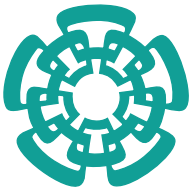

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Computación

Programación automática de sistemas Web a partir de descripciones XML. Caso de estudio: Sistemas de Información Geográfica

Tesis que presenta

Claudia Guadalupe Hernández Pérez

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica
con opción en Computación

Director de la Tesis

Dr. Sergio Victor Chapa Vergara

México, D.F. **Junio 2008**

Resumen

En el desarrollo de sistemas de información, los tiempos y costos de desarrollo son un factor importante, sobre todo considerando que los requerimientos son extensos y la aplicación final necesita ser robusta. La amplia gama de aplicaciones provoca una variabilidad en la especificación de requerimientos que a su vez origina una prolongación en el desarrollo de los sistemas, por la cantidad de código que se necesita producir. Una solución ha sido la generación automática de código, que usada en distintas capas del sistema beneficia en los tiempos y costos. En esta tesis el objetivo es el desarrollo de un generador de código de sistemas Web con un caso de estudio de Sistemas de Información Geográfica (SIG).

El diseño de un sistema Web es soportado por aspectos principales de su interfaz, funcionalidad, navegación y manejo de transacciones, por lo cual un generador de código debe permitir una serie de descripciones que se transformen en una codificación automática. La ventaja es cuando hay nuevos requerimientos las modificaciones se hacen sólo en las descripciones para generar de nuevo código ad-hoc.

El problema consiste en desarrollar un generador automático de sistemas Web basado en descripciones XML (*eXtensible Markup Language*) añadiendo características para visualización de mapas y manejo de datos geográficos con base en GML (*Geography Markup Language*). La aplicación Web generada se coloca en un servidor para una conexión JDBC a una base de datos relacional con un modelo de datos geográficos vectoriales de mapas del INEGI con escala 1:50,000. La visualización se lleva a cabo en el navegador del cliente en donde se tiene un soporte para XHTML (*eXtensible Hypertext Markup Language*).

Los resultados de este trabajo muestran una contribución en tecnología de software y en metodología para el desarrollo de base de datos geográficas. El campo de SIG es muy amplio en sus aplicaciones, manteniendo un problema abierto en el desarrollo de base de datos geográficas. Este proyecto cubre un espacio en el desarrollo de SIG con perspectivas a futuro en dispositivos móviles.

Abstract

In the development of information systems, the time and development costs are an important issue, especially considering that the requirements are extensive and the final application needs be robust. The wide range of applications leads to variability in the specification of requirements which in turn creates an extension in the development of systems, by the amount of code that needs to produce. One solution has been automatically generating code, used in different layers of the system benefits in time and cost. In this thesis the objetive is to develop a code generator of Web systems with a case study of Geographic Information Systems (GIS),

The design of a Web system is supported by major aspects of its interface, functionality, navigation and management of transactions, by which a code generator should allow a number of descriptions that are transformed into an automatic encoding. The advantage is when there are new requirements changes are made only in descriptions to generate code ad-hoc again.

The challenge is to develop an automatic generator of Web systems based in descriptions XML (*eXtensible Markup Language*) by adding features for viewing maps and management of geographic data based on GML (*Geography Markup Language*). The Web application generated is placed on a server for a JDBC connection to a relational database with a model of geographic data vector of INEGI maps with scale 1:50,000. The display is done in the client browser where it has a support for XHTML (*eXtensible Hypertext Markup Language*).

The results of this project show a contribution in software technology and methodology for the development of geographic database. The field of GIS is very broad in their applications, maintaining an open problem in developing geographic database. This project covers an area in the development of GIS with future prospects in mobile devices.

Agradecimientos

A mis padres y hermanos.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico recibido.

A todos los investigadores y personal administrativo que conforma el departamento de Computación. Especialmente a mi director de tesis el Dr. Sergio Victor Chapa Vergara por todo el apoyo brindado para la terminación de mis estudios de maestría.

Al M. en C. Noe Sierra Romero por su tiempo y paciencia.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En la actualidad, los datos nominales y espaciales se procesan mediante Sistemas de Información Geográfica que son capaces de almacenar y gestionar tales datos de una forma eficiente. Un punto importante es que aún no existe una metodología que ayude a la construcción de tales sistemas de información geográfica de manera que soporte cambios en los requerimientos sin afectar la integridad del mismo. Por consiguiente, se tiene un problema de eficiencia en el desarrollo de software.

En el presente trabajo de tesis se propone el desarrollo de un generador automático de código para sistemas Web a partir de una descripción basada en XML, con una aplicación que muestra ventajas con la generación de un sistema Web capaz de adaptarse fácilmente a los cambios en la especificación de requerimientos. Por ende, el impacto tecnológico es la mejora en el desarrollo de software. La finalidad del sistema es mejorar la eficiencia en la creación de subsistemas, reduciendo tiempos en el desarrollo de SIG fácilmente. El proyecto es relevante para el desarrollo de SIG para hidrología, como caso de estudio para el estado de Colima.

1.1. Antecedentes y motivación

Los Sistemas de Información Geográfica representan una herramienta útil para el control ambiental y toma de decisiones. Existen herramientas comerciales para la creación de apliaciones SIG, sin embargo, hay que integrar varias de las herramientas para conjuntar la funcionalidad deseada. El proceso de desarrollo de un SIG o de cualquier otro sistema implica una extensión en tiempo que puede verse prolongada cuando los requerimientos son muy cambiantes.

La necesidad de una metodología para el desarrollo de sistemas Web que involucren el manejo de datos complejos (nominales y geográficos) implican también una necesidad de planteamientos de nuevas arquitecturas para la generación de dichos sistemas. Es imprescindible enfocarse al ahorro de tiempo y dinero para la implementación de sistemas cuando el usuario final solicita cambios durante el ciclo de vida del sistema. Así, surge la propuesta

de una arquitectura que involucra la generación automática de código para sistemas Web que considera información geográfica.

1.2. Objetivos del proyecto

Objetivo general

Diseñar y construir una herramienta de generación automática de código para Sistemas de Información Geográfica en entornos Web a partir de esquemas de descripción (entidades, lógica de negocios, interfaz) basados en GML y XML. Se tratará de generar de forma automática la mayor parte del sistema, tratando de cubrir el 100 % del mismo.

Objetivos particulares

- Definición de la extensión del lenguaje GML (GML + x)
- Asociación del modelo geográfico al esquema GML + x
- Definición de la arquitectura del sistema
- Diseño y construcción de un generador de código

1.3. Justificación

El aumento del nivel de abstracción en el desarrollo de software se ha llevado a cabo desde hace tiempo. En las décadas pasadas los investigadores y desarrolladores de software han creado abstracciones que los ayudan a programar en términos de diseño. Estas abstracciones incluyen lenguajes y plataformas que elevan el nivel de abstracción pero que aún conservan un enfoque orientado a abstracciones en el dominio de tecnologías computacionales más no a abstracciones en el espacio del problema. En 1980, surgió la propuesta de la Ingeniería de Software Asistida por Computadora (Computer Aided Software Engineering, CASE por sus siglas en inglés) que permitía a los desarrolladores expresar sus diseños en términos de representaciones de programación gráfica de propósito general. Las debilidades de las herramientas CASE radican en su falta de soporte en Calidad de Servicio (tolerancia a fallas, seguridad, distribución transparente) y en la poca capacidad para escalar a sistemas complejos [1].

Por otro lado, la evolución en lenguajes y plataformas ha incrementado el nivel de abstracción hasta llegar a la programación orientada a objetos donde ya existen bibliotecas con funcionalidades que el desarrollador sólo tiene que utilizar a su conveniencia. Sin embargo, aún existe una fuerte dependencia entre el código desarrollado y el uso de dichas bibliotecas. El código de muchas aplicaciones aún es generado y mantenido manualmente lo que provoca un gran consumo de tiempo y esfuerzo principalmente para desarrollo, configuración y aseguramiento de calidad [3].

El desarrollo de software lleva implícito el mantenimiento del mismo, tarea que se vuelve difícil cuando la implementación ha sido llevada a cabo por grupos diferentes, esto provoca que los desarrolladores atiendan las secciones sensibles a cambios en los requerimientos [2]. De esta manera, el tiempo de desarrollo de las aplicaciones y aún más su mantenimiento representa un costo elevado. Por lo tanto, el acelerar el proceso de desarrollo de sistemas apresura también la obtención de información para soportar el proceso de toma de decisiones en el campo ambiental y desarrollo sustentable [5].

Ante todas las situaciones que surgen cuando se necesita un desarrollo de software orientado a Web, un generador automático de código a partir de descripciones XML soluciona los problemas que resultan en el transcurso de realización del desarrollo de algún sistema. La facilidad en los cambios en dichas descripciones, a su vez facilitan las modificaciones del sistema ante el cambio en los requerimientos. Por consiguiente el proceso de desarrollo se acelera y los costos disminuyen.

1.4. Descripción del proyecto

El proyecto de tesis consiste en la construcción de una aplicación para la generación automática de código a partir de descripciones XML (*eXtensible Markup Language*) y GML (*Geography Markup Language*). Dichas descripciones abarcan estructura, comportamiento, interfaz y acceso de tal forma que la codificación que tengan que realizar los desarrolladores sea mínima. El caso de estudio es un Sistema de Información Geográfica, que cuenta con un modelo de datos complejos nominales y espaciales, cuya descripción se basa en el lenguaje de marcas GML, el cual puede reflejar aspectos particulares de estos sistemas. El trabajo de tesis se divide en tres aspectos fundamentales que abarcan la implementación total del proyecto. En las siguientes secciones se hace una descripción general de dichos aspectos.

1.4.1. Módulo de descripciones

El primer módulo consiste en el desarrollo de esquemas y descripciones que detallan todos los aspectos del sistema. Dichas descripciones se dividen en varios documentos, cada uno con una finalidad específica. De manera general las descripciones están compuestas por:

1. **Descripción de entidades**. Las entidades del sistema, son básicamente entidades existentes en la base de datos, las cuales incluyen las descripciones geográficas con características geométricas y espaciales. La base utilizada fue el estándar GML 3.1.1 que publica *Open Geospatial Consortium Inc*. [32] que proporciona una gama de esquemas que sirven para desarrollar esquemas propietarios. De manera particular, tomamos como base un esquema en específico (*features.xsd*) que a su vez hace referencia a otros esquemas del estándar. Consideramos que esta base es suficiente para

desarrollar las descripciones en su parte geográfica. El objetivo de estas descripciones fue abarcar aspectos que detallen la situación de la entidad dentro de la base de datos, aspectos que serían de utilidad para el acceso a la misma.

2. **Descripción de interfaz**. Este tipo de descripción abarca propiedades y caractgerísticas relacionadas con la interfaz Web. En lo posible, el objetivo es detallar las características básicas de la interfaz. Se tienen en cuenta aspectos de métrica de diseño como los mostrados en el curso en línea Diseño de Interfaces Visuales [39]. Debido a que una interfaz Web puede estar diseñada de muchas maneras, es un tanto difícil realizar descripciones de tales características. Por tal motivo, se propone un diseño de interfaz específico que hasta cierto punto sea parametrizable. De esta manera, la interfaz puede ser descrita de manera más general. Para describir la interfaz es necesario dividir el sistema en módulos. A cada módulo se le asocia una interfaz principal. Como se mencionó anteriormente la interfaz esta predefinida en cuanto a su estructura principal. Consta de un encabezado de página, un menú, un submenú y la parte de contenidos. Entre las características que se detallan están,

   2.1. Dimensiones del encabezado de página, del menú y submenú

   2.2. Color, fuente y tamaños de letra

   2.3. Nombre del archivo que desplegará la interfaz

   2.4. Directorio donde estará ubicado el archivo que desplegará la interfaz

   2.5. Color general de la interfaz. Este color define el tema que identificará a esa interfaz para representar así un módulo del sistema

   La parte de contenidos, que es una sección de la interfaz estática, no se define en este archivo de descripción. La razón es que los contenidos no son estáticos, en algunas ocasiones dependerán de los resultados de consultas realizadas por el usuario. En el punto 1 de la sección 4.3.1.11 del capítulo 4 se hace mención de la forma general en la cual están descritos estos contenidos.

3. **Descripción de menús**. A cada uno de los módulos que se han identificado en el sistema se les asocia un menú, ésto con la finalidad de hacer una mejor organización de las funciones que proveerá el sistema. Cada menú de cada módulo está descrito en un archivo de descripción donde se específica cada componente del menú y a qué enlace está asociado. Dentro de esta descripción también se especifican características de presentación, tales como color, fuente, tamaño de fuente, etc. Generalmente cada opción del menú tiene un enlace hacia un submenú por lo que también se realizan descripciones de submenús para cada opción de cada módulo identificado.

4. **Descripción de comportamiento**. Dentro de estas descripciones se abarca el detalle del comportamiento del sistema. Aquí se describe la forma en la que se ha de navegar en el sistema. La descripción de comportamiento está dividida en módulos de la misma forma en que se han identificado los módulos en el sistema. Cada módulo se divide en entidades y cada entidad en modos de comportamiento (especificados

también en las opciones de los submenús). Cada modo de comportamiento representa una funcionalidad en el sistema y consta de 4 secciones:

4.1. Forma básica. Es un formulario inicial en el que se pueden realizar búsquedas de información. En la forma básica se especifican también las características de la interfaz.

4.2. Forma de resultados. Es la forma en que se presentarán los resultados de la forma básica, donde se especifican características de interfaz. Debido a que los resultados que se despliegan en esta forma dependen de los parámetros que introdujo el usuario en la forma básica, es necesario crear un documento de descripción con los resultados de la consulta formada a partir de dichos parámetros. Para lograrlo es necesaria la creación del documento de forma dinámica, es decir, en el momento en que una página Web se haya cargado.

4.3. Forma de detalle. Es la forma que sirve para presentar la información completa acerca del elemento que el usuario haya elegido en la forma de resultados, es decir, los detalles de la información de un elemento en particular. De manera similar que en la forma anterior, se genera un documento de descripción dinámico, puesto que también depende de la selección del usuario. Básicamente, se especifican parámetros para indicar la forma en que se han de desplegar los resultados. Dentro de esta forma existen operaciones (transacciones) que pueden ser aplicadas al elemento al que se este haciendo referencia.

4.4. Forma de transacción. Esta forma es consecuencia de la forma anterior en la que se ha de realizar una transacción en la base de datos (inserción, actualización o eliminación) respecto al elemento que haya sido seleccionado en las formas anteriores. Básicamente, se especifican parámetros para la formulación de la transacción a realizar.

Cada una de las formas mencionadas contienen la descripción de pre-acciones y post-acciones que indican el flujo de la navegación. De manera general, cada acción describe parámetros como nombre de la página Web a la que se dirige, nombre de la página de la cual fue invocada la página Web actual y los parámetros que han de ser transferidos en cada invocación a una página Web. Para la validación de información que el usuario introduce en las diferentes formas descritas se cuenta con una biblioteca de funciones (creada por el generador de código en *JavaScript* [10]) que únicamente habrá de ser agregada al código generado por la aplicación para que pueda ser utilizada por el sistema.

5. **Descripción de mapas**. Debido a que el caso de estudio es un Sistema de Información Geográfica, es fundamental la generación de mapas. Se cuenta con descripciones que detallan la manera en que han de desplegarse dichos mapas en las páginas Web. Es importante mencionar que para dar interactividad al usuario se cuenta con funciones predefinidas de *JavaScript* que proporcionan principalmente funciones de acercamiento, alejamiento y selección de entidades geométricas.

Las descripciones están desarrolladas en XML tomando como base esquemas desarrollados en XML-Schema (*eXtensible Markup Language Schema*). Dichos esquemas sirven como plantillas que predeterminan los documentos XML. Así se pueden validar dichos documentos de modo que sean legales. El generador de código recibe como entrada dichas descripciones y las procesa para generar la aplicación final, razón por la cual los documentos deben estar sintácticamente bien formados. Es importante mencionar que el generador de código tiene una asociación con una aplicación de modelado visual [40] que proveerá todas las descripciones con el diseño que el generador requiere.

1.4.2. Módulo generador de código

El siguiente módulo consiste en la implementación del generador de código. Esta actividad es la más extensa y su complejidad radica en la asociación del modelo plasmado en las descripciones a código interpretable por un navegador.

El generador de código está desarrollado en Java en conjunto con JDOM (*Java Document Object Model*) [29] y xTags (*Tag Libraries*) [23] para el manejo de documentos y transformaciones a código HTML (*HyperText Markup Language*) [25] o SVG (*Scalable Vector Graphics*) [27]. El generador de código lee las descripciones y las convierte en código que pueda ser compilado y ejecutado.

El generador de código se auxilia de plantillas [31] que consisten de métodos prefabricados que realizan varias tareas y de hojas de estilo para las transformaciones a HTML o SVG. La razón por la cual se ha elegido trabajar con plantillas es debido al hecho de que describir un método resulta complicado, puesto que un método puede realizar una gran variedad de funciones y de alguna manera es difícil que el generador interprete las descripciones que son de por sí complicadas. De esta manera, los métodos contenidos en plantillas son parametrizables, es decir, reciben parámetros que hasta cierto punto determinarán la función que realizará (e.g. indicar el tipo de dato que regresará).

Ahora bien, las plantillas de hojas de estilo consisten en código XSLT (*eXtensible Stylesheet Language Transformations*) [13, 24] que puede programarse para que las descripciones XML se transformen en código HTML embebido en las páginas Web. Básicamente se tendrá código XSLT que involucre formularios y elementos de dichos formularios, menús y submenús. De esta manera el generador lee las descripciones, utiliza las plantillas XSLT previamente definidas y se realiza la transformación para generar código HTML embebido dentro de un JSP.

Debido a que se manejan datos geográficos, es necesario el uso de mapas dentro de la aplicación que se va a generar. Así se hace uso de plantillas para transformar las descripciones de datos geográficos a mapas visibles en la Web. Para lograr esto, es necesario el uso de hojas de estilo que trabajan de manera similar a las plantillas mencionadas anteriormente, sin embargo, el código transformado es código SVG propio para manejo de gráficos [19, 20]. Dicho código SVG, a diferencia del HTML, no puede estar embebido dentro del

JSP, por lo que debe quedar almacenado en un archivo y luego invocarlo en HTML mediante *frames*. El procedimiento general de transformación de descripciones mediante plantillas de estilos a visualización en la Web se muestra en la figura 1.1.

