 <%
        }
 %>

</table>
</form>
</body>
</html>
```

Ejemplo 4.14: JSP que despliega la información de Autores.

El ejemplo 4.14 muestra el código de un JSP, el cual despliega la información de la base de datos, diseñada en la sección 3.4.5.4, para los datos de autores existentes en la tabla TBL_Autor_Revisor. El ejemplo 4.14, invoca la clase de consulta del ejemplo 4.13, mediante un bean. El JSP presentaría la información de Autor como sigue:

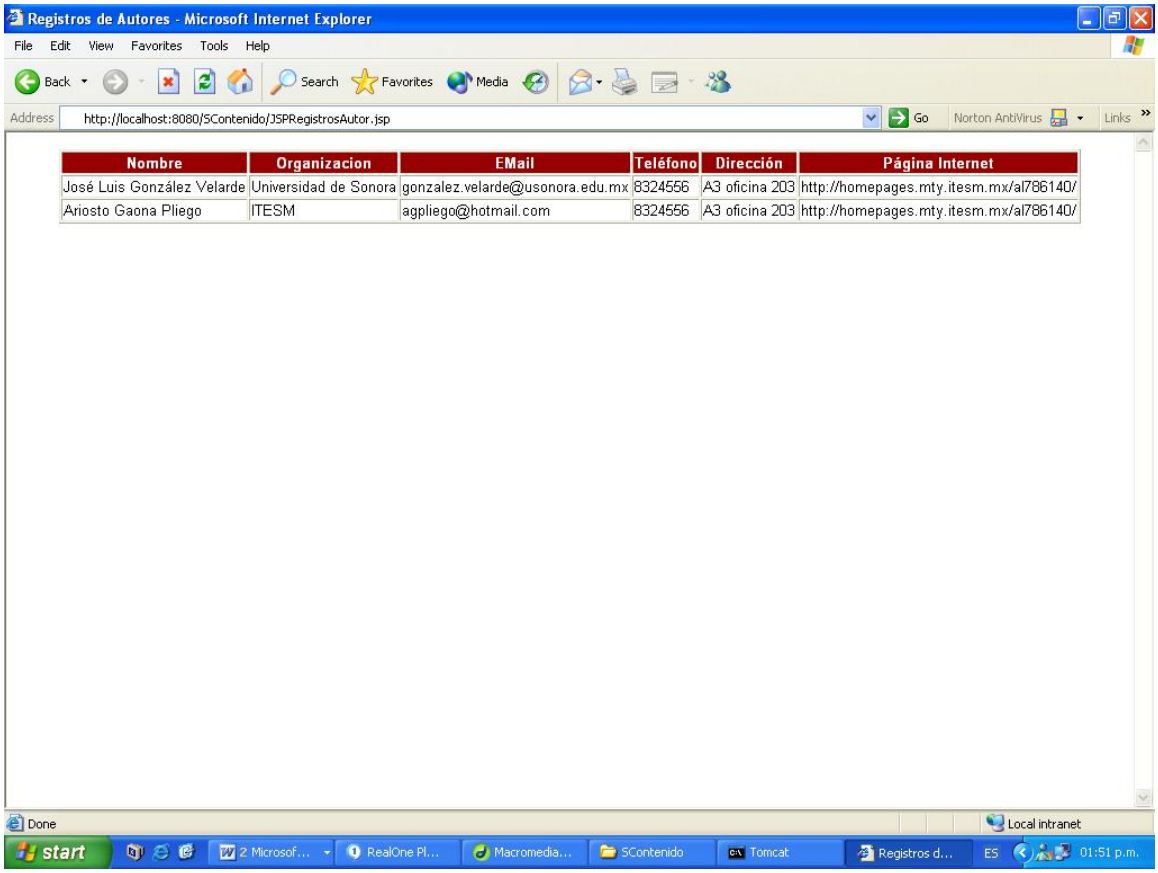


Figura 4.1: Desplegado de página JSP.

Es importante mencionar que las clases que se creen para interactuar con la página JSP, deberán encontrarse compiladas dentro de la carpete que contega el sistema, por ejemplo:

TOMCAT_HOME/webapps/SContenido/WEB-INF/classes/opexml/misclases.class

Dónde:

- *TOMCAT_HOME*: es la dirección en que se instaló el servidor.
- *webapps*: es la carpeta que existe por default, en la cuál el servidor busca los sistemas para desplegar.
- SContenido, es el nombre de la carpeta y que se le asignó al sistema que despliega la información de contenidos curriculares.
- *WEB-INF*: Es la carpeta que debe existir en toda carpeta que contenga un sistema, y en la cuál se almacenan las clases de java.

- *classes:*es la carpeta en la que se crean los paquetes de clases, y debe existir necesariamente dentro de WEB-INF.
- *opexml:* es el nombre que se asignaó al paquete que contiene las clases que se invocan en el JSP.

Para llamar un JSP se debe iniciar el servidor y después escribir en la barra de dirección, en un navegador de Internet como Netscape o Internet Explorre, la dirección:

http://localhost:8080/SContenido/JSPRegistroAutor.jsp

Donde:
- *localhost:8080*: Es la máquina y puerto, local en este caso, en que el servidor corre.
- *SContenido*: Es la carpeta en que están los archivos jsp y class, del sistema.
- *JSPRegistroAutor.jsp*: Es el nombre del jsp que despliega la información.

4.2.2.4 Aplicación del DTD en XML

La etapa 4 de la sección 4.2.2, implica la aplicación del documento DTD en un Documento XML para validar el contenido y la estructura de la información descrita en XML.

Para validar un documento XML, con el DTD establecido, se debe colocar la siguiente instrucción al inicio del documento XML que se leerá:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Main SYSTEM "Contenido.dtd">
```