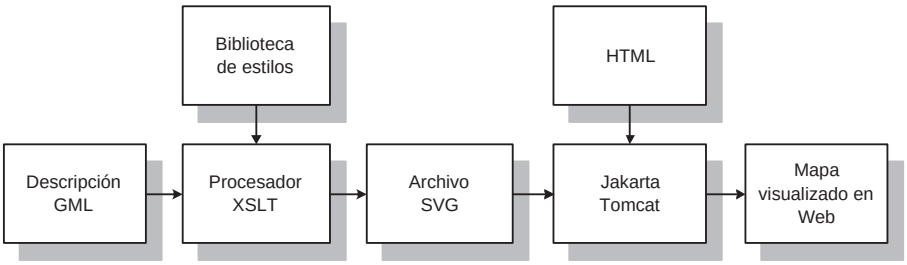

Figura 1.1: Proceso de conversión de descripción a visualización Web

Para lograr lo anterior es necesario que el generador tenga una estructura definida, de modo que reciba como entrada las descripciones, que pueda interpretarlas y que genere el código correspondiente. Se tienen definiciones para los parámetros de las entidades, para las páginas JSP y para los métodos que representan el comportamiento. Además, se cuenta con métodos para la lectura y escritura que involucra el manejo de documentos XML. El generador cubre las secciones:

- Generación de interfaz Web
- Generación de módulos de comportamiento
- Formación de consultas y transacciones a la base de datos
- Generación de documentos XML
- Transformación de documentos XML
- Generación de *beans*
- Generación de JSPs
- Generación de mapas

Básicamente cada sección representa un módulo de la aplicación. Los módulos interactúan entre sí para generar el código final.

1.4.3. Módulo de aplicación final

El último módulo consiste de la aplicación final. Este módulo representa la aplicación final en donde se obtiene el código generado incluyendo clases, código javascript, código SVG y páginas JSP organizado en directorios de forma tal que sea posible agregar la aplicación a un contenedor de aplicaciones Web (e.g. Jakarta Tomcat) y que pueda ejecutarse desde un navegador. De manera general, el sistema proveerá las siguientes funcionalidades:

- Autenticación de usuario
- Subsistema de consulta y captura de datos nominales y espaciales. Éstos serán para la capa de hidrología de la base de datos geográfica del estado de Colima
- Visualización de mapas en la capa de hidrología
- Interacción del usuario con el sistema para la captura y funciones de visualización (seleción de entidades geográficas)

La figura 1.2 muestra un entorno global en el que está situado el proyecto de tesis tanto a nivel de herramienta como a nivel de resultado final. Se observa la relación que tiene con otro proyecto de tesis [40] para conformar todo un ambiente funcional sobre sistemas SIG. CAD-WEB [40] tiene el propósito de modelar la base de datos y el sistema Web para dar como resultado el modelado plasmado en descripciones que representan la entrada al CAD-SIG para generar el código de la aplicación Web denominada como ATIC considerando un caso de estudio específico (capa de hidrología del estado de Colima). CAD-WEB se fundamenta en modelos establecidos y tecnologías de información que hacen posible la implementación de un generador de código basado en descripciones realizadas a partir de un diseño.

1.5. Organización de la Tesis

El presente documento se divide en 7 partes en las que describimos el trabajo desarrollado para generar automáticamente la aplicación Web a partir de descripciones XML para un Sistema de Información Geográfica.

En el capítulo 1 se presenta una introducción donde se explican los antecedentes y motivación del presente trabajo de tesis. También se proporciona una descripción general del proyecto, así como los objetivos y la justificación del mismo.

En el capítulo 2 definimos un Sistema de Información Geográfica, sus características y su importancia en el campo ambiental. Se definen los modelos de datos y las herramientas computacionales que son necesarias para su representación. Se hace una breve descripción de la integración de Sistemas de Información Geográfica sobre Internet.

En el capítulo 3 se presentan dos modelos de desarrollo de aplicaciones que involucran técnicas de generación de código existentes y una descripción del proceso de generación de

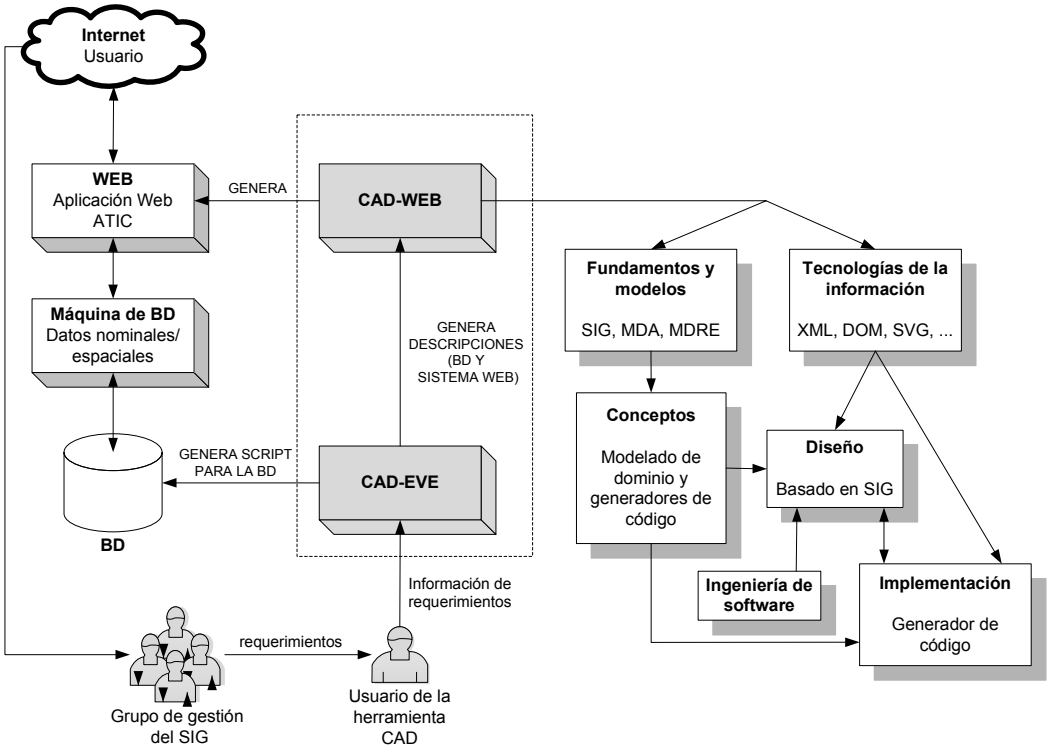

Figura 1.2: Entorno de ubicación del proyecto de tesis

código. Finalmente, se hace una descripción de las tecnologías de información existentes que son utilizadas en el presente proyecto.

En el capítulo 4 se describe el desarrollo del generador de código mediante descripciones GML y XML. Se muestran los esquemas que se siguieron en las descripciones. El proceso de generación de código, la descripción de las transformaciones realizadas y finalmente la integración de módulos en la aplicación final.

En el capítulo 5 se hace una descripción técnica de la estructura y funcionamiento del generador de código. En el capítulo 6 se muestran las pruebas de unidad realizadas sobre el caso de estudio elegido y la prueba completa del generador. Finalmente en el capítulo 7 se mencionan las conclusiones del trabajo desarrollado y el trabajo futuro. Se presenta un comparativo con otras herramientas existentes de generación de código y se mencionan las ventajas del presente proyecto.

Capítulo 2

Sistemas de Información Geográfica

Durante los últimos años hemos sido testigos del incremento en el interés sobre aplicaciones SIG (Sistema de Información Geográfica) en desarrollo sustentable y planeación. En particular, en hidrología y recursos acuíferos el impacto en la calidad de los SIG y el manejo ambiental es importante, por consiguiente el desarrollo en la tecnología SIG es importante [7]. Los objetivos son capturar, almacenar, manipular, analizar y visualizar los diversos conjuntos de datos geo-referenciados. La hidrología, inherentemente espacial integra de manera natural a un SIG, en donde involucran tres componentes principales [5]:

- construcción de datos espaciales
- integración de capas del modelo espacial
- SIG e interfaz del modelo

Antes del advenimiento de la unidad hidrográfica de Sherman (1932), el modelo hidrológico era principalmente empírico y basado en datos limitados. En las siguientes décadas, aproximadamente en los 50's, los modelos analíticos simples recibieron gran atención y el procesamiento de datos fue manejado manualmente de una forma sencilla. A partir de la década de los 60's existió un importante avance en la obtención de datos de modo que el acceso a la información era más fácil, pasando posteriormente a la década de los 70's donde las computadoras permiten la aceleración de la tecnología de los SIG [5].

2.1. ¿Qué es un SIG?

El término Sistema de Información Geográfica (SIG) data de mediados de los 60's y parece provenir de dos aplicaciones separadas, una en Canadá y la otra en Estados Unidos [5]:

- Administración de información relacionada y recolectada por el inventario de los recursos naturales de Canadá, obteniendo estimados del área disponible para cierto tipo de usos y planificación.

- Acceso a diferentes tipos de datos de grandes volúmenes de información para análisis y presentación de los resultados en formato mapa, el objetivo fue usarlos en modelos de transporte a gran escala y mapeo de los censos de población en Estados Unidos.

Un SIG es una tecnología basada en computadora de propósito general para el manejo de datos geográficos en forma digital. Es un sistema que tiene un conjunto de subsistemas que sirven para: la captura, el almacenamiento, el análisis, la visualización y la graficación de diversos conjuntos de datos espaciales o geo-referenciados [7].

La característica principal de los SIG es el manejo de datos complejos basados en datos geométricos (coordenadas e información topológica) y datos de atributos (información nominal), los cuales describen las propiedades de los objetos geométricos tales como: puntos, líneas y polígonos. El núcleo de software de los SIG es el sistema manejador de base de datos que en inglés ha sido denominado DBMS *Data Base Managemente System*. Dicho sistema debe tener la capacidad para almacenar y gestionar las entidades asociando su representación geométrica con su representación nominal constituída con atributos [5].

Los SIG contemplan herramientas de explotación de datos, las cuales implican un subsistema de postprocesamiento de consultas y resultados a través de reformateo, tabulación, graficación y trazos de mapas [5]. Para éste último, la representación gráfica implica el uso de herramientas para cartografía.

Comúnmente en un SIG la consulta, el despliegue de un mapa, el reporte tabular y el gráfico, no es el resultado final. En su amplio abanico de aplicaciones, los SIG tienen como objetivos analizar la información espacial y modelar los procesos dinámicos que generan y conforman la información almacenada en una base de datos. Con frecuencia los usuarios quieren analizar situaciones, hacer inferencias o simular procesos, con el fin de tomar decisiones. El interés radica en el análisis espacial con métodos deterministas o estocásticos, tanto en su modalidad discreta como contínua [5].

De la amplia gama de posibilidades que tienen los SIG en el manejo de información, un requerimiento importante es brindar una interfaz de usuario amigable. En principio, se puede mencionar que existe una extensión de SQL (*Structured Query Language*) y GSQL (*Geographic Structured Query Language*). Los lenguajes o ambientes visuales para base de datos han sido una área relativamente reciente en computación, lo cual tomó una importancia especial en los SIG por la naturaleza visual-espacial que tiene la información geográfica [5].

Los modelos en hidrología se han incrementado enormemente tanto en escala como en complejidad. Los hidrologistas han encontrado en los SIG una muy útil tecnología que puede manejar una basta cantidad de datos y sus relaciones espaciales que aparecen en distintas fuentes. Actualmente, los proyectos de aplicación de SIG mantiene diversos enfoques [7]:

- modelos de cuencas hidrográficas
- predicción de flujos

- hidrología de aguas subterráneas
- monitoreo de la calidad del agua y contaminación
- evaluación en la erosión del suelo, etc.

Los modelos hidrológicos se incorporan en la capa de explotación de datos, lo cual implica crear un nivel que sirva de eslabón o interfaz entre el SIG y los modelos hidrológicos. Tres niveles de liga entre un SIG y un modelo hidrológico son los que se pueden señalar [5]:

1. Nivel de presentación. Es el uso del SIG sólamente para determinar los parámetros que requieren los modelos hidrológicos existentes. Esto significa que los SIG son usados para construir datos de entrada y presentar la información en sus diferentes formas.

2. Nivel de herramienta. Corresponde a usar un SIG en el nivel básico de gestión ligado con un módulo de análisis espacial. Éste puede actuar como herramienta para analizar y visualizar resultados del modelo de hidrología. La combinación de un SIG y un modelo hidrológico hace completo uso del análisis espacial del SIG para derivar datos distribuidos.

3. Nivel de integración. El tercer nivel, es el que tiene como finalidad encajar completamente los modelos hidrológicos en los SIG. La integración conduce a un sistema automático el cual tiene todos los mecanismos para acceder datos de entrada, módulo de análisis espacial y explotación del modelo hidrológico.

2.2. Modelos de base de datos

Para que un proyecto de SIG tenga éxito, requiere de modelos de datos que tomen en cuenta la información espacial y nominal, caracterizada por la aplicación del proyecto. El objetivo de la construcción de los modelos de datos es tener independencia entre conceptualización, representación e implementación de la base de datos geográfica [5].

2.2.1. El modelo conceptual de datos EVE: el modelo Entidad-Vínculo-Extendido

Los problemas de la complejidad de los objetos de datos individuales y el volúmen de los datos científicos que se desprenden de los modelos espaciales-temporales, requieren del diseño y desarrollo de un sistema operativo que recupere, almacene y organice eficientemente los datos. Por un lado, el modelo relacional basado en esquemas de nombres de atributos que conforman relaciones, ha sido relevante para los datos convencionales, con grandes ventajas de desempeño e independencia [8]. Sin embargo, el principio del modelo

relacional de asumir tablas simples y no conjuntos en los espacios de nombres, constituye uno de los escollos principales para datos complejos y científicos. La concepción de especialización y generalización que se plantea en el modelo conceptual entidad-vínculo-extendido (EVE) resulta una visión jerárquica la cual es eficiente para la representación y gestión de un gran número de objetos complejos de datos [41]. El objetivo es obtener la ventaja de los despliegues de datos basados en atributos y la recuperación de sub-conjuntos de objetos de grandes volúmenes de datos científicos.

El modelo ER, del inglés *Entity-Relationship*, fue propuesto por Peter Chen [5] a mediados de los setentas. Su valor principal es el de un modelo formal de nivel de abstracción muy alto, que basado en una técnica diagramática sencilla puede codificar la información del mundo real. El resultado fue un fundamento para la tecnología de base de datos y un método exitoso para el problema de diseño de base de datos en ingeniería de software. A partir del modelo original ER propuesto por Chen, se han llevado a cabo diversas investigaciones, aplicaciones y extensiones. Algunas de ellas han sido los sistemas basados en el modelo Entidad-Vínculo-Extendido (EVE) realizados en el departamento de Computación del Cinvestav [41, 40]. Éstos ofrecen como ventajas adicionales la asociación ternaria, generalización y especialización; pudiendo incorporar fácilmente los conceptos orientados a objetos..

La meta de modelar base de datos es diseñar una base de datos eficiente y apropiada. Algunas de las fortalezas de la metodología son: rendimiento, integridad, comprensibilidad y expandibilidad.

2.2.2. Modelos de base de datos geográfica

Dentro de la construcción de un SIG, un objetivo preponderante es la definición de un marco de conceptos computacionales para un sistema de base de datos geográfica [5]. Este sistema esta fundamentado en un modelo lógico de datos asociado a un modelo físico embebido en un Sistema Manejador de Base de Datos. Varios subsistemas soportan lenguajes que atienden la definición, al almacenamiento y a la gestión de la base de datos. En el caso de datos geo-referenciados es necesario que el modelo lógico contemple los datos nominales y espaciales; así como lenguajes que manejan dichos tipos de datos. Un adecuado marco computacional es fundamental para que los modelos científicos ejecuten una gestión eficaz y eficiente para el análisis espacial de datos y simulación.

2.2.2.1. Modelo relacional de datos

El modelo relacional de datos ha sido y es exitoso en la práctica, especialmente en aplicaciones tradicionales en donde se tienen grandes cantidades de datos homogéneos nominales. Sin embargo, al mismo tiempo que la simplicidad resulta ser una ventaja, por otro lado muestra restricciones. Las ventajas del enfoque relacional pueden ser resumidas en [8]:

- Se construye en base a la teoría matemática de relacionales y conjuntos. El elemento estructural es el de relación sobre el cual se definen operaciones de conjuntos.
- Se define formalmente lenguajes de consultas: algebraico, cálculo relacional de tuplas y cálculo relacional de dominios; todos éstos equivalentes entre sí.
- Genera automáticamente accesos de rutas para relaciones, ofreciendo una alta independencia de datos.
- Optimiza lenguajes descriptivos y el lenguaje de cálculo relacional.
- Brinda un enfoque entendible y comprensible para un usuario.

A pesar de todas estas ventajas una principal debilidad que enfrenta este modelo es el nivel de abstracción matemática para su diseño. Diseñar una base de datos basada en el enfoque relacional requiere de una habilidad de abstracción elevada para resolver problemas, sus principios matemáticos son determinantes, sin embargo, la conceptualización del mundo real requiere ciertas habilidades de abstracción.

2.2.2.2. Modelo geográfico de datos

La información y los datos es una división que se presenta entre la concepción en el mundo real y su representación simbólica codificada. La complejidad de la realidad geográfica da lugar a diversas formas de adquirir la información, asociada a los eventos espaciales-temporales. Su codificación a representaciones finitas simbólicas (por ejemplo atributos de valores de suelo), son conformadas en estructuras más complejas (por ejemplo matriz de variables del suelo), las cuales están asociadas a una localización espacial y que varía con el tiempo [5]. La computadora es el instrumento tecnológico que convierte la información geográfica en un número finito de registros de base de datos y permite llevar a cabo computaciones numéricas que dependen directamente de una cierta longitud de palabra.

Las bases de datos geográficas toman en cuenta dos clases principales de datos: espaciales (localización o gráficos) y nominales (atributos no gráficos). La componente espacial de un SIG se describe con frecuencia como una serie de capas, o coberturas, donde cada una contiene una serie de rasgos que son relacionados funcionalmente con un tema. Por ejemplo, las cartas del INEGI, azul, negro, verde, etc. [5]. Es importante mencionar que cada capa se registra posicionalmente de acuerdo a otras capas a través de un sistema coordenado común en una proyección dada (*Universal Tranversal de Mercator* cónica o cilíndrica). Los objetos pueden ser puntos, líneas o áreas, que pueden ser descritas por atributos que toma valores dentro de un dominio el cual tiene asociado: escalas, operaciones y predicados.

2.2.2.3. Modelo espacial de datos

Para describir la fisonomía espacial geográfica, la realidad debe ser concebida mediante un modelo espacial de datos el cual puede ser descrito en dos niveles: modelo de objetos y modelo de campo.

1. Modelo de objetos.

   1.1. Punto. Un objeto cero-dimensional que especifica la localización geométrica a través de un conjunto de coordenadas. Un nodo es un tipo especial de punto, también un objeto cero-dimensional, que es una conexión topológica o punto final que puede especificar una localización geométrica.

   1.2. Línea. Un objeto uni-dimensional que es un segmento de línea determinado entre dos puntos. Formas especiales de líneas incluyen las siguientes:

   - Cadenas, como una serie de puntos que dan un segmento de línea.
   - Arcos, como el lugar geométrico de todos los puntos que forman una curva definida por una función matemática
   - Polígonos, como una secuencia consecutiva y dirigida de segmentos de línea o arcos con nodos en cada punto terminal.

   1.3. Área. Un objeto bi-dimensional que es contínuo y acotado, un objeto que puede incluir su frontera. Áreas individuales son representadas por polígonos.

   1.4. Celda. Un objeto bidimensional que representa un simple elemento en un espacio discreto referenciado a un superficie contínua.

   1.5. Pixel. Un objeto pictórico bidimensional que es el elemento más pequeño indivisible de una imagen.

   1.6. Símbolo. Elemento gráfico que representa alguna característica a los puntos sobre un mapa.

2. Modelado de campo.
   Representa la variación espacial de una simple variable por una colección de objetos discretos. Un campo puede ser asociado con una variable medida sobre una escala contínua o discreta. Una base de datos geográfica puede comprender varios campos de los cuales seis son los que comúnmente son usados en un SIG.

   2.1. Muestreo irregular de puntos. La base de datos contiene un conjunto de tripletas (x, y, z) que representan valores de puntos de una variable en un conjunto finito de localizaciones irregularmente distribuidas. Por ejemplo, medidas de precipitación, en puntos geográficos de coordenadas espaciales (x, y) con altura z.

   2.2. Muestreo regular de puntos. El muestreo de puntos es un arreglo que se mapea sobre un enrejado regular. Este campo tiene una similitud con el anterior excepto por el espaciamiento regular en donde se tienen localizadas las mediciones. Ejemplo de este caso es el *Digital Elevation Model* (*DEM*), donde se tiene topografía en un modelo de elevación.

2.3. Contornos. La base de datos contiene un conjunto de líneas, cada una representada por un conjunto de parejas ordenadas (x, y) que están asociadas a un sólo valor z. Los puntos en cada conjunto se asume que están linealmente conectados. Ejemplos, líneas de contorno cartográfica de una misma elevación, mapas hisoyético de datos.

2.4. Polígonos. El área es dividida en un conjunto de elementos poligonales, de tal forma que cualquier localización cae dentro de un polígono. Cada polígono tiene asociado un valor cuya variable se asume tiene para todas las localizaciones con el polígono. Las fronteras son definidas por un conjunto de parejas ordenadas (x, y) de puntos. Ejemplos son AGEB's (Áreas Geoestadísticas Básicas), uso de suelo, área básica de simulación, polígonos de Thiessen.

2.5. Enrejado de celdas. El área es dividida en un enrejado regular de celdas, como por ejemplo un enrejado de diferencias finitas. Cada celda contiene un valor y la variable se supone que tiene un valor para todas las localizaciones dentro de la celda. Un ejemplo, son los valores en imágenes de percepción remota.

2.6. Red triangular. El área es dividida en triángulos irregulares. El valor de la variable es especificada en cada vértice del triángulo y se supone que varia linealmente sobre el triángulo. Como ejemplo más ilustrativo se tiene el modelo de elevación TIN (*Triangular Irregular Network*).

2.2.2.4. Modelo de atributos nominales

Los datos no espaciales describen las características de las capas gráficas y los fenómenos que ocurren en localizaciones geográficas específicas. Por ejemplo, en una capa gráfica de hidrología, para el modelo de lluvia y corrientes, podemos incluir localizaciones de estaciones meteorológicas. Esta puede incluir una serie de variables no espaciales como pueden ser: el tiempo, humedad, temperatura, etc. En el caso del modelo relacional de datos, el problema ha sido tratado ampliamente considerando un esquema de relación, con dominios que son asignados a nombres de atributos. En las bases de datos geográficas, el requerimiento de una vinculación con los modelos espaciales propician una clasificación de cuatro clases de datos de tipo nominal.

- Atributos nominales. Las entidades geográficas están parcialmente determinadas por un conjunto de atributos nominales. Cada atributo proporciona información descriptiva, que puede ser cualitativa y cuantitativa, acerca de las características que asume la fisonomía de un mapa. Los atributos tienen asignados dominios homogéneos de valores y son vínculados a elementos gráficos, a través de identificadores comunes. Dichos identificadores son denominados geocódigos y sirven de llave única que asocia los registros espaciales y no espaciales.

- Referencias a datos geográficos. Son aquellos conjuntos de información geográfica digital que sirven para referir espacialmente otros datos manejados por los SIG.

- Índices geográficos. Describe incidentes o fenómenos que ocurren en una localización específica. A diferencia de los atributos, éstos no describen rasgos del mapa en sí mismo. Este tipo de datos describe átomos o acciones (tales como muestreo y protocolos de análisis), los cuales pueden ser relacionados a localizaciones geográficas (tales como estaciones de monitoreo de la calidad del agua). Los datos de referencia geográfica representan una entidad adicional en el modelo geográfico que identifica la localización de los elementos o fenómenos. Con una mayor perspicacia en el modelo EVE, podemos incorporar otros atributos que mantenga cierta participación en la vinculación.

- Vínculos espaciales. Comúnmente existen algunas descripciones conceptuales entre los objetos espaciales. Las asociaciones espaciales son definidas principalmente como descripciones de proximidad, adyacencia y conectividad de los rasgos de los mapas.

2.2.3. El modelo objeto-relacional de Postgres

Postgres [28] es un modelo relacional extendido el cual asume el modelo relacional de datos tradicional e incluye alguna de las características del modelo orientado a objetos. Los conceptos de generalizaciones y especializaciones que se definen en el modelo extendido EVE derivan en un modelo de datos lógico, el cual debe incluir el empotrado de tipos extras, considerando la posibilidad de definir tipos de datos abstractos, soporte para herencia, soporte para objetos compartidos y capacidad para soportar reglas y funciones para manejar datos de una manera sencilla. En principio, la capacidad que tiene Postgres para incluir lo anterior, es una razón principal para usarla como una plataforma adecuada de desarrollo, más aún, el tiempo y el espacio son dos factores importantes que refuerzan la decisión de establecer Postgres como la base para un SIG espacio-temporal. Postgres tiene la capacidad de:

- incluir el manejo del tiempo, tanto del mundo real, como el tiempo de la base de datos,

- soportar tipos de datos geométricos (puntos, líneas, polígonos) y operadores espaciales asociados (área, perímetro, distancia),

- manejar tipos de *objetos grandes* los cuales puede incluir bitmap para datos basados en *raster*.

Como un resumen de datos de tipo geométrico de Postgres:

- point. Consiste de un par de coordenadas, las cuales describen una localización en un espacio de dos dimensiones. Los puntos geográficos pueden ser usados para identificar características topolgógicas, parecidas a lo que son nodos de polígonos, o entidades geográficas a nivel más alto como pueden ser cabecera municipal, ciudad, estación de transporte.

- lseg. Representa un simple segmento de línea recta y consiste de un punto inicial y un punto final. Su uso principal es como parte de una entidad más grande y más complicada.
- path. Consiste de un arreglo de lseg.
- polygon. Consiste de un arreglo de lseg en donde coincide el punto inicial con el final.
- box. Describe un rectángulo.

2.2.3.1. PostgresSQL objeto-relacional

Con base en la naturaleza de los sistemas de base de datos orientados a objetos, actualmente, existe un debate de su uso estándar y los sistemas relacionales [28]. Existen pocos prototipos de investigación, los cuales pueden combinar las buenas características de SQL con accesos simples y conexiones estándar con ODBC o JDBC. El punto, es tomar la dirección de los sistemas orientados a objetos, tomando ventaja de las tecnologías eficientes relacionales, produciendo un nuevo paradigma que son los *Sistemas de Base de Datos Objeto-Relacional* [5], como lo es PostgresSQL.

PostgreSQL es considerado como uno de los mejores gestores de base de datos de todo el software abierto[1]. Muchas empresas han emprendido el uso de esta herramienta beneficiandose en la reducción de costes y aumentando la fiabilidad. Dentro de sus características se tiene:

- cumple completamente con ACID,
- cumple con ANSI SQL,
- integridad referencial,
- replicación (soluciones comerciales y no comerciales) que permiten la duplicación de bases de datos maestras en múltiples sitios de réplica,
- interfaces nativas para ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python,
- una API[2] abierta,
- procedimientos almacenados,
- soporte nativo SSL: Secure Socket Layer,
- lenguajes procedurales,
- extensiones para SHA1, MD5, XML y otras funcionalidades,

[1] PostgreSQL es un código abierto desarrollado en la Universidad de California en Berkeley
[2] Interfaz de programación de aplicaciones, paquetes de clases de Java [11]

- sistema de tipos de datos extensible para proveer tipos de datos definidos por el usuario y rápido desarrollo de nuevos tipos.

La forma tradicional de almacenar información espacial digital es utilizar algún formato de los programas SIG como ArcView[3], MapInfo[4] o GRASS[5]. En general, estos formatos son poco flexibles y bastante anticuados, ya que son un conjunto de archivos que almacenan la información sobre las coordenadas, la proyección espacial y los datos asociados a cada elemento. El formato más común utiliza archivos con extensión DBF para almacenar los datos, formato que presenta muchas limitaciones que parecen totalmente arbitrarias como por ejemplo nombres de campos limitados a 10 caracteres, 3 o 4 tipos de datos, entre otras. En cambio PostGIS (y en general cualquier extensión espacial a un RDBMS[6]) soluciona estos problemas y además, permite una mayor flexibilidad porque hace posible realizar operaciones espaciales en la fuente de datos misma, lo que conlleva varias ventajas, entre ellas:

1. Facilidades para el programador. Evita el tener que implementar ciertas operaciones en la aplicación (intersección, búsqueda por cajas, proyección geográfica de los datos, etc.). Además se puede optar por desarrollar partes de la lógica de la aplicación vía procedimientos almacenados, o generar nuevos conjuntos de datos a partir de los existentes de manera mucho más fácil a través de vistas, subconsultas, uniones o tablas temporales.

2. Mayor control sobre la aplicación. Al separar los datos del lugar donde se encuentra la aplicación (y no tener necesariamente que compartir archivos vía un sistema de archivos distribuido si queremos separar los datos de la aplicación). Este control presenta beneficios en varios niveles, como el poder distribuir la carga hacia el RDBMS, o la aplicación, según el tipo de ésta y sus características.

PostGIS [22] es simplemente una extensión a PostgreSQL, que define nuevos tipos de datos, crea dos tablas con información relevante al sistema (proyección de los datos y columna que posee la información geográfica) y define también las funciones de manejo de información como procedimientos almacenados.

2.2.3.2. El modelo OMG abierto

La arquitectura dirigida por modelos (*Model-Driven Architecture* [38], MDA por sus siglas en inglés) está basada en los estándares establecidos de OMG (*Object Management Group*), MDA separa negocios y la lógica de la aplicación de la plataforma. Los modelos de plataforma independiente de una aplicación o de funcionalidad y comportamiento de

[3]Sistema completo para proporcionar y usar información geográfica [33]

[4]Líder de mercado en herramientas de software para la visualización y análisis de datos corporativos a través de mapas [35]

[5]SIG usado para administración y analisis de datos geoespaciales, procesamiento de imágenes, producción de gráficas/mapas, modelado espacial y visualización [37]

[6]Sistema Manejador de Base de Datos Relacional

sistemas se construyen usando UML y los otros estándares de modelado asociados. También se pueden realizar a través de MDA sobre cualquier plataforma, abierta o propietaria, incluyendo servicios Web, J2EE y otros. Estos modelos documentan la funcionalidad del negocio y comportamiento de una aplicación separada del código que la implementa.

2.3. La capa WebSIG

2.3.1. El Web general

Es básicamente un medio de comunicación de texto, gráficos y otros objetos multimedia a través de Internet, es decir, la web es un sistema de hipertexto que utiliza Internet como su mecanismo de transporte o desde otro punto de vista, una forma gráfica de explorar Internet. La Web fue creada en 1989 en un instituto de investigación de Suiza [9], la Web se basa en buscadores y el protocolo de transporte de hipertexto http (*Hypertext Transport Protocol*). La mayoría de los documentos de la Web se crean utilizando lenguaje HTML (*Hypertext Markup Language*).

La Web se ha convertido en un medio muy popular de publicar información en Internet, y con el desarrollo del protocolo de transferencia segura, la Web es ahora un medio de comercio electrónico donde los consumidores pueden realizar compras o transacciones bancarias.

2.3.2. El Web geográfico

WebSIG [4] es la aplicación de la tecnología Web a SIG. Basado en los protocolos TCP/IP, HTTP, FTP, DNS y SMTP. Internet nos provee de servicios como World Wide Web y FTP. Gran parte de la atención se ha enfocado en desarrollar funcionalidades SIG sobre Internet. Web y SIG se combinan para hacer un Web SIG (WWW SIG o SIG sobre Internet), que lo hace más conveniente para liberar y compartir información espacial, mientras se toma ventaja de los recursos de hardware existentes. Como resultado, Web SIG amplía el rango de acceso, establece el acceso transparente a usuarios al SIG, reduce el costo del sistema y simplifica la operación. Web SIG puede ser planeteado en dos categorías:

1. El servidor desempeña todos los requerimientos de los clientes. Este tipo de Web SIG usa CGI *(Common Gateway Interface)* o ASP *(Active Server Pages)* principalmente, lo cual es muy eficiente en el desarrollo de programas.

2. El cliente puede desarrollar partes de los requerimientos de sí mismo. Este tipo de Web SIG adopta Java, que pueden reducir la carga de trabajo en el servidor, pero excede el tiempo de desarrollo del programa.

En la actualidad, las técnicas de desarrollo de Web SIG incluyen CGI, ASP, ISAPI *(Internet Server Application Programming Interface)* y NSAPI *(Netscape Server Application Programming Interface)* y tecnología Java.

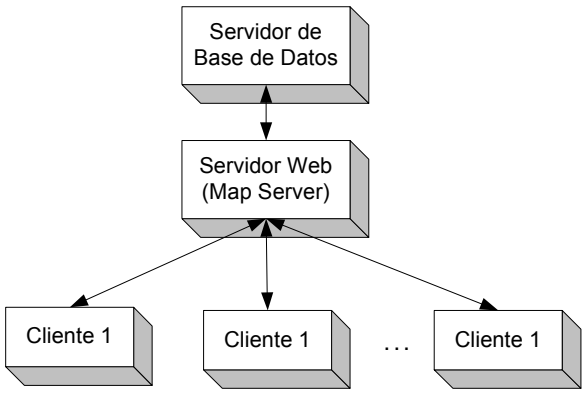

Figura 2.1: La estructura del sistema [4]

Las flechas en la figura 2.1 denotan flujo de datos. El servidor de base de datos almacena los datos. El programa ASP se ejecuta sobre un servidor Web. Por medio de IIS (*Internet Information Services*), ASP responde a las peticiones de los clientes, invoca datos de la base de datos, invoca a los módulos de manejo de datos SIG y regresa los resultados a los clientes.

2.4. Conclusiones

Durante este capítulo se han descrito los fundamentos necesarios enfocados a un Sistema de Información Geográfica. Estos fundamentos son la base teórica para el desarrollo de la tesis. En la figura 1.2 del capítulo 1 se muestra un diagrama que representa la estructura general del proyecto, los fundamentos de SIG, los modelos existentes que tienen como base la generación de código para formar código independiente de la plataforma de desarrollo, así como las herramientas para el modelado de la información.

Capítulo 3

Modelos y tecnologías de la información

En este capítulo se describe el diseño de la aplicación propuesta que dará solución al problema que trata la tesis. Inicialmente se propone una arquitectura de ambiente funcional en la cual se utilizará la aplicación, así como la arquitectura propia del generador. Finalmente, se describe paso a paso el diseño de cada componente de la aplicación con base en las arquitecturas propuestas.

3.1. Modelos de desarrollo

En la actualidad, los desarrolladores usan lenguajes orientados a objetos que permiten utilizar bibliotecas para minimizar la necesidad de reinventar servicios comunes, sin embargo, existen problemas que radican principalmente en el crecimiento de la complejidad de la plataforma, ya que se van creando dependencias más fuertes entre clases y métodos que pueden complicar la tarea de programación [1], [2]. Además, se requiere que los desarrolladores pongan especial atención en detalles tácticos de programación, por lo que es difícil saber qué partes de las aplicaciones son susceptibles a efectos debido a cambios en los requerimientos del usuario, en la plataforma o el lenguaje. Muchas aplicaciones son codificadas y mantenidas manualmente lo que provoca tiempo y esfuerzo excesivo. Todos estos problemas dan como resultado que los desarrolladores implementen soluciones no óptimas que duplican código, violan principios de arquitectura y hacen complicada la evolución del sistema.

3.1.1. Ingeniería dirigida por modelos

Una propuesta para atacar los problemas es desarrollar tecnologías de Ingeniería dirigida por Modelos (Model-Driven Engineering, MDE por sus siglas en inglés) [1]. MDE es una herramienta que unifica la documentación, el análisis, y la transformación de la información hacia varias fases, a través del ciclo de vida de un sistema, tomando en cuenta aspectos de la estructura y comportamiento de la aplicación. La figura 3.1 muestra la relación entre vistas, modelos e implementaciones. Más que replicar las abstracciones que

proveen los lenguajes de programación, los modelos deben abstraer elementos seleccionados de los sistemas complejos implementados. Así se puede generar más de un modelo para así generar más de una vista.

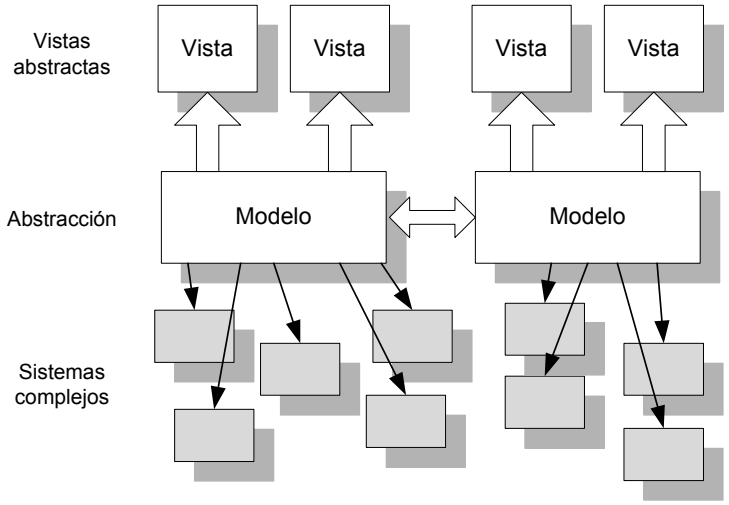

Figura 3.1: Relación entre vistas, modelos e implementación [1]

MDE se sustenta sobre dos aspectos fundamentales, un lenguaje de modelado y una máquina de transformación:

- Lenguajes de modelado de dominio específico, que formalizan la estructura de la aplicación, comportamiento y requerimientos dentro de dominios particulares.
- Máquinas de transformación y generadores, que analizan ciertos aspectos de los modelos y sintetizan código fuente, entradas a simulación, descriptores de despliegue XML o representaciones de modelo alternativas.