La instrucción anterior llama a *Contenido.dtd,* que es el archivo que indica cuál debe ser el formato con el que el documento XML debe cumplir. El archivo DTD puede ubicarse en el directorio <TomCat_Home>bin/, para ser invocado por el servidor, en donde TomCat_Home refiere al path donde fue instalado previamente el servidor. De acuerdo a la versión del servidor, puede requerirse colocar el archivo DTD en otro directorio diferente.

Un documento XML que aplica y valida su información usando el DTD definido en la figura 3.8 es:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Main SYSTEM "Contenido.dtd">
<Main>
  <DatosBasicos>
     <Autor>
       <NombreA>
         <NombreAE>José Luis</NombreAE>
         <APaternoA> González </APaternoA>
         <AMaternoA> Velarde </AMaternoA>
       </NombreA>
```

```
    <DatosAutor>
      <EMailA>gonzalez.velarde@usonora.edu.mx </EMailA>
      <OrganizacionA> Universidad de Sonora </OrganizacionA>
      <TelefonoA> 8324556 </TelefonoA>
      <DireccionA> A3 oficina 203 </DireccionA>
      <PaginaA> http://homepages.mty.itesm.mx/al786140/</PaginaA>
     </DatosAutor>
  </Autor>

  <Revisor>
      <NombreR>
        <NombreRE> Alejandro </NombreRE>
        <APaternoR> Sanchez </APaternoR>
        <AMaternoR> Silva </AMaternoR>
      </NombreR>
      <DatosRevisor>
        <EMailR>asilva@usonora.edu.mx </EMailR>
        <OrganizacionR> Universidad de Montemorelos </OrganizacionR>
        <TelefonoR> 8332643 </TelefonoR>
        <DireccionR> FIT A8 </DireccionR>
         <PaginaR> http://homepages.mty.itesm.mx/al784830/ </PaginaR>
     </DatosRevisor>
  </Revisor>

  <Revisor>
      <NombreR>
        <NombreRE> ariosto </NombreRE>
        <APaternoR> Gaona </APaternoR>
        <AMaternoR> Pliego </AMaternoR>
      </NombreR>
      <DatosRevisor>
         <EMailR>agpliego@hotmail.com </EMailR>
         <OrganizacionR> ITESM </OrganizacionR>
         <TelefonoR> 8324556 </TelefonoR>
         <DireccionR> A3 oficina 203 </DireccionR>
          <PaginaR> http://homepages.mty.itesm.mx/al786140/</PaginaR>
     </DatosRevisor>
  </Revisor>

  <Carrera> licenciatura en ciencias computacionales </Carrera>
  <DirInternet> http://infocbi.uam.mx/licenciatura/computacion/212353.html  </DirInternet>
  <Asignatura> Sistemas Operativos 1 </Asignatura>
</DatosBasicos>