Esta propuesta es importante para el tema de tesis debido a que separa la parte de descripción de la aplicación de la parte de generación de código. En cuanto a lo referente a descripción de la aplicación se abarca tanto la estructuración de la aplicación como el funcionamiento de la misma permitiendo hacer una generación automática más completa, dejando muy poca codificación manual al desarrollador.

3.1.2. Ingeniería inversa dirigida por modelos

Considerando que los cambios en los requerimientos son una característica inevitable en el desarrollo de aplicaciones, es conveniente abarcar modelos que tengan en cuenta estos aspectos. La ingeniería inversa es el proceso de comprensión de software y de producir un modelo de éste a un nivel de alta abstracción disponible para documentación, mantenimiento o reingeniería. Sin embargo, existen dos grandes problemas:

- Es difícil predecir que tanto tiempo requerirá la ingeniería inversa.
- No hay estándares para evaluar la calidad de la ingeniería inversa que se desarrolla.

La *ingeniería inversa dirigida por modelos* (Model-Driven Inverse Engineering, MDRE por sus siglas en inglés) puede atacar estas dificultades [2]. Un *modelo* es una representación de alto nivel de algún aspecto de un sistema de software. La propuesta MDRE usa especificación formal y generación de código automático para revertir el proceso de ingeniería inversa. Para lograrlo, se genera un modelo expresado por un lenguaje de especificación formal soportado por una herramienta de generación de código para producir otras versiones del sistema original. Si la versión generada demuestra cercanía a la original, entonces la ingeniería inversa es suficiente.

MDRE utiliza dos tipos de modelo, un *modelo de programa* y un *modelo de dominio* de la aplicación. El modelo de programa provee la representación de las funciones que el programa ejecuta (por ejemplo especificaciones algebraicas). El modelo de dominio de la aplicación expresa conceptos de dominio, sus relaciones y sus significados independientemente del programa. Debido a que en MDRE el modelo de programa y el modelo de dominio están presentes, se pueden hacer conexiones explícitas entre ellos.

El proceso MDRE incluye tres pasos suponiendo que se tiene ya una versión inicial de la aplicación:

- Se construye un modelo de dominio a partir de descripciones sobre la aplicación.
- Se construye un modelo de programa expresando el código fuente en la que vaya a ser desarrollada la aplicación, como una especificación.
- Se definen interpretaciones para conectar las operaciones del modelo del programa y conceptos de dominio.

Finalmente se ejecuta un generador de código para producir una aproximación a la aplicación original. Si la aplicación generada produce resultados muy cercanos a los originales entonces la ingeniería inversa ha tenido éxito.

La figura 3.2 muestra un ejemplo de ingeniería inversa aplicada a una aplicación numérica llamada ZBRENT escrita en lenguaje C. El dominio de la aplicación es computación numérica, específicamente, la búsqueda de la raíz de una función real. Se construye un modelo de dominio obteniendo material sobre análisis numérico. Luego se utiliza SLANG (lenguaje de especificación formal para describir el dominio de la aplicación y del programa) para modelar el dominio y el programa. Finalmente se usa el generador de código de SLANG para producir una versión del modelo ZBRENT al que se le aplico la ingenería inversa y se compara con el programa original con un conjunto de funciones de prueba.

Por lo tanto, es posible compartir el modelo de dominio, en otras palabras, el modelo del dominio tiene un valor que va más allá que un simple programa de ingeniería inversa. Así se puede amortizar los costos de modelado de dominio en actividades de mantenimiento para la misma aplicación o para otras similares [2].

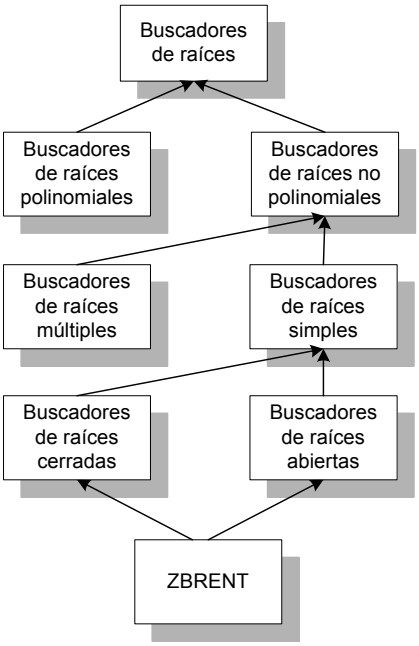

Figura 3.2: Ingeniería inversa aplicada a la aplicación numérica ZBRENT [2]

3.2. Generación de código

3.2.1. ¿Qué es un generador de código?

La generación de código es la técnica de escribir y usar programas que construyen aplicaciones y código del sistema [18]. Un generador de código lee de un diseño, usa un conjunto de plantillas para construir el código de salida que implementa el diseño. La separación entre la lógica de la generación de código y el formateo de la salida en las plantillas es semejante a la separación entre la lógica de negocios y las interfaces de usuario en aplicaciones Web [31]. La figura 3.3 muestra la forma básica de un generador de código.

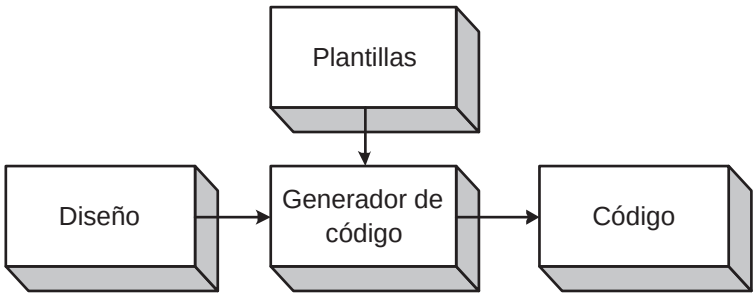

Figura 3.3: Forma básica de un generador de código

Los asistentes *(wizard)* son generadores pasivos. Éstos escriben código una sóla vez y éste código esta disponible para darle mantenimiento. Los generadores de código son activos ya que continuamente dan mantenimiento al código sobre múltiples ciclos de operación. Conforme el diseño cambia, la entrada al generador cambia también y se genera nuevo código para que corresponda con el diseño.

3.2.2. Características de un generador de código

Las características de un generador de código óptimo son:

- Calidad. Se requiere que el código de salida sea al menos tan bueno como el que pudiésemos haber escrito a mano. La propuesta basada en plantilla de los generadores construyen código que es fácil de leer y depurar. Debido a la naturaleza activa del generador, los errores encontrados en el código de salida pueden ser arreglados en las plantillas.

- Consistencia. El código debe usar clases, métodos y nombres de argumentos consistentes. Esto hace a las interfaces fáciles de usar y más fácil añadir más código al ya existente.

- Productividad. Se debe generar el código más rápido que escribirlo a mano. Sin embargo, no se puede lograr la generación en el primer ciclo del mismo. El valor de productividad viene después cuando se regenera el código para atender el cambio de requerimientos.

- Abstracción. Debemos estar habilitados para especificar el diseño en una forma de abstracción, libre de detalles de implementación. De esta forma, se puede replantear el generador si se desea mover a otra plataforma de tecnología.

3.2.3. El proceso de generación de código

En este apartado se muestra una vista global para el desarrollo de un generador de código dentro de una organización de ingeniería de software[1]. Las siguientes secciones siguen el ciclo de vida de un generador: la determinación del problema, el desarrollo y el mantenimiento [21].

3.2.3.1. Determinación del problema

El primer paso es determinar la necesidad y decidir si la generación de código es una solución correcta. El método formal ayuda a definir con más claridad el problema.

[1]Organización de desarrollo de aplicaciones, el caso de estudio es una aplicación de contabilidad empresarial [21]

El método formal

Este método involucra al generador en un proyecto de desarrollo completo siguiendo el proceso de diseño e implementación de proyectos de ingeniería de software con el que cuente la organización. La primera fase del diseño de software es la fase de requerimientos, en la cual se recolecta información acerca de lo que el generador necesita hacer. En esta fase, se requiere definir el ámbito de trabajo manejado por el generador. En particular se debe tener claro lo siguiente:

- Definir lo que hará el generador en su primera versión. Por ejemplo, una versión en la que se generen las interfaces del sistema.
- Establecer lo que hará el generador en sus siguientes versiones. Esto ayudará a definir la arquitectura del generador. Por ejemplo, la navegación entre las interfaces del sistema.
- Qué es lo que el generador tendrá bajo su responsabilidad. Esto permitirá asegurar qué características claves son implementadas y cuales no. Por ejemplo, la generación de interfaces, navegación, transacciones, etc.
- Cómo es que el generador podrá ser mantenido.
- Definir el tipo de generador que se implementará.
- Establecer qué herramientas de desarrollo se utilizarán.

3.2.3.2. Definición de la infraestructura

Después de determinar el problema que se requiere resolver y cómo se va a resolver, el siguiente paso es tener las herramientas listas. Este proceso involucra varios pasos. Los más importantes son controlar las versiones a través del desarrollo, configurar el ambiente de pruebas y seleccionar el conjunto de herramientas que serán usadas.

Asegurar el control del código fuente

El control del código fuente es muy importante cuando se trabaja con generadores de código ya que pueden existir bloques de código grandes que se sobreescriben continuamente. Es necesario que se este familiarizado con el sistema de control de código fuente para así hacer pruebas sobre algunos bloques de código y no sobre todo el proyecto. Se debe usar un sistema de control de código fuente para el generador y el código que crea, particularmente cuando se integra un generador en un sistema existente para reemplazar código. Esto permite ejecutar y probar el generador de forma aislada sin preocuparse de corromper el flujo principal.

Estandarizar las herramientas de desarrollo

Si se están usando diferentes herramientas de desarrollo en el generador, se debe dedicar un poco de tiempo para preparar esas herramientas. De manera particular, se debe tomar

tiempo para crear un paquete de instalación con los archivos ejecutables y los módulos extras que se requieran para dar soporte al generador. Asegurar el documento de procedimiento de instalación.

3.2.3.3. Desarrollo

El desarrollo es la fase más crítica, ya que es en esta fase donde se hace la implementación completa del generador de acuerdo a la arquitectura y a las especificaciones planteadas al principio de este procedimiento.

3.2.3.4. Despliegue y documentación

Esta fase se concentra en crear las herramientas de instalación. Se debe construir y documentar una herramienta de instalación de los componentes requeridos para el generador. Además se necesita facilitar pruebas de instalación y dentro de la documentación describir lo que el generador puede realizar y como es que lo hace, qué archivos afecta y el modo de ejecución.

3.2.3.5. Mantenimiento

Si el generador crea una gran porción de código, hay que darle mantenimiento si es necesario desde la arquitectura del generador.

3.3. Tecnologías de información

Se refiere al estudio, diseño, desarrollo, puesta en práctica, ayuda o gerencia de los sistemas de información computarizados, particularmente usos del software y hardware. La tecnología de la información (TI) se entiende como aquellas herramientas y métodos empleados para recabar, retener, manipular o distribuir información. La tecnología de la información se encuentra generalmente asociada con las computadoras y las tecnologías afines aplicadas a la toma de decisiones.

La tecnología de la Información (TI) está cambiando la forma tradicional de hacer las cosas, las personas que trabajan en gobierno, en empresas privadas, que dirigen personal o que trabajan como profesional en cualquier campo utilizan la TI cotidianamente mediante el uso de Internet, las tarjetas de crédito, el pago electrónico de la nómina, entre otras funciones; es por eso que la función de la TI en los procesos de la empresa como manufactura y ventas se han expandido grandemente [34].

Las siguientes son las herramientas de tecnología de información que se utilizaron en el desarrollo de la tesis.

- XML (Extensible Markup Language), desarrollado por el consorcio World Wide Web, se basa en el uso de marcas o etiquetas para diferenciar los diversos elementos que pueden existir en un documento. La principal característica de XML es su

caracter estricto; un documento XML ha de cumplir las normas de sintaxis y estructuración para que sea reconocido como tal; para ello, el documento debe de estar bien formado [9]

- XML-Schema, desarrollado por el consorcio World Wide Web, describe la estructura autorizada, el contenido y las limitantes del documento XML. La especificación de XML-Schema define varios tipos de datos incorporados tales como String, Integer, Boolean, Date, Time, etc. y provee la capacidad de definir nuevos tipos. Otra característica clave de XML-Schema es que soporta herencia permitiendo rehúso de módulos de software eficiente [12].

- GML (Geography Markup Language), desarrollado por el consorcio OpenGIS, es una gramática escrita en XML-Schema para modelado, transporte y almacenamiento de información geográfica [14].

- JDOM es una API (Application Programming Interface) en Java desarrollada por Jason Hunter para crear y manipular documentos XML [29].

- XSLT, desarrollado por el consorcio World Wide Web, es un lenguaje de programación que nos permite transformar un documento XML en otro (SVG o HTML) [19, 13].

- SVG (Scalable Vector Graphics), desarrollado por el consorcio World Wide Web, es básicamente un documento XML para describir gráficas en 2D con interactividad y animación. Se puede integrar con HTML, JavaScript y DOM [19, 27].

3.4. Conclusiones

Durante este caítulo, se hizo una descripción sobre los modelos que sirven de base para el desarrollo de la tesis. Cada uno de éstos abarca de manera general la idea que maneja el generador de código a partir de descripciones. De esta manera, la tesis se sustenta en modelos que cubren la problemática planteada y que dan el fundamento necesario para el desarrollo de una arquitectura del proyecto. Es importante el conocimiento sobre el funcionamiento de un generador de código puesto que es el núcleo de la aplicación. Los beneficios de un generador de código mencionados nos dan la pauta para lograr que se implemente un generador de código con estas características y así cumplir con un buen funcionamiento. Finalmente, cada una de las tecnologías de información que son utilizadas para el diseño y la implementación de la aplicación se describieron brevemente. Con la descripción de modelos, del generador de código y de las herramientas a utilizar, se puede abordar de lleno el diseño del proyecto.

Capítulo 4

Diseño de la aplicación

En este capítulo se describe el diseño de la aplicación propuesta que da solución al problema que trata la tesis. Inicialmente se propone una arquitectura de ambiente funcional en la cual se utilizará la aplicación, así como la arquitectura propia del generador. Finalmente, se describe paso a paso el diseño de cada componente de la aplicación con base en las arquitecturas propuestas.

4.1. Definición del problema

Recientemente, en el problema de SIG, la atención se ha enfocado a SIG en Internet, el cual tiene por objetivo liberar y compartir información espacial [4], con la finalidad de dar soporte durante el proceso de toma de decisiones. El desarrollo y mantenimiento de las aplicaciones representa un costo (tiempo y dinero) que llega a ser muy elevado cuando existen cambios en la especificación de requerimientos y aún más cuando el desarrollo se ha llevado a cabo por diferentes grupos de trabajo.

El problema principal a resolver es la ausencia de una metodología para el desarrollo de Sistemas de Información Geográfica. Con esto se tiene una mayor eficiencia en el trabajo de adquisición de datos y es un soporte al proceso de desarrollo de SIG con la característica de ser adaptable a la variabilidad de requerimientos. Para solucionar el problema se ha de establecer una metodología basada en la utilización de una especificación de la caracterización del sistema (estructura, interfaz, funcionamiento) para la generación de la aplicación Web de un Sistema de Información Geográfica auxiliándose de la generación de código.

4.2. Arquitectura del ambiente funcional

El desarrollo de un generador de código implica varias etapas como se explico en la sección 1.4 del capítulo 1, desde la recopilación de requerimientos del sistema hasta la fase de pruebas y regeneración de código. En cuanto al diseño de la aplicación es necesario definir una arquitectura base sobre la cual se va a partir para el desarrollo del generador de código.

4.2.1. Ambiente funcional

En primera instancia hay que ubicar el generador de código dentro de un esquema global para comprender como es que interactúa con otros elementos. En la figura 4.1 se muestra el esquema que representa la ubicación del generador y que se describe enseguida.

1. Se observa como la base es el Sistema Operativo (i.e. Mac OS, Linux, Windows) sobre el que el generador puede funcionar.

2. Enseguida se encuentra el conjunto de APIs (i.e. NetBeans, JDeveloper) para el lenguaje de programación Java (J2SE).

3. El sistema manejador de Base de Datos que en este caso es PostgreSQL por las razones que en el capítulo 2 se mencionaron.

4. Algunas de las aplicaciones comerciales que existen para visualización y consulta de mapas (i.e. Grass, MapServer). Arriba de este nivel se tiene le conector a Base de Datos (JDBC, ODBC) y PostGIS que es el módulo de PostgreSQL para el manejo de datos geográficos.

5. Estos tres primeros niveles (descritos en 1, 2, 3 y 4) constituyen una base para el generador, puesto que son los elementos que el generador ocupa para que el código generado funcione dentro de un ambiente Web.

6. Al mismo nivel que la aplicación del generador de código se encuentra el servidor de aplicaciones Tomcat y el servidor Web Apache quienes interactúan para publicar la aplicación en la Web.

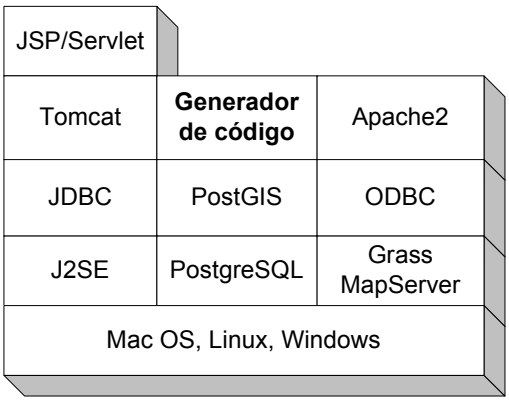

Figura 4.1: Ambiente funcional del generador de código

4.2.2. Arquitectura básica

La arquitectura del sistema se muestra en la figura 4.2. Ésta consta de tres módulos principales: el módulo de descripción, módulo de generador de código y módulo de aplicación final. El módulo de descripción, como su nombre lo indica contiene la descripción completa del sistema, el módulo generador de código lee las descripciones y las convierte en código ejecutable, finalmente el módulo de aplicación final es el código que ya ha sido generado y puede ejecutarse desde un servidor Web. A continuación se explica el diseño de cada uno de los módulos y sus componentes.

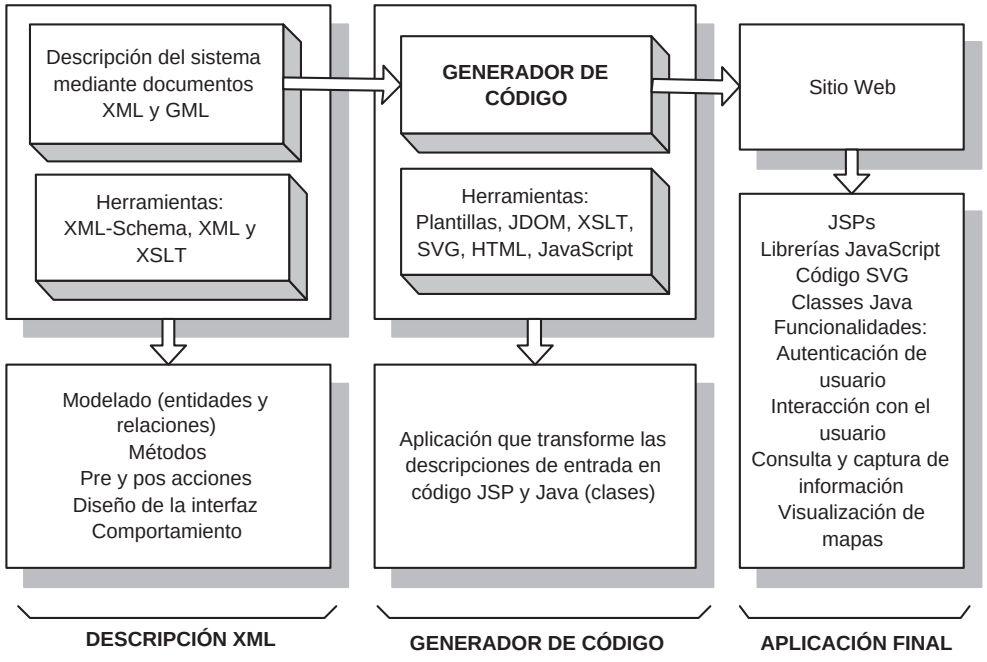

Figura 4.2: Arquitectura básica del generador de código

4.3. Descripción de módulos de la aplicación

Para realizar el diseño de la aplicación se siguió la arquitectura propuesta (figura 4.2), la cual se constituye en tres etapas principales de diseño señaladas: descripción XML, generador de código y aplicación final.

4.3.1. Descripción XML

XML es la base para el desarrollo de todas las descripciones realizadas. Para realizar una descripción de manera coherente también se desarrollaron esquemas basados en XML Schema [12]. Éstos sirven para validar las instancias XML que finalmente son las descripciones. El tipo de validaciones que se realizan en XML Schema es principalmente para

validar estructuras, patrones de los campos, tipos de los campos, etc. De esta forma, podemos hacer una instancia de un esquema y obtener así una descripción bien formada que cumple con el contenido establecido en el esquema.

Para cada descripción se realizó su esquema. Para cualquier sistema que se desee describir, se tiene el establecimiento de un conjunto de descripciones. Cada una de las descripciones realizadas está enfocada a un ambiente Web, puesto que las etiquetas establecidas van dirigidas a elementos de páginas Web. A continuación se describe el diseño y en qué consiste cada una de dichas descripciones.

4.3.1.1. Estructura de un elemento encabezado

Dentro de las descripciones existen definiciones en común, es decir, bloques de descripciones que son utilizados en varios documentos. Un elemento encabezado es un bloque de éstos. La descripción de un elemento encabezado sirve para definir la información que deseamos consultar, es decir, sobre qué campos deseamos obtener datos. Para definir un elemento encabezado en cualquier descripción es necesario definirle ciertos parámetros.

La figura 4.3 muestra la estructura de un elemento encabezado. Generalmente se identifica por un nombre del campo que describe lo que el campo significa, la definición del campo en sí y una etiqueta para indicar si se requiere que sea un enlace a otra página. A su vez, la definición del campo (figura 4.4) contiene atributos que los describen de una mejor manera:

- hay que determinar si el campo pertenece a la entidad que la descripción maneja o no. En el primer caso se define como no externa y en el segundo caso como externa,
- si el campo es externo, entonces hay que definir de que entidad proviene,
- si el campo es externo y se ha definido la entidad de la cual proviene, hay que definir cual es el campo por el cual se relaciona con la entidad que la descripción maneja (llave foránea),
- si el campo se relaciona a través de varias entidades, entonces hay que especificar cada una de la misma manera que en el punto anterior.

Los elementos encabezado se agrupan dentro de una etiqueta que contiene atributos como tipo de letra, tamaño de letra, y color de la misma, de esta manera, se cubre la descripción para la forma en cómo se despliega la información en la página Web. En la figura 4.5 se muestra la estructura de los elementos encabezado agrupados.

La agrupación de los elementos encabezado está asociada directamente con una sección de una página Web como se muestra en la figura 4.6. La primera fila describe el significado de cada campo, de los cuales se desea información y las siguientes filas corresponden a los datos resultado de la consulta. En el ejemplo de la figura 4.6, los datos de la primera columna son ligas que invocan a una nueva página Web.

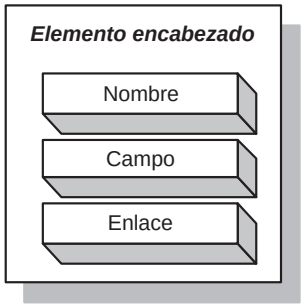

Figura 4.3: Estructura de un elemento encabezado

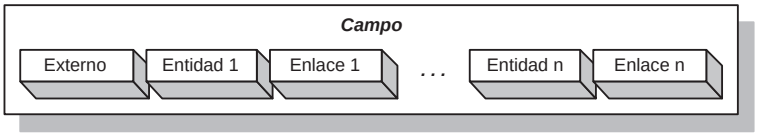

Figura 4.4: Estructura de un campo

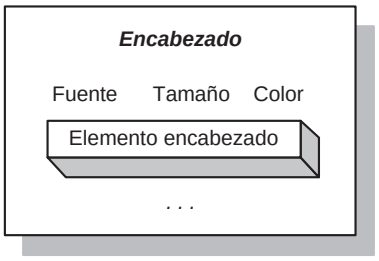

Figura 4.5: Agrupación de elementos encabezado

Nombre del elemento 1	Nombre del elemento 2	. . .	Nombre del elemento n
Dato 11	Dato 21	. . .	Dato n1
Dato 12	Dato 22	. . .	Dato n2
. . .	. . .	. . .	. . .
Dato 1m	Dato 2m	. . .	Dato nm

Figura 4.6: Elementos encabezado en Web

4.3.1.2. Estructura de una acción

Para definir la navegación del sistema es necesario definir una estructura que represente una acción. En la figura 4.7 se representa gráficamente la estructura diseñada para una acción. Los parámetros que toma en cuenta son:

- Nombre de la página a la cual dirigirse.
- Parámetros que deben ser enviados a la página a la cual se ha de dirigir la acción. Estos parámetros son un conjunto de campos como los de la figura 4.4.

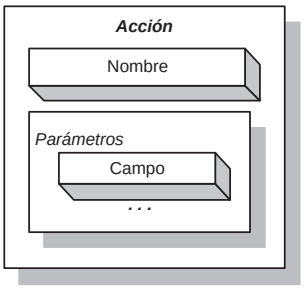

Figura 4.7: Estructura de una acción

Siguiendo esta definición, pueden existir acciones que representen el avance o bien el retroceso hacia una página, en otras palabras, tomando en cuenta el lugar donde se han definido las acciones, pueden existir pre-acciones que se dirigen a una página antes de la actual y post-acciones que se dirigen a otra nueva página. Estas estructuras pueden representar un enlace para determinar la navegación entre páginas.

4.3.1.3. Autenticación de usuario

Esta descripción consiste en el detalle del acceso del usuario al sistema. Se contemplan detalles como usuario y contraseña, así como detalles de diseño de la página que va a autentificar al usuario (color de página, logotipo, título). En la figura 4.8 se muestra gráficamente en qué consiste esta descripción. Todos los elementos que la conforman contribuyen tanto

a la generación de la página que ha de publicarse en la Web, como a la generación de la consulta a la Base de Datos, para poder autenticar al usuario y darle acceso a un módulo del sistema.

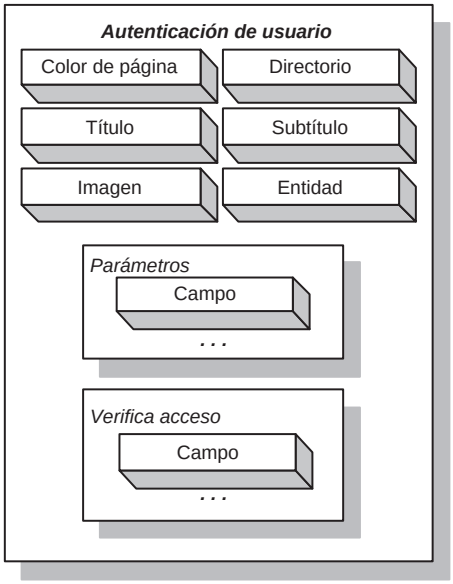

Figura 4.8: Estructura del esquema de autenticación de usuario

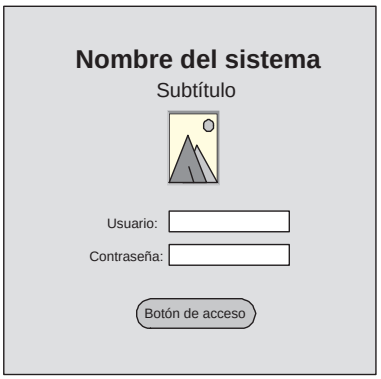

Figura 4.9: Estructura en Web de la autenticación de usuario

En la figura 4.9 se muestra la estructura que tendrá la página Web que se generará a partir de la descripción de autenticación de usuario. Básicamente son los parámetros de interfaz de la descripción de autenticación de usuario los que pueden visualizarse en la página. Los demás parámetros como *entidad y campos* son utilizados internamente por el generador de código para realizar las consultas.

4.3.1.4. Conexión a la Base de Datos

Esta descripción contiene los elementos necesarios para realizar una conexión a la Base de Datos. Ésta será de utilidad para realizar consultas y transacciones en el sistema. La figura 4.10 muestra la estructura del documento XML, los atributos que lo constituyen son el *driver* (manejador) mediante el cual se realizará la conexión, la dirección IP del servidor al cual se conectará (servidor que contiene la Base de Datos), el puerto mediante el cual se enlaza, con qué usuario y contraseña ha de conectarse y cual es la dirección IP del servidor donde la aplicación Web generada será instalada.

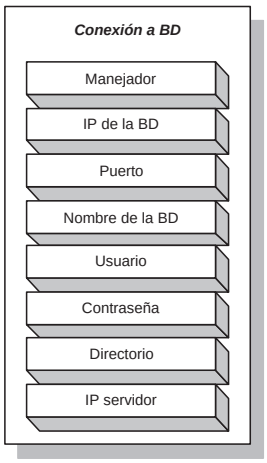

Figura 4.10: Estructura de la conexión a la Base de Datos

4.3.1.5. Directorios

La aplicación Web que será creada debe estar contenida en un árbol de directorios preeestablecido, por lo que ha de existir una descripción XML que genere dicho árbol de directorios. Lo que en esta descripción interesa detallar es el nombre de la carpeta que contendrá la aplicación y el nombre de la carpeta de cada uno de los módulos existentes en la misma. Por cada módulo definido se establecerá un árbol de directorios por omisión, pero esto se hace de forma automática dentro del generador de código. Esto será explicado en la sección 5.4.10 del capítulo 5. La figura 4.11 muestra como está estructurada esta descripción.

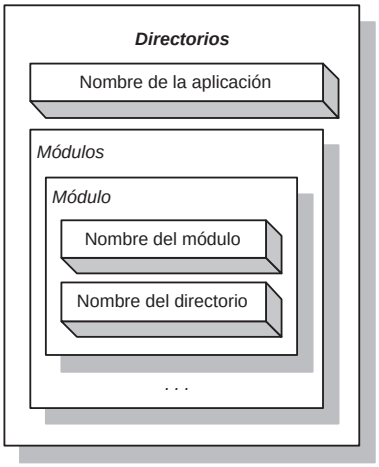

Figura 4.11: Estructura general de los directorios del sistema

4.3.1.6. Páginas

De manera general podemos definir el formato de una página JSP (*Java Server Page*) en secciones como se muestra en la figura 4.12:

- Importaciones. Se definen las importaciones propias de Java o bien de *drivers* externos.
- *Beans*. Se hace la invocación a los *beans* que sean necesarios para el funcionamiento de la página.
- Conexión a Base de Datos. Se realiza la validación de una conexión exitosa o fallida.
- Documentos XML. Se invocan los métodos necesarios para el llamado y la ejecución de una consulta. También se invocan métodos para la creación de documentos XML con información resultante de las consultas.
- Transformaciones. Invocación a transformaciones de documentos XML mediante una hoja de estilo.
- Cierre de conexión. Se asegura el cierre de conexión a la Base de Datos.

Con base en la estructura definida para una página JSP, se diseño la descripción de páginas del sistema. En la figura 4.13 se muestra como está estructurado el documento. Para la generación de código de una página JSP son necesarios datos tales como el nombre de la página, la localización (directorio de la aplicación Web en el cual estará ubicado el código generado de dicha página) y los *beans* que la página va a usar. Para cada *bean* que este definido en una página se debe de especificar su identificador y la clase de la cual provienen (ruta donde se encuentran ubicados los *beans*).

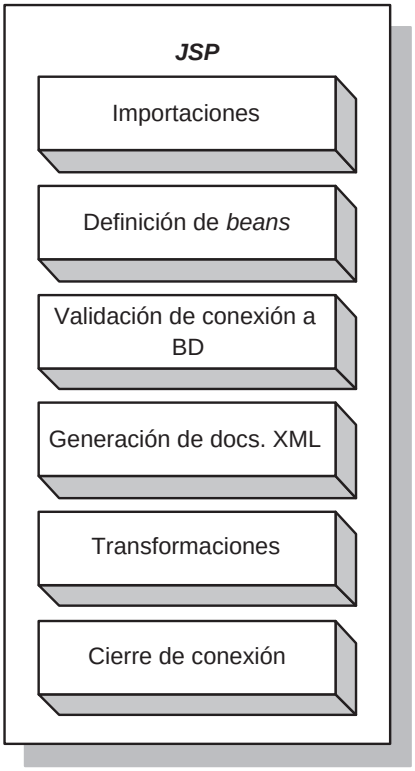

Figura 4.12: Estructura general de un JSP

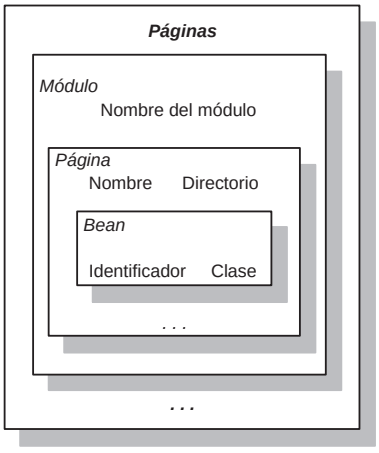

Figura 4.13: Estructura general de una página

4.3.1.7. Catálogos

En esta descripción se realiza el detalle de todos los Catálogos existentes en el sistema. El diseño esta hecho para ser la única página de catálogo de modo que los parámetros que recibe sean variables y pueda desplegar datos de cualquier catálogo. Para que funcione de esta manera se deben definir parámetros como se muestran en la figura 4.14, básicamente estos parámetros serán de utilidad para la generación automática de la consulta referente al catálogo.

Los parámetros que han de definirse son: entidad de la que proviene el catálogo (tabla de la Base de Datos que hace referencia a este catálogo), directorio donde se ubicará el código generado de la página referente al catálogo, parámetros de diseño (título, color de letra, tamaño de letra), campos que se desea aparezcan en el catálogo como información a desplegar (elementos encabezado), las condiciones necesarias para el despliegue de información del catálogo y el número de filas de información que se podrá visualizar (se hizo el diseño para que el catálogo se muestre por paginación).

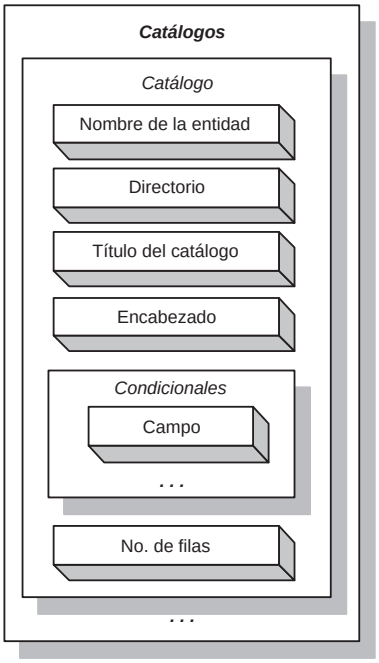

Figura 4.14: Estructura general de un Catálogo

En la figura 4.15 se muestra la estructura que tendrá un catálogo en una página Web. Se pueden observar los parámetros de diseño mencionados y los botones para avance y retroceso de información. Como se observa, se puede hacer la búsqueda de información para filtrar resultados. Para visualizar los resultados se cuenta con un grupo de elementos encabezado, donde se podrá elegir algún dato de la primera columna (liga) para determinar que ese dato es el que se desea seleccionar.

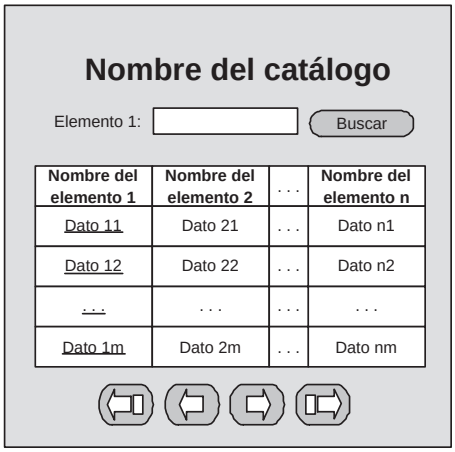

Figura 4.15: Estructura de la página Web de un Catálogo

4.3.1.8. Entidades

En esta descripción se hace la definición de cada una de las entidades involucradas, sean geométricas o no. De manera general se especifica el nombre descriptivo de la entidad, la tabla de la cual proviene esa entidad, el directorio donde se alojará el código generado y el tipo de entidad (geométrica, catálogo o nominal). Para cada entidad se define una lista de propiedades. Cada propiedad tiene ciertos atributos que describen cada una de las mismas. Dichos atributos son:

- nombre descriptivo de la propiedad o campo de la entidad,
- tipo de la propiedad o campo (caracter, entero, geométrico, etc.),
- longitud del campo (este atributo es opcional pues se coloca si el tipo de propiedad anterior es caracter),
- nombre de la columna a la que esta asociada esta propiedad,
- etiqueta para indicar si la propiedad es una llave primaria,
- etiqueta para indicar si la propiedad es una llave foránea,
- identificador de la propiedad o campo,
- etiqueta para indicar si la propiedad es única,
- etiqueta para indicar si la propiedad puede ser nula.

De esta manera, quedan completamente definidas cada una de las entidades existentes en el sistema, por lo que podemos encontrar las relaciones existentes entre éstas mediante los atributos que se han definido. La figura 4.16 muestra de manera gráfica la estructura de esta descripción.