<Contenido>
  <IDDocumento> 1 </IDDocumento>
  <Objetivo>
      <IDObjetivo> 1 </IDObjetivo>
      <ObjetivoE>
          Al finalizar el curso el alumno podrá conocer y utilizarlas funciones y técnicas Básicas  de sistemas
          operativos con orientación monousuaria para optimizar o sugerir mejoras en el diseño de sistemas propios y
          Se percibirá en forma introductoria aspectos relevantes a seguridad de sistemas operativos multiusuarios
  </ObjetivoE>
  </Objetivo>

  <InformacionExtra>
      <IDInformacion> 2 </IDInformacion>
      <ValorIE> MultiProcesador  Linux</ValorIE>
      <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
      <TipoInformacionIE> Texto </TipoInformacionIE>
   </InformacionExtra>

   <InformacionExtra>
      <IDInformacion> 1 </IDInformacion>
      <ValorIE> Si </ValorIE>
      <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
      <TipoInformacionIE> Texto </TipoInformacionIE>
      <IDPertenenciaIE> 2 </IDPertenenciaIE>
   </InformacionExtra>
```

```
    <InformacionExtra>
       <IDInformacion> 3 </IDInformacion>
       <ValorIE> MultiProcesador  Windows</ValorIE>
       <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
       <TipoInformacionIE> Texto </TipoInformacionIE>
    </InformacionExtra>

    <InformacionExtra>
       <IDInformacion> 1 </IDInformacion>
       <ValorIE> No </ValorIE>
       <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
       <TipoInformacionIE> Texto </TipoInformacionIE>
       <IDPertenenciaIE> 3 </IDPertenenciaIE>
    </InformacionExtra>

    <InformacionExtra>
       <IDInformacion> 4 </IDInformacion>
       <ValorIE> MultiUsuario  Linux</ValorIE>
       <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
       <TipoInformacionIE> Texto </TipoInformacionIE>
    </InformacionExtra>

    <InformacionExtra>
       <IDInformacion> 1</IDInformacion>
       <ValorIE> Si </ValorIE>
       <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
       <TipoInformacionIE> Texto </TipoInformacionIE>
       <IDPertenenciaIE> 4 </IDPertenenciaIE>
    </InformacionExtra>

    <InformacionExtra>
       <IDInformacion> 5 </IDInformacion>
       <ValorIE> MultiUsuario  Windows</ValorIE>
       <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
       <TipoInformacionIE> Texto </TipoInformacionIE>
    </InformacionExtra>