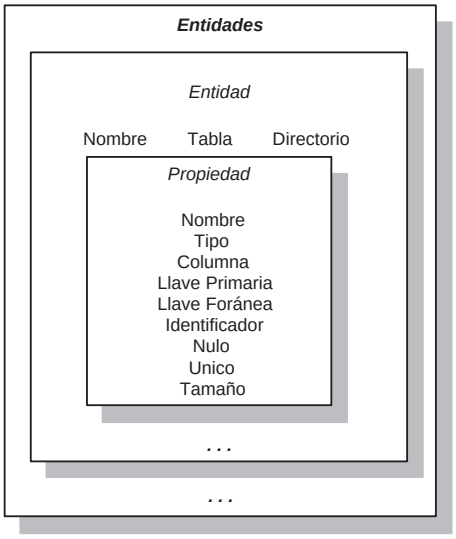

Figura 4.16: Estructura general de una entidad del sistema

4.3.1.9. Interfaz

Esta descripción contiene el detalle de las interfaces principales de cada uno de los módulos identificados en el sistema. Por cada interfaz se define el nombre de la misma, el directorio donde se va a colocar el código generado de este archivo y el color general de la interfaz. Cada interfaz esta compuesta por tres secciones:

- Encabezado de página. Esta sección, como su nombre lo indica, es un encabezado de la página, donde generalmente se muestra el nombre del módulo, algún logotipo y la fecha del sistema. Se definen las dimensiones del encabezado de la página Web, el título (incluyendo tipo, tamaño y color de letra), alguna leyenda opcional, la hora, la fecha y si el encabezado contendrá algún logotipo.

- Menú. Esta sección contiene el menú principal del módulo del cual se esta definiendo la interfaz. Únicamente se detallan las dimensiones de esta sección ya que la descripción de los menús se hace en un archivo de descripción a parte.

- Submenú. Esta sección es similar a la sección de menú, sólo que la descripción que se realiza es para submenús. En la subsección 4.3.1.10 se explica la diferencia entre la descripción de menú y submenú

- Contenido. Esta sección aunque no está explicita en la descripción, se considera parte de la interfaz, donde se desplegarán los formularios y mapas del sistema.

La figura 4.17 muestra la estructura de la interfaz para cada módulo registrado en el sistema y la figura 4.18 muestra la estructura básica de la interfaz en una página Web.

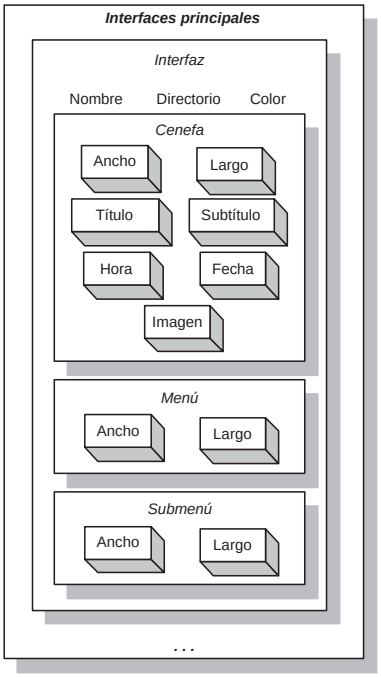

Figura 4.17: Estructura general de una interfaz principal del sistema

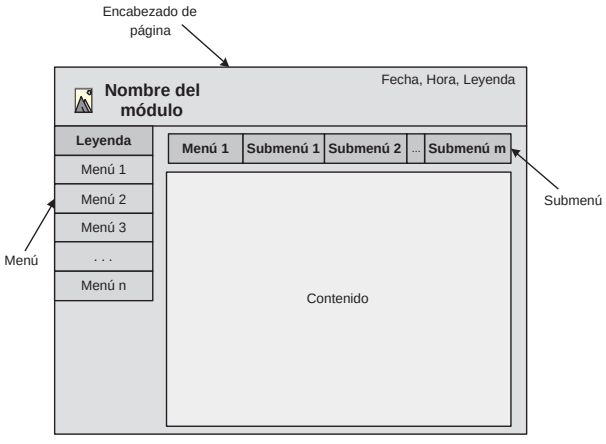

Figura 4.18: Organización de una interfaz principal del sistema

4.3.1.10. Menú y Submenú

La explicación de esta descripción es común tanto para menú como para submenú. La descripción menú se realiza para cada uno de los módulos identificados. La descripción submenú se realiza para cada entidad de cada módulo identificado. Cada descripción de menú o submenú contiene los siguientes parámetros:

- Parámetros para el estilo de los textos de cada menú (fuente, color de fuente y tamaño de la fuente).
- Encabezado que es un texto que encabezará todo el menú.
- Menú, donde cada menú definido contiene un texto que lo describe:
  - un color,
  - un enlace (hacia una nueva página),
  - el nombre de la sección donde aparecerá la nueva página,
  - un texto que describe la acción del menú (visible en el menú cuando el *mouse* está encima del mismo),
  - el tipo de menú al que se refiere (menú, submenú o espacio en blanco).

De esta manera, se pueden definir tantos componentes de menú o submenú como se requieran para cada descripción de menú de cada módulo del sistema. La figura 4.19 muestra la estructuración de la descripción de menús y submenús. En la figura 4.18 se visualiza el menú y el submenú dentro de una página de interfaz completa. Aunque la descripción es la misma para menús y submenús, la forma en que se despliegan en la página es diferente debido a la forma en que se transforma una descripción y otra.

4.3.1.11. Comportamiento

Esta es la última de las descripciones y la más compleja, ésto debido a que involucra todo el comportamiento que tendrá el sistema. Para hacerlo más modular, el comportamiento se describe para cada módulo identificado en el sistema, luego en cada módulo se describe para cada entidad que pertenezca a ese módulo. La figura 4.20 muestra la estructura general de la descripción de comportamiento.

Cada módulo de comportamiento tiene un nombre, un directorio (en el cual se almacenará el código que se genere a partir de esta descripción) y el nombre de la entidad geográfica que esta relacionada a este módulo. Luego cada entidad contiene el nombre de dicha entidad y su descripción.

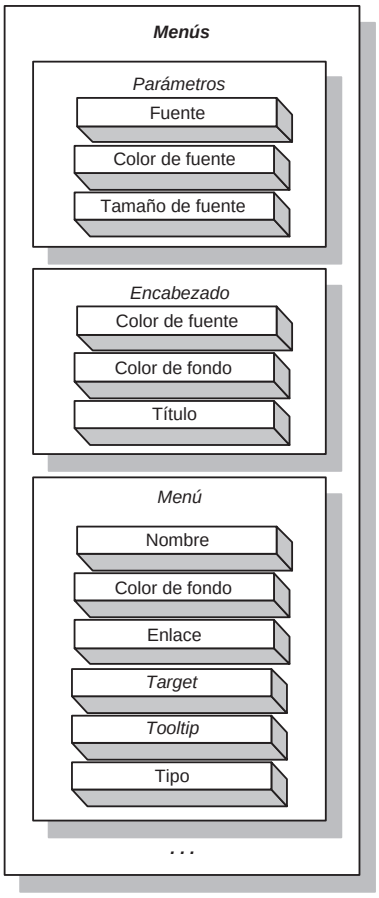

Figura 4.19: Estructura general de menú o submenú

Para cada entidad se realiza una descripción dividida en modos de comportamiento que básicamente son funciones que se realizan dentro del sistema. Los modos que se han identificado son:

- Modo de consulta. Se realizan consultas de la entidad a la que se esté haciendo referencia.
- Modo de inserción. Se realiza la inserción de una entidad a la que se esté haciendo referencia.
- Modo de eliminación. Se elimina una entidad específica.
- Modo de actualización. Se modifican los datos pertenecientes a una entidad.

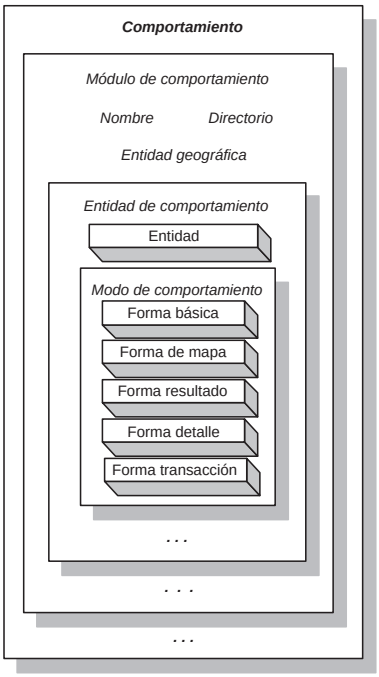

Figura 4.20: Estructura general de la descripción de comportamiento

Para hacer la descripción de cada modo de comportamiento se definieron módulos de descripción llamados *formas*. Cada una de estas formas contiene una descripción de formularios, acciones (navegación entre páginas) y definición de parámetros que se envían en los formularios. Cada forma tiene un nombre que se utilizará para definir el nombre del archivo (una página tipo *Java Server Page*) que contendrá el código generado de esta descripción. Los diferentes tipos de formas se describen a continuación:

1. **Forma básica.** Contiene la descripción de los elementos de un formulario tales como (etiquetas, cajas de texto, listas desplegables, botones, radio botones o botones para carga de archivos). Principalmente esta forma se utiliza en todos los modos de comportamiento como base para la búsqueda de información a consultar, insertar, eliminar o actualizar.
   En la figura 4.21 se muestra la estructura de la forma básica. La definición de elementos del formulario se realiza por renglones, es decir, se definen todos los elementos necesarios por fila como si tuvieramos una tabla. Generalmente, en la última fila se describe un botón que invoca a una nueva página. La ubicación de la forma básica en la página Web se muestra en la figura 4.22. Cuando se han definido todos los elementos del formulario, se definen los parámetros que han de ser enviados a la página siguiente (si es que ésta existe). Cada parámetro se describe como se explicó en la sección 4.3.1.1, tomando en cuenta solamente la definición del campo, es decir, sin el nombre descriptivo y sin la etiqueta de enlace. De esta manera están bien definidos los parámetros (de que entidad provienen, si son llaves primarias o foráneas y con

qué entidad se relacionan).

2. **Forma de mapa.** Esta forma describe la consulta y el despliegue de mapas. La figura 4.23 muestra cómo esta estructurada una forma de mapa. Esta dividida en tres secciones:

   - Datos. En esta sección se define la entidad geográfica sobre la cual se va a graficar y también se define una forma básica como la descrita en el punto 1. En esta forma básica se define el formulario con los parámetros que se utilizarán para hacer la búsqueda de información para el despliegue del mapa.
   - Mapa. En esta sección se definen las características del mapa, es decir, las dimensiones del área donde se va a graficar el mapa y los elementos encabezado (información geográfica que se desea graficar).
   - Datos de resultado. En esta sección se definen las características de la página que despliega información acerca de una entidad específica. Se describe el título de la página, el cólor del encabezado de la tabla en la que se desplegará la información, el color de cada fila de la tabla y por último los elementos encabezado (información nominal que se desea consultar acerca de la entidad geográfica).

   La figura 4.24 muestra como se visualiza la forma de mapa en una página Web.

3. **Forma de resultado.** Esta forma describe los resultados que fueron arrojados al realizar una consulta con ayuda de la forma básica. Los parámetros que en esta forma se definen son:

   - título de la página (con atributos de color de letra, tipo y tamaño de letra)
   - color de encabezado y color de fila de la tabla donde se desplegarán los resultados
   - los elementos encabezado (información nominal que se desea consultar)
   - pre acciones
   - post acciones

   En la figura 4.25 se muestra la estructura de esta forma y en la figura 4.26 se muestra la visualización en la Web. En este caso, no existen pre-acciones y la post-acción esta representada por la liga en cada dato de los elementos encabezado.

4. **Forma de detalle.** Esta forma representa el detalle de resultados de una entidad específica que fue seleccionada en la forma de resultado. Tiene la misma estructura que la forma de resultado. La figura 4.27 muestra la visualización en la Web.

5. **Forma de transacción.** Esta forma representa las transacciones que pueden hacerse en la Base de Datos. Se describe el tipo de transacción que ha de hacerse (inserción, eliminación o actualización) y de manera opcional puede contener una forma básica (la forma básica se usa cuando es una inserción o una actualización). En la figura 4.28

se muestra la estructura de una forma de transacción y en la figura 4.29 se muestra la visualización en la Web.

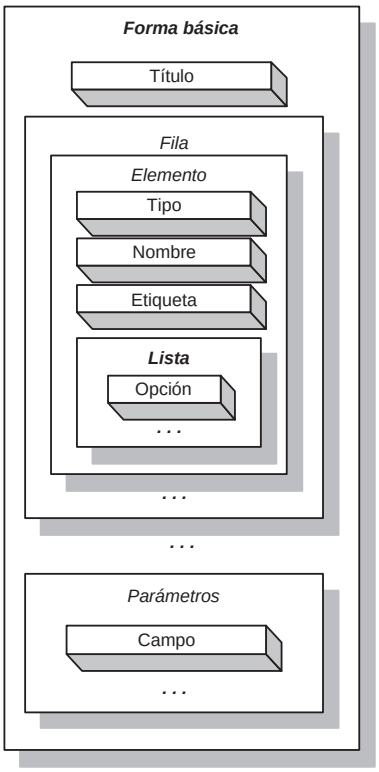

Figura 4.21: Estructura general de una forma básica

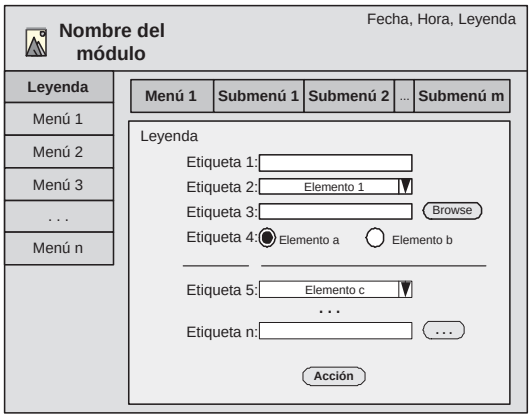

Figura 4.22: Visualización de una forma básica en la Web

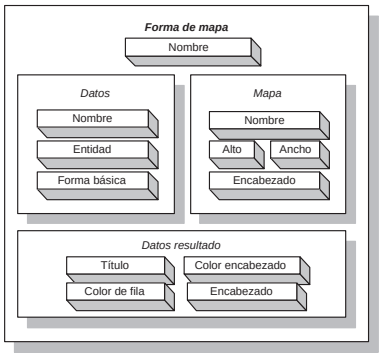

Figura 4.23: Estructura general de una forma de mapa

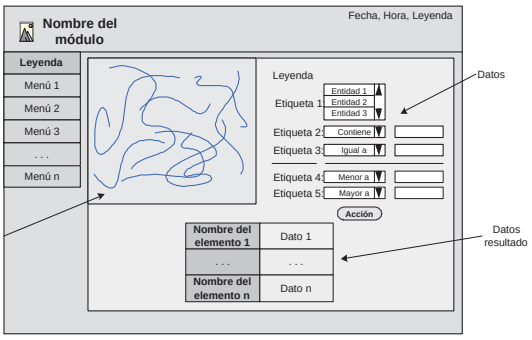

Figura 4.24: Visualización de una forma de mapa en la Web

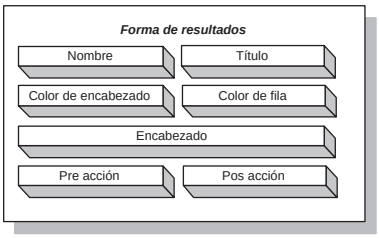

Figura 4.25: Estructura general de una forma de resultados

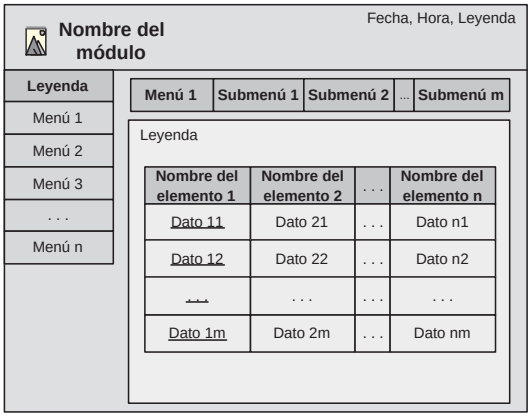

Figura 4.26: Visualización de una forma de resultados en la Web

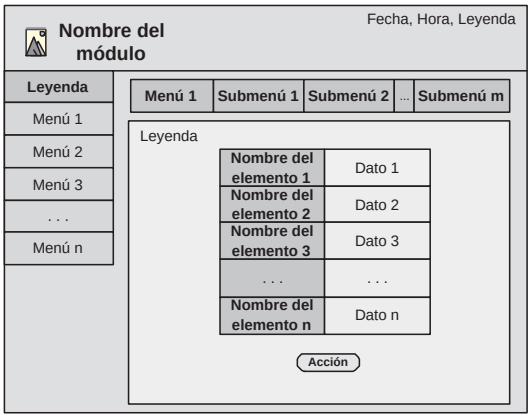

Figura 4.27: Visualización de una forma de detalle en la Web

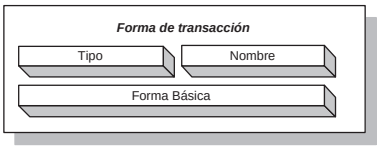

Figura 4.28: Estructura general de una forma de transacción

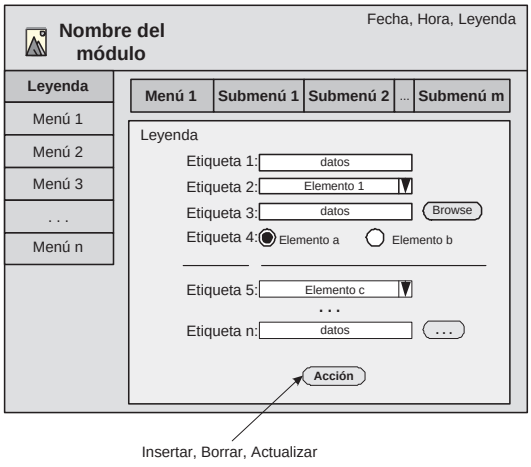

Figura 4.29: Visualización de una forma de transacción en la Web

Cada modo de comportamiento puede tener una o más formas para conseguir la funcionalidad requerida. En el capítulo 6 se muestra un caso de estudio donde se ejemplifica el uso de formas en los diferentes modos de comportamiento.

Las estructuras que hasta el momento se han explicado, son los esquemas de descripción, sus instancias y lo que representan (entidades de la Base de Datos, campos de las entidades, páginas, enlaces). De esta manera, el conjunto de documentos conforman la descripción del sistema que es la entrada a la aplicación generador de código para obtener la aplicación Web completa.

4.3.2. Generador de código

La aplicación generador de código es la encargada de leer las descripciones y convertirlas en una aplicación que contenga código ejecutable. Con ayuda de un Servidor de aplicaciones Web como Tomcat, la aplicación generada automáticamente podrá visualizarse en la Web. El generador de código está completamente desarrollado de lenguaje Java

El generador de código esta constituido por módulos que interactuan entre sí para lograr su objetivo. La figura 4.30 muestra los módulos mencionados y de manera general la forma en la que están relacionados.