    <InformacionExtra>
       <IDInformacion> 1 </IDInformacion>
       <ValorIE>      <![CDATA[
                                  Diagram:
            AGP2
         Pruba 2 <------------ Prueba 1
           |          <3>       ^
           | <1>                |  <1> = baja
           V                    |  <2> = der
         PRB 3----------------------+   <3> = izq
                    <2>
                         ]]> </ValorIE>
       <EtiquetaContenidoIE> Ejemplo </EtiquetaContenidoIE>
       <TipoInformacionIE> Texto </TipoInformacionIE>
       <IDPertenenciaIE> 5 </IDPertenenciaIE>
    </InformacionExtra>
  </Contenido>
</Main>
```

Ejemplo 4.15: Documento XML que aplica un DTD.

En el ejemplo 4.15 se muestra un documento XML que cumple con las especificación de estructura definida es el DTD de la figura 3.8. El documento describe la información de un contenido curricular del campo de computación e informática, aplicada a la carrera de licenciatura de ciencias computacionales, aplicada a la carrera de Sistemas Operativos, describiendo objetivos e Información Extra.

Nótese que en el último segmento de información extra, en el ejemplo 4.15, el valor está dado entre las instrucciones: <![CDATA[*valor de la información*]]>, estas

instrucciones permiten la existencia de cualquier carácter en XML. Si la instrucción anterior, elementos en la información que se contiene como: <, >, <>, </>, entre otros, provocarian que no se reconozcan como parte de la información, sino como parte de los tags definidos en el documento XML.

4.3 Resumen

En este Capítulo se describió el proceso para la creación de una aplicación que maneje la información semiestructurada. Se implantó el modelo de datos semiestructurado definido en el Capítulo 3, para el caso de los contenidos curriculares de computación e informática, apoyándose de las herramientas, surgidas también, de la aplicación de la metodología del Capítulo 3.

La implementación descrita en este Capítulo, cubrió los aspectos de: la obtención de la información de un contenido curricular descrito en un documento XML, la creación de la base de datos que almacena la información semiestructurada, las clases para almacenar y reconstruir un documento curricular en la base de datos y las diversas opciones de consultas existentes en base al modelo de datos semiestructurado planteado.

Con la implementación se facilitó la obtención, integración, almacenamiento y manejo de la información semiestructurada, con lo cuál se cubren los objetivos con los que debía cumplir la propuesta del modelo de datos semiestructurado resultante de la aplicación de la metodología definida en el Capítulo 3.

Durante la implementación, el reto afrontado más complejos correspondió al aprendizaje del manejo de información en documentos con formato XML; su lectura e interpretación, usando Java, se complicó al no tener experiencia en el tema, lo que forzó a entender a un nivel avanzado los conceptos y definiciones tanto de XML como de DTDôs. Así mismo, cabe mencionar que la tecnología usada para la realización de este prototipo, no presento ni una limitación.

Capítulo 5

Conclusiones

Este Capítulo presenta las conclusiones a partir de la elaboración de este documento de tesis. En la sección 5.1 se presenta el trabajo realizado, la sección 5.2 muestra las aportaciones realizadas con el modelo de datos semiestructurado desarrollado para el manejo de contenidos curriculares. Finalmente, la sección 5.3 muestra el trabajo futuro identificado para el manejo de datos semiestructurados.

5.1 Trabajo Realizado

La conclusión y aportación principal de este trabajo de tesis es la definición de un modelo de datos semiestructurado aplicado al manejo de contenidos curriculares del campo de computación e informática.

El modelo de datos semiestructurado ha sido definido a partir de una metodología definida en el Capítulo 3, la cuál se basa en el estudio de la información semiestructurada a manejar. Adicionalmente, el modelo ha sido validado mediante la implementación de un prototipo de software.

5.2 Aportaciones

De tal forma, los objetivos alcanzados por este trabajo de tesis han resultado en las siguientes aportaciones:

- Definición de una metodología para definir modelos de datos semiestructurados.
- Desarrollo de un modelo de datos semiestructurado orientado al manejo de contenidos curriculares del campo de computación e informática, definido a partir de la metodología propuesta en esta tesis.
- Definición de estructuras para facilitar la representación de cualquier tipo de información semiestructurada que pueda presentarse.
- Facilidades, en el diseño, para la obtención e integración de la información semiestructurada a almacenar.
- Almacenamiento de la información que permite: la reconstrucción exacta del documento almacenado, anidamiento de información hasta un fondo arbitrario, búsquedas fáciles dentro de la base y almacenamiento de cualquier tipo de estructuras.

El modelo de datos semiestructurado realizado en esta tesis, se basó en el estudio de de los tipos de modelos existentes: complejos y free - form. Lo que permitió distinguir las características y fallas que los modelos de datos semiestructurados existentes presentaban, con lo cual se definieron los objetivos que el modelo propuesto debía cumplir: adecuado almacenamiento de los datos y facilidades para la obtención e integración de la información.

La creación de un prototipo, a partir de los elementos resultantes del Capítulo 3 e implementados en el Capítulo 4, ha probado que el modelo de datos semiestructurado, resultante de la propuesta de este documento de tesis, cumple con los objetivos que se plantearon.