La figura 4.30 muestra tres secciones:

1. La sección **E/S XML** consta del manejador de documentos XML, se encarga de la lectura de documentos. De esta forma, las descripciones pueden leerse como cadenas o bien como documentos para su posterior procesamiento en el generador de código.

2. En la sección **Business class** se encuentran todas las clases que el generador de código utiliza para producir la salida. Generalmente son clases que van estrechamente relacionadas con las estructuras de descripción que se mencionaron en la subsección anterior. Cada clase tiene sus métodos para asignar y obtener los valores de sus propiedades. Dichos valores son la información contenida en las descripciones. La tabla 4.1 muestra las clases que se identificaron. Para una mayor referencia, el diagrama de clase sintetizado de la figura 4.31 contiene la descripción completa del generador.

3. La sección **Generador de código** es la implementación de la aplicación en sí. Dentro de esta sección se encuentran módulos relacionados que interactúan para generar el código requerido.

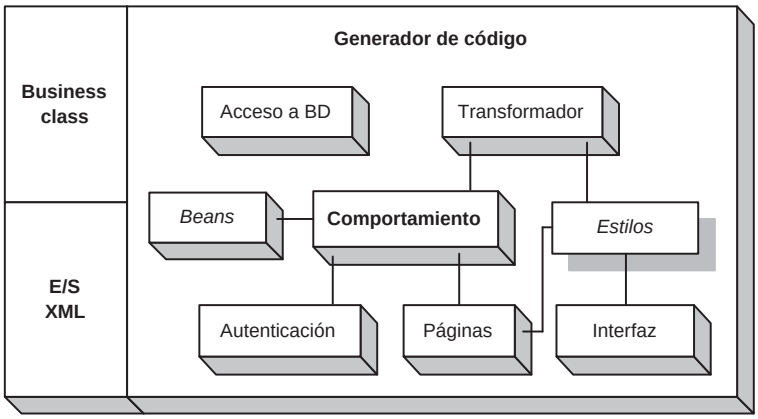

Figura 4.30: Estructura general del generador de código

Clases del generador de código	
Acción	Forma básica
Modo de comportamiento	Módulo de comportamiento
Encabezado de página	Catálogo
Conexión a Base de Datos	Forma de detalle
Dimensión	Entidad
Comportamiento de Entidad	Campo
Fuente	Marco principal
Elemento encabezado	Localización
Autenticación	Forma de mapa
Menú	Página
Propiedad	Punto
Forma de resultado	Forma de transacción
Bean	Documento
Tabla	

Tabla 4.1: Clases del generador de código

El diagrama de clases sintetizado del generador de código de la figura 4.31 muestra cada una de las secciones y los módulos de los que esta constituída toda la aplicación. Ésta se dividió en paquetes y cada uno de éstos contiene clases que implementan la funcionalidad del generador de código. En el capítulo 5 se explica la implementación de cada sección y se detalla el diagrama de clases.

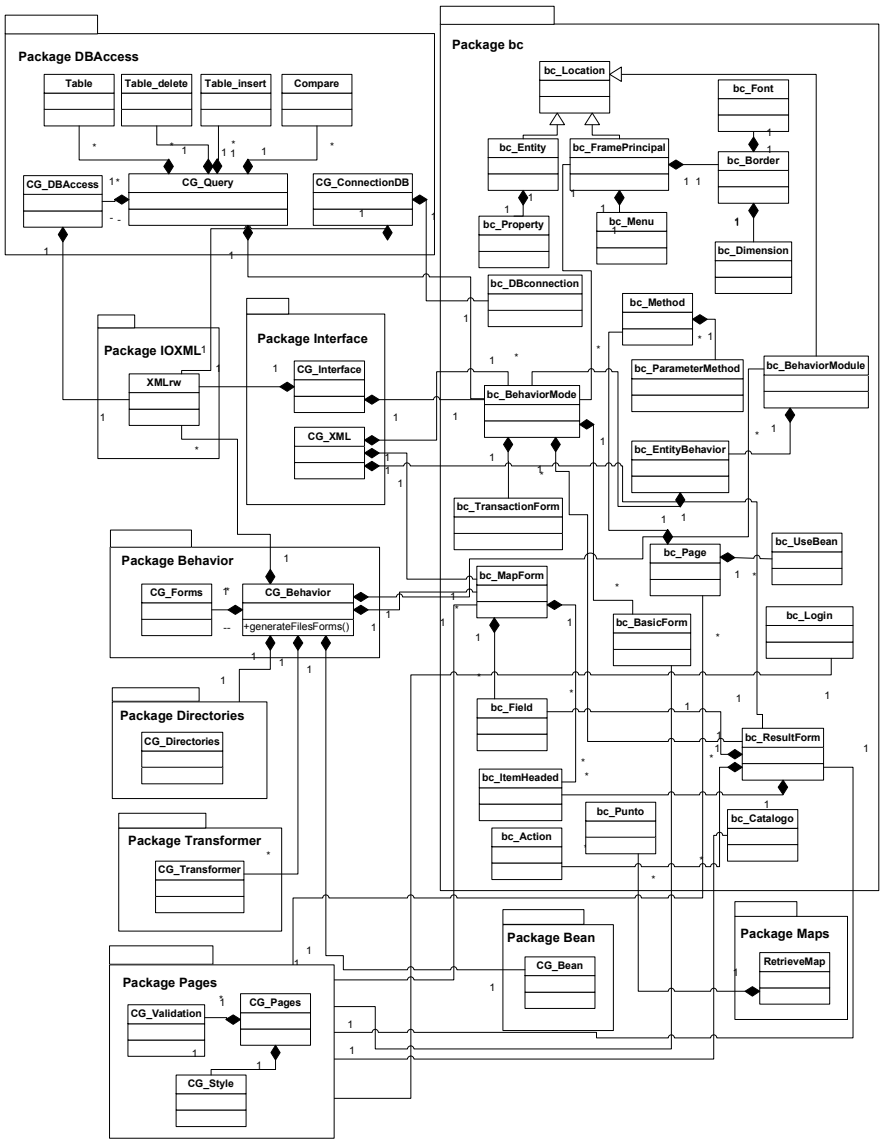

Figura 4.31: Diagrama de clases sintetizado

4.3.2.1. Funcionamiento del generador de código

Como se muestra en la figura 4.30, la sección generador de código se compone de varios módulos. Cada uno de éstos tiene una función específica (generar archivos de conexión a Base de Datos, generar páginas JSP, generar archivos de *beans* de consulta, etc.) y puede auxiliarse de otros módulos para llevar a cabo su tarea, incluso se auxilia de las funciones que realizan las otras secciones para así completar su trabajo.

Independientemente de la tarea de cada módulo, el objetivo en común es generar código. Para que esto se cumpla, el generador contiene plantillas preestablecidas para generar páginas JSP, métodos y *beans*, entonces el generador, en base a las plantillas, completa la información proveniente de las descripciones para generar archivos que puedan ser ejecutados. Sin embargo, el trabajo del generador no es simplemente rellenar plantillas, sino generar las consultas, transacciones (inserciones, actualizaciones, elminaciones), generar métodos que generen otros documentos XML y generar las consultas geográficas así como la generación de documentos SVG para visualizar los mapas en una página JSP. En seguida se explica la función de cada módulo.

1. **Módulo Acceso a Base de Datos**. Es el que se encarga de generar el método de conexión a la Base de Datos, el método de ejecución de una consulta y el cierre de la conexión. Se auxilia de la clase *Conexión a Base de Datos* de la sección de entidades del generador. El objeto instanciado de esta clase contiene toda la información necesaria proveniente de la descripción.

2. **Módulo de *Beans***. Este módulo genera código para *beans* que cumplen una tarea diferente en el generador:

   - *Beans* de entidad. Se encarga de generar *beans* de las entidades del sistema. Existe una descripción de entidades del sistema cuya información es asignada a la clase *Entidad* y de aqui se genera el *bean* de cada una de las entidades existentes en la descripción.
   - *Beans* de consulta. Crea los *beans* de generación de consultas. Para construir este código no hay una descripción precisa, en realidad se genera a partir de algunos elementos de varias descripciones. Una consulta puede generarse a partir de los parámetros de una forma básica y de los elementos encabezado de una forma de resultados.
   - *Beans* de transacción. Crea los *beans* con los métodos para generar las cadenas de transacción a partir de un objeto de la clase *Forma de transacción* que contiene la información de la descripción.
   - *Beans* de Procedimientos Almacenados. Genera el *bean* de procedimientos almacenados para las transacciones del sistema. Se auxilia del *bean* de transacción pero en estos *beans* se generan procedimientos que serán cargados en la Base de Datos. A su vez, se generan los métodos para invocar dichos procedimientos desde la aplicación Web.

- *Beans* de documentos XML. En algunos casos, por ejemplo en el de una actualización, es necesario desplegar un formulario con datos existentes (resultado de una consulta) para que el usuario los modifique. La descripción correspondiente al modo de comportamiento *actualizar* contiene una forma básica para desplegar el formulario con la información a modificar. Es necesario entonces que el *bean* contenga un método que genere un documento XML de tipo forma básica pero con información que proviene del resultado de la consulta de la entidad del sistema que el usuario desee actualizar. Otra posibilidad es la generación de mapas, en este caso el *bean* tiene un método que genera código SVG con información resultado de una consulta geográfica. Este *bean* es de utilidad también para generar el documento XML que describe a los catálogos del sistema.

3. **Estilos**. No son un módulo del generador de código, son archivos auxiliares que realizan la transformación de *formas* a código HTML, XHTML [26] o SVG. Las transformaciones pueden realizarse de formas diferentes. En secciones posteriores se explica con más detalle la estructura de los archivos de estilo y los diferentes procedimientos de transformación.

4. **Módulo de interfaz**. Este módulo tiene la función de generar las páginas (JSP) que conforman las interfaces principales. Para lograrlo, de la descripción de interfaz se obtiene un arreglo de objetos instanciados de la clase de tipo *Marco principal*, este tipo de clase contiene una clase encabezado de página, y una clase menú conforme a la descripción. Con este arreglo de objetos podemos generar una página para el encabezado de página, una para el menú y una para el contenido central. La figura 4.18 muestra como está estructurada una interfaz principal. Para lograr la generación de páginas es necesario el uso de estilos.

5. **Módulo de autenticación de usuario**. Este módulo se encarga de generar las páginas para la autenticación del usuario. La descripción mostrada en la figura 4.8 contiene la información necesaria para crear un objeto de la clase *Autenticación* y a partir de este objeto generar el código requerido (páginas JSP y métodos de consulta). La autenticación de usuario permite a éste, el acceso a uno de los módulos del sistema.

6. **Módulo de páginas**. Este módulo se encarga de generar páginas JSP en base a plantillas y a la descripción de páginas contenida en un arreglo de objetos de la clase *Página*. De esta manera, el módulo recibe partes de código que hay que insertar en las plantillas para crear una página completa. El módulo de páginas es auxiliar del módulo central de comportamiento, ya que por cada *forma* que encuentre en la descripción, se invoca a este módulo para la generación de su página correspondiente. A su vez este módulo se auxilia de los *estilos* para realizar transformaciones.

7. **Módulo transformador**. Este módulo se encarga de realizar una transformación a partir de una sección de descripción y un estilo. Principalmente este módulo es utilizado para realizar transformaciones de *formas básicas* a código HTML y XHTML.

8. **Módulo de comportamiento**. Es el módulo central que invoca a los métodos contenidos en los demás módulos. Desde éste módulo se hace la lectura de toda la descripción de comportamiento y se vacía en objetos de la clase *Módulo de comportamiento*, entonces el generador revisa cada objeto e invoca al módulo de páginas, al módulo de transformación y al módulo de *beans* para así obtener código de las páginas JSP, de beans de generación de consultas, de documentos XML y de transacciones.

En la figura 4.32 se muestra los tipos de archivos que se generan. Por una parte se genera código HTML contenido en páginas, generalmente proveniente de la transformación de *formas básicas* a través de un estilo. El segundo tipo de código es XHTML que va estrechamente relacionado con el tercer tipo, pues el código SVG generado a través de una transformación se embebe dentro de la página JSP con contenido de tipo XHTML. El cuarto tipo es código Java, es decir, todos los *beans* generados que ya han sido detallados en el punto 2 de la sección 4.3.2.1. El quinto y último tipo es un archivo de texto que contiene los procedimientos almacenados, este archivo es cargado a la Base de Datos para tener registrados dichos procedimientos y así la aplicación realice transacciones sin tener que ejecutarlas desde el cliente.

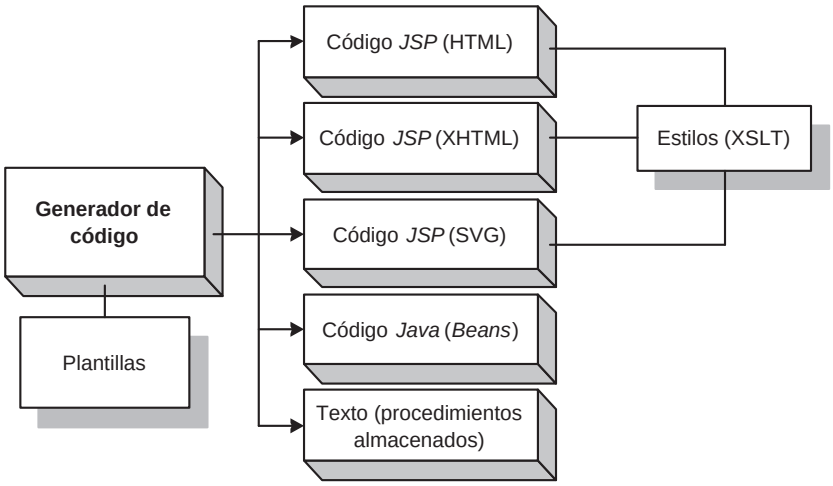

Figura 4.32: Tipo de código generado por la aplicación

Para una mejor organización del código generado, cada archivo se distribuye en carpetas previamente creadas como se muestra en la figura 4.33. Se crea una carpeta con el nombre de la aplicación Web, dentro de esa carpeta se crea una carpeta de imagenes y una carpeta WEB-INF con las librerías correspondientes, dentro de WEB-INF se crea la carpeta *classes* donde se crea la estructura de carpetas para alojar los *beans* generados. Además dentro de la carpeta con el nombre de la aplicación se crea una carpeta por cada módulo identificado en el sistema y por cada una de estas carpetas se crea una estructura para alojar las páginas JSP. Este paso no esta en ningún módulo del generador, se encuentra en un pequeño IDE que se detalla en el capítulo 5.

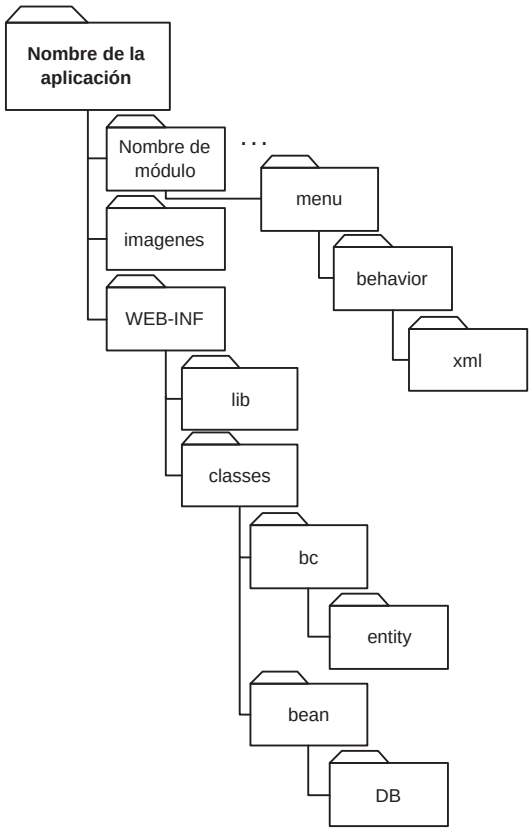

Figura 4.33: Distribución de directorios para el código generado

4.3.3. Aplicación final

El último módulo de la arquitectura diseñada es la aplicación final, es decir, el resultado que produce el generador de código. Al finalizar el proceso de generación se obtiene una carpeta con toda la aplicación Web generada y un archivo *.txt* con los procedimientos almacenados para las transacciones. Este archivo debe ser cargado en la Base de Datos para que las llamadas a los procedimientos almacenados de las transacciones funcionen desde la aplicación Web.

Finalmente, para que los *beans* generados trabajen sin problemas hay que compilarlos y así la aplicación Web estará lista para ser colocada en algún servidor de aplicaciones Web. Para *Tomcat* basta con colocar la carpeta ya compilada dentro de su directorio de aplicaciones *webapps* y reiniciar *Tomcat* para visualizar la aplicación Web en algún navegador. En la figura 4.34 se muestra el ambiente funcional de la aplicación Web generada.

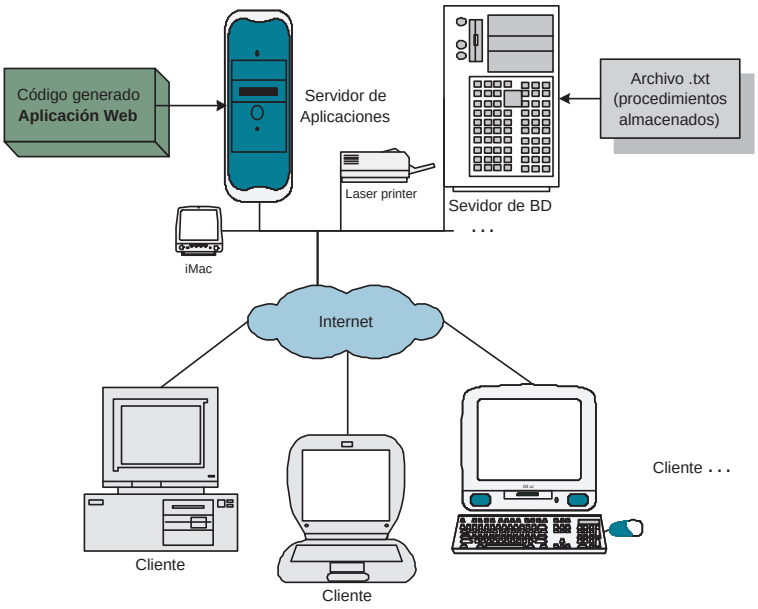

Figura 4.34: Ambiente funcional de la aplicación generada

4.4. Conclusiones

El diseño de la aplicación es fundamental para determinar el funcionamiento del generador de código. Durante la explicación de este capítulo se pueden observar características de diseño que van enfocadas a un ambiente Web. Cada descripción tiene el objetivo de generar una sección del código de la aplicación. Así, a partir de la descripción se cuenta con un modelo completamente definido para generar una aplicación Web, con las restricciones propias que se han determinado en el modelo. Estas restricciones se refieren a una estructura de interfaz fija y parcialmente parametrizable, modelado de comportamiento basado en módulos y con una estructura fija principalmente. El diseño representa la entrada a la siguiente fase de la aplicación: el generador de código.