Las lecciones aprendidas durante la elaboración de este trabajo de tesis son:

- La definición de modelos de datos para el manejo de información semiestructurada, debe partir del estudio de las características de la información que se va a manejar.
- Deben tenerse definidas las operaciones que se desean realizar sobre la información, porque, tienen implicaciones sobre el diseño del modelo de datos que se cree para el manejo de la información.
- La experiencia en el diseño de sistemas, usando herramientas de modelado como UML y Entidad - Relación, facilitará la definición del modelo de datos semiestructurado que se desarrolle.

5.3 Trabajo Futuro

El uso y manejo de información semiestructurada sigue teniendo una cantidad de retos que pueden ser afrontados en trabajos futuros. A continuación se presentan posibles líneas de investigación identificadas a partir de este trabajo de tesis:

- Integración de información basada en semántica a cualquier nivel de anidamiento.
- Añadir elementos para la administración de la información como:
    - Registro y Control de Cambios, y
    - Registro de asociaciones entre diversos documentos.
- Aplicación del modelo de datos propuesto en otros campos distintos al de contenidos curriculares.
- Agregar mas opciones de búsquedas, por ejemplo, permitiendo crear consultas personalizadas por el usuario.
- Permitir obtener información de de documentos de diferentes formados, por ejemplo, PDF.
- Agregar opción de agregar y descargar diversos documentos al mismo tiempo.
- Permitir a la aplicación tener la habilidad de identificar información ya almacenada que ya había sido almacenada, evitando tener duplicidad.
- Permitir manejar diferentes versiones de documentos.

Referencias Bibliográficas

[1] François Bry, Dan Olteanu and Sebastian Schaffert. Towards grouping constructs for semistructured data. *Proceedings of 12th International Workshop on Database and Expert Systems Aplications*, pages: 66-70, Munich, Germany, September 2001.

[2] Sudarshan S. Chawathe, Serge Abiteboul and Jennifer Windom. Representing and querying changes in semistructured data. *Proceedings of 14th International Conference on Data Engineering,* pages: 4-13, Orlando, FL, USA, February 1998.

[3] Elisa Bertino and Elena Ferrari. XML and data integration. *Internet Computing, IEEE,* 5(6):76-76, November 2001.

[4] Mengchi Liu and Tok Wang Ling. A Data Model for Semistructured Data with Partial and Incosistent Information. *Proceedings of 7th International Conference on Extending Database Technology 2000.* Konstanz, Germany, March 2000.

[5] Harms David. *JSP, Servlets, and MySQL*. M&T Books, New York, 2001.

[6] Serge Abiteboul. Semistructured data: from practice theory. *Proceedings of Annual IEEE Symposium on Logic in Computer Science,* pages: 379-386, Boston, MA, USA, June 2001.

[7] Arciniegas Fabio. *Programación Avanzada Con XML.* McGraw-Hill, 2002.

[8] Dan Suciu. Management of semistructured data. *ACM SIGMOD Record*, 26(4):4-7, December 1997.

[9] http://www.visual-paradigm.com/download.php, Visual Paradigm, Octubre 2008.

[10] Alin Deutsch, Mary Fernandez and Dan Suciu. Storing Semistructured data with STORED. *Proceedings of International Conference on Management of Data and Symposium on Principles of Database Systems*, pages: 431-442, Philadelphia, Pennsylvania, USA, 1999.

[11] Antonio Badia. Conceptual Modeling for Semiestructured Data. *Proceedings of the Third International Conference on Web Information System Engineering,* pages: 170-177, December 2002.

[12] Serge Abiteboul, Peter Buneman, Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, San Francisco, 2000.

[13] http://jakarta.apache.org/, The Apache Software Foundation, Octubre 2008.

[14] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. *IEEE Internacional Conference on Data Engineering,* pages: 251- 260, March 1995.

[15] Sudarshan Chawathe, Hector García-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinoum Jeffrey Ullman and Jennifer Widom. The TSIMMIS Project: Integration of the Heterogeneous Information Sources. *16th Meeting of the information Processing Society of Japan*, pages: 7-18, Tokyo, Japón, 1994.

[16] R. Ahmed, P. De Smedt, W. Du, W. Kent, M.A. Ketabchi, W.A. Litwin, A. Rafii and M.-C. Shan. The Pegasus Heterogeneous Multidatabase System. *Computer*, 24(12):19-97, December 1991.

[17] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. STRUDEL: A web-site management system. *Proceedings ACM SIGMOD International Conference on Management of Data*, pages: 549-552, Tucson, Arizona, USA, May 1997.

[18] Anthony Tomasic, Louiqa Raschid and Patrick Valduriez. Scaling Acces to Heterogeneous Data Sources with Disco. *IEEE transactions on Kowledge and Data Engineering*, 10(5):808-823, Sep/Oct 1998.

[19] AnHai Doan, Ying Lu, YoonKyong Lee and Jiawei Han. Object Matching for Information Integration: A Profiler-Based Approach. *Proceedings of Information Integration on the Web 2003,* pages: 53-58, Acapulco, Mexico, August 2003.

[20] Juan Carlos Lavariega Jarquín. Integración de Información, Servicios y Aplicaciones en Internet y Dispositivos Móviles. *Propuesta de Cátedra de Investigación en el tecnológico de Monterrey, Campus Monterrey,* Monterrey, Nuevo León, Octubre 2002.

[21] Chris Clifton, Hector García-Molina and David Bloom. HyperFile: A Data and Query Model for Documents. *ACM SIGMOD Record*, 30(1):47-54, March 2001.

[22] http://java.sun.com/j2se/1.4.2/download.html, Sun Microsystems, Octubre 2008.

[23] http://dev.mysql.com/downloads/mysql/5.0.html, Sun Microsystems, Octubre 2008.