Capítulo 5

Implementación del generador de código

Una vez presentado el diseño de la aplicación, en este capítulo se procede a especificar como se llevó a cabo la implementación completa de la misma. Inicialmente, se definen los esquemas y las instancias de los documentos de descripción. Posteriormente, se detallan las clases que fueron implementadas y como es que se relacionan para generar el código. Finalmente, se explica la creación de un pequeño IDE que es una interfaz para el manejo amigable de la aplicación.

5.1. Herramientas de desarrollo

Como se mencionó en la sección 4.3.1 del capítulo 4, los esquemas de documentos de descripción se desarrollan en XML-Schema y las instancias en XML. Por otro lado, Netbeans [36] es un IDE de software libre que provee un editor de esquemas y documentos XML. Netbeans es un entorno de desarrollo, una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. El generador de código esta implementado en Java (J2SE). Para el manejo de documentos XML dentro del generador se utilizó DOM4J [30] como una biblioteca incluída del proyecto, así como xTags para procesar documentos XML dentro de un JSP. Las clases creadas y las clases de las librerías interactúan entre sí para crear el código deseado.

Para el desarrollo del IDE del generador de código se utilizó *Swing* [11] que permite crear interfaces de forma sencilla, pues sólo se seleccionan los componentes desde una paleta integrada en la interfaz de Netbeans. En las siguientes secciones se detalla el desarrollo de las clases utilizadas en la aplicación.

5.2. Implementación de descripciones

Para cada uno de los diseños explicados en la sección 4.3.1 del capítulo 4 se implementó su esquema XML. Para generar las instancias de cada esquema es necesario considerar el caso de estudio, el cual se detalla en el capítulo 6. En la figura 5.1 se muestran los diferentes esquemas de los cuales está compuesta la aplicación. Por cada uno de éstos, se

pueden tener una o más instancias, ésto dependerá de los requerimientos definidos para la aplicación Web que se pretenda generar.

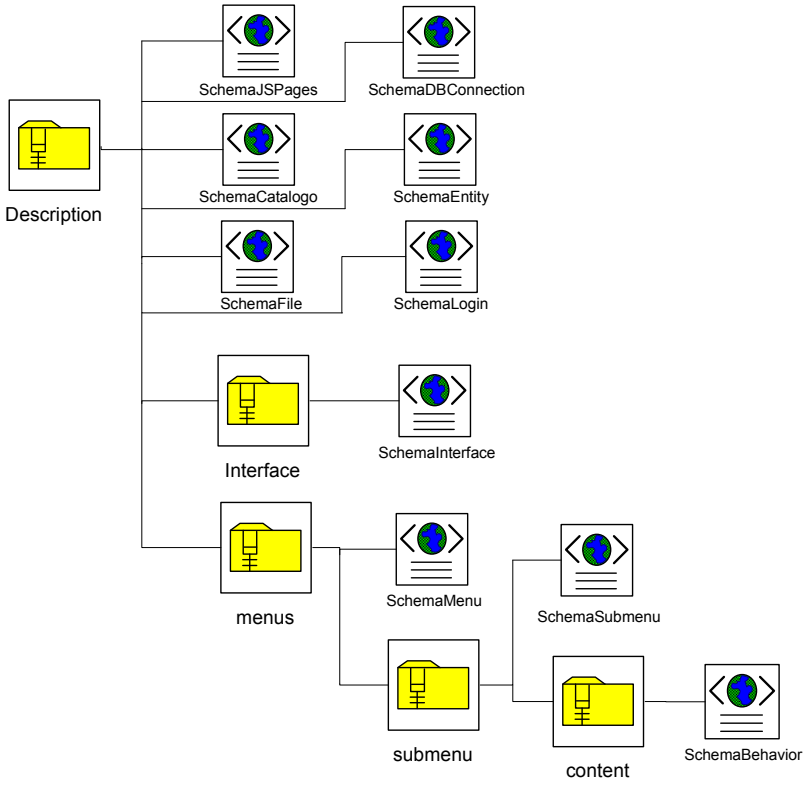

Figura 5.1: Esquemas de descripción

Si en los requerimientos se ha identificado un módulo y en éste se ha identificado un menú entonces se hace una instancia del esquema *schemaMenu* y una del esquema *schemaSubmenu*. Si se define otro módulo y a su vez otro menú, entonces se hacen nuevas instancias de los esquemas y así para cada módulo y menú identificado. En el caso del esquema *schemaBehavior* se hace una instancia por cada módulo identificado donde se incorpora toda la información de comportamiento del mismo. Para los demás esquemas se hace una sola instancia, por ejemplo, del esquema *schemaInterface* basta una sola instancia para plasmar la información referente a todas las interfaces de la aplicación Web.

En la figura 5.2 se muestra un fragmento del esquema *schemaMenu*. Dentro de éste se van definiendo los posibles elementos que tendrá una instancia. A su vez, se define el número de veces que podrá aparecer el elemento, el tipo de información que puede contener (string, integer, token, etc.).

La definición de elementos se va formando mediante agrupaciones de tipo simple o complejo. Un tipo complejo se utiliza cuando el elemento a definir tiene subelementos.

Los tipos simples son utilizados cuando se desea hacer una restricción sobre un elemento, por ejemplo, un elemento color que sea una cadena con un formato hexadecimal. En el esquema de la figura 5.2 hay una restricción para definir los colores de fuente y el color de fondo del menú. Se especifica que deseamos un formato hexadecimal para definir el color requerido. De esta manera, se pueden definir restricciones para fechas, direcciones IP y números de puertos. También dentro de un esquema se puede especificar que la información que contenga algún elemento se encuentre dentro de un grupo de valores determinado, por ejemplo, que el elemento <type> para una *forma de transacción* únicamente pueda contener los valores *insert, delete* y *update*.

Las ventajas que ofrece XML-Schema permiten definir la estructura de un documento y hacer restricciones sobre la información del mismo. Cada esquema de la figura 5.1 contiene ciertas restricciones que permiten instanciar un documento de descripción con la seguridad que está construido sintácticamente bien formado.

```
- <xsd:schema targetNamespace="http://xml.netbeans.org/examples/targetNS" elementFormDefault="qualified">
  - <xsd:element name="menus">
    - <xsd:complexType>
      - <xsd:sequence minOccurs="1" maxOccurs="1">
        - <xsd:element name="parametros" minOccurs="1" maxOccurs="1">
          - <xsd:complexType>
            - <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element name="fuente" minOccurs="1" maxOccurs="1" type="xsd:string"/>
              - <xsd:element name="colorfuente" minOccurs="1" maxOccurs="1">
                - <xsd:simpleType>
                  - <xsd:restriction base="xsd:string">
                      <xsd:pattern value="#[0-9A-Fa-f]{6}"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>
                <xsd:element name="tamafuente" minOccurs="1" maxOccurs="1" type="xsd:integer"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        + <xsd:element name="encabezado" minOccurs="1" maxOccurs="1"></xsd:element>
        - <xsd:element name="menu" minOccurs="1" maxOccurs="unbounded">
          - <xsd:complexType>
            - <xsd:sequence>
                <xsd:element name="nombre" minOccurs="0" maxOccurs="1" type="xsd:string"/>
              - <xsd:element name="colorfondo" minOccurs="0" maxOccurs="1">
                - <xsd:simpleType>
                  - <xsd:restriction base="xsd:string">
                      <xsd:pattern value="#[0-9A-Fa-f]{6}"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>
                <xsd:element name="link" minOccurs="0" maxOccurs="1" type="xsd:token"/>
                <xsd:element name="target" minOccurs="0" maxOccurs="1" type="xsd:string"/>
                <xsd:element name="tooltip" minOccurs="0" maxOccurs="1" type="xsd:string"/>
                <xsd:element name="clase" minOccurs="0" maxOccurs="1" type="xsd:string"/>
              </xsd:sequence>
              <xsd:attribute name="id" type="xsd:token" use="required"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

Figura 5.2: Esquema de descripción Menú

La figura 5.3 muestra una instancia del esquema *schemaMenu*. Se pueden definir elementos de menú con la estructura y las restricciones definidas en el esquema. Existe un

sólo elemento <parametros>, un <encabezado> y varios elementos <menu> que describen aspectos de la interfaz y navegación que tendrá este menú en una página Web.

```
- <menus>
  - <parametros>
      <fuente>arial</fuente>
      <colorfuente>#FFFFFF</colorfuente>
      <tamafuente>11</tamafuente>
    </parametros>
  + <encabezado id="I"></encabezado>
  - <menu id="0">
      <clase>espacio</clase>
    </menu>
  - <menu id="1">
      <nombre>Acueducto</nombre>
      <colorfondo>#CCCC00</colorfondo>
    - <link>
        centro.jsp?arriba=menu/submenuacueductocorrientes.jsp&abajo=msg.jsp
      </link>
      <target>centro</target>
      <tooltip>Acueducto</tooltip>
      <clase>menu</clase>
    </menu>
  + <menu id="2"></menu>
  + <menu id="3"></menu>
  + <menu id="4"></menu>
  + <menu id="5"></menu>
  + <menu id="6"></menu>
  + <menu id="7"></menu>
  + <menu id="8"></menu>
  + <menu id="9"></menu>
  + <menu id="0"></menu>
  + <menu id="10"></menu>
  + <menu id="0"></menu>
  + <menu id="11"></menu>
  + <menu id="0"></menu>
</menus>
```

Figura 5.3: Instancia de descripción Menú

El conjunto de instancias de descripción conforman la entrada al generador de código, de aquí la importancia de que su estructura e información sean permitidas en los esquemas. Podría pensarse que será tedioso generar el conjunto de instancias dentro de un editor de documentos XML, sin embargo, las descripciones serán generadas mediante un modelador de datos según el trabajo de tesis [40] que crea las descripciones sin que el usuario interactúe con el código XML directamente. El modelador es un proyecto de tesis que proporciona las herramientas necesarias para que el usuario introduzca la información de los requerimientos y ésta se transforme a documentos de descripción. El modelador define la estructura de la base de datos y las características de la aplicación Web que son requeridas por el usuario final. En otras palabras, el modelador genera un diseño para el desarrollo de una aplicación Web plasmado en descripciones XML. La ventaja radica en la facilidad para realizar cambios en el diseño y verlos automáticamente reflejados en las descripciones. Así, si varían las descripciones, el generador de código lee los cambios obteniendo una nueva versión del código generado con las modificaciones requeridas.

5.3. Uso de clases

Para la implementación del generador de código se utilizaron clases provenientes de *drivers* que fueron añadidos a la aplicación. La figura 5.4 muestra la jerarquía de interfaces y la figura 5.5 muestra la jerarquía de clases que es utilizada en el generador de código. Ambas jerarquías pertenecen al *driver* DOM4J y son integradas a la aplicación para manipular documentos.

La clase *DocumentHelper* es la encargada de crear un documento, a partir de éste, se tiene acceso a las interfaces de la figura 5.4. Los métodos permiten crear elementos, atributos, asignar información a éstos y crear elementos anidados. También pueden removerse elementos y sus elementos anidados.

Mediante éstas clases también es posible leer un documento XML que se encuentre almacenado físicamente. Una vez que se lee el documento, es posible recorrerlo y obtener la información del mismo mediante *Iteradores*. Se define un *iterador* para cada elemento y así se pueden recorrer sus elementos hijos. Lo mismo sucede con los atributos de cada elemento. Así, las interfaces *Element* y *Attribute* tienen métodos que permiten conocer el nombre del elemento o atributo y su contenido. Dependiendo de cómo se desee manejar, un documento puede ser manipulado como un tipo *Document* o bien como un *String*.

La API de Java tiene una definición de clases que permiten realizar transformaciones de documentos XML. La figura 5.6 muestra el diagrama de clases que permite integrar dicha funcionalidad al generador de código.

Una instancia de *TranformerFactory* puede ser usada para crear un objeto *Transformer* indicando con el método *newTransformer* el documento fuente de la transformación. Después basta con utilizar el método *transform* de la clase *Transformer* para realizar la transformación de un documento XML indicándolo como parámetro del método. El resultado se almacena en un objeto de tipo *StreamResult* que podrá contener un documento XML, texto plano, XHTML o cualquier documento de marcas.

5.4. Clases implementadas

La implementación del generador de código esta clasificada en paquetes para organizar las clases de acuerdo a la función que tengan. La figura 5.7 muestra la estructura de paquetes de la que consta el generador.

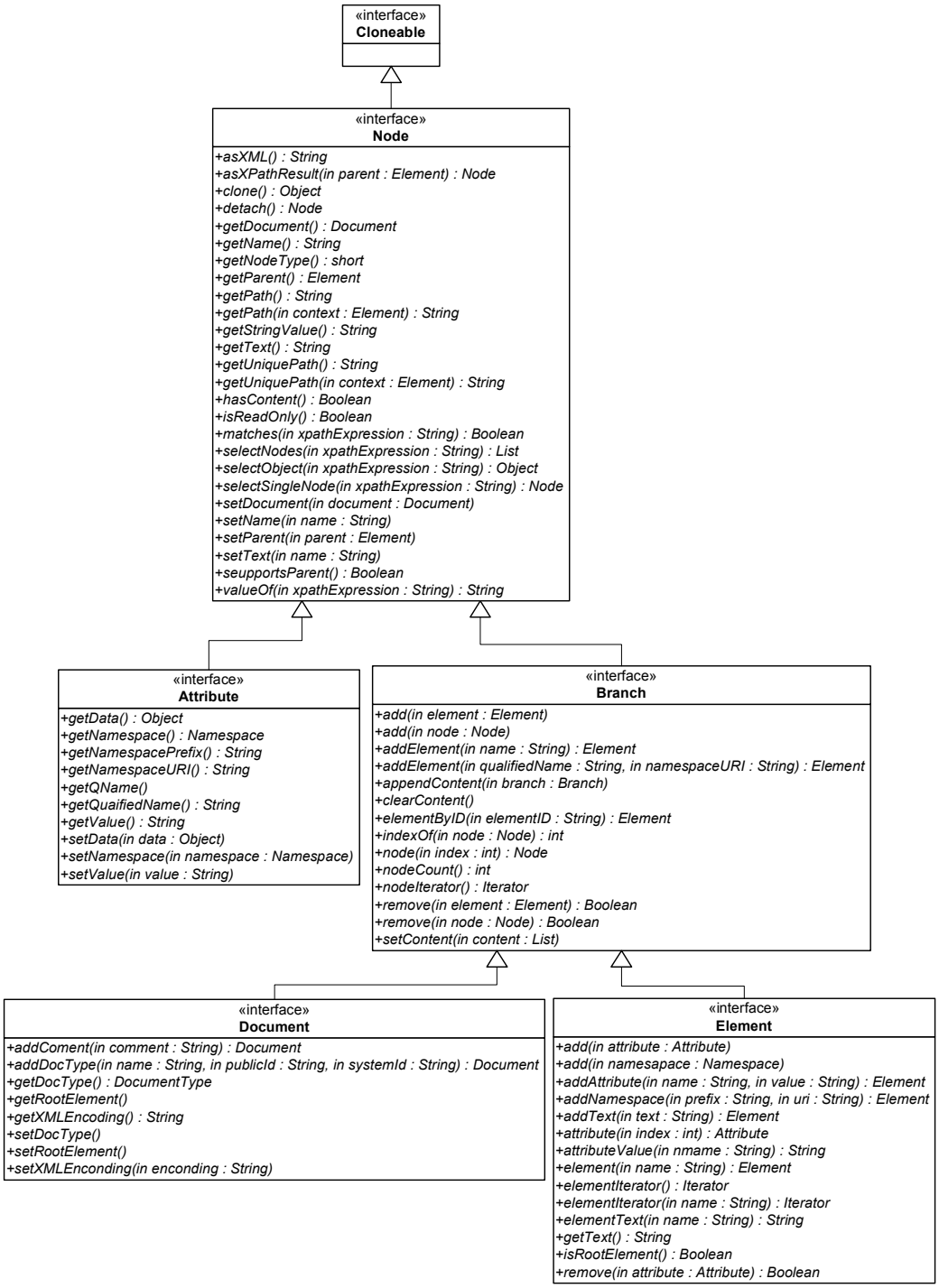

Figura 5.4: Jerarquía de interfaces de DOM4J

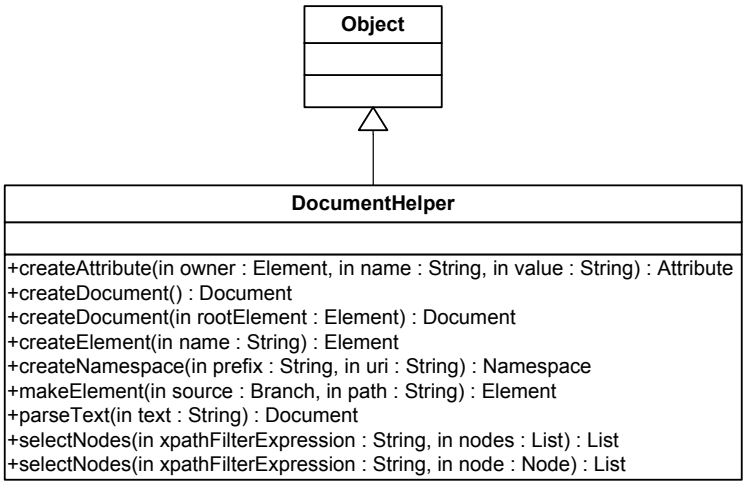

Figura 5.5: Clase *DocumentHelper* para el manejo de documentos XML

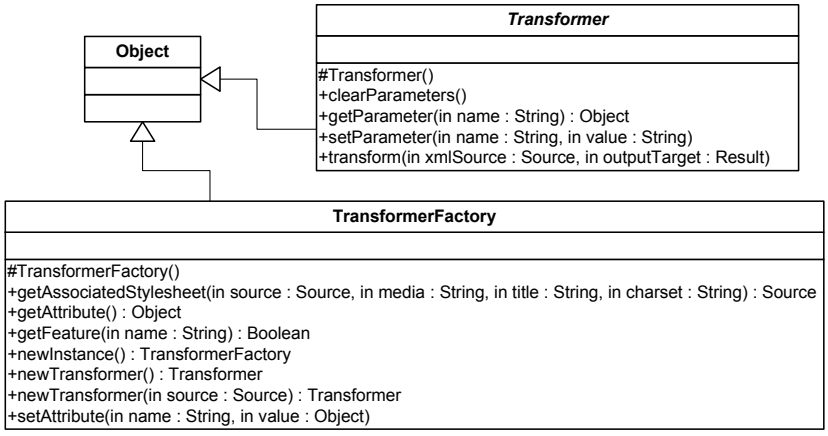

Figura 5.6: Diagrama de clases para transformación de documentos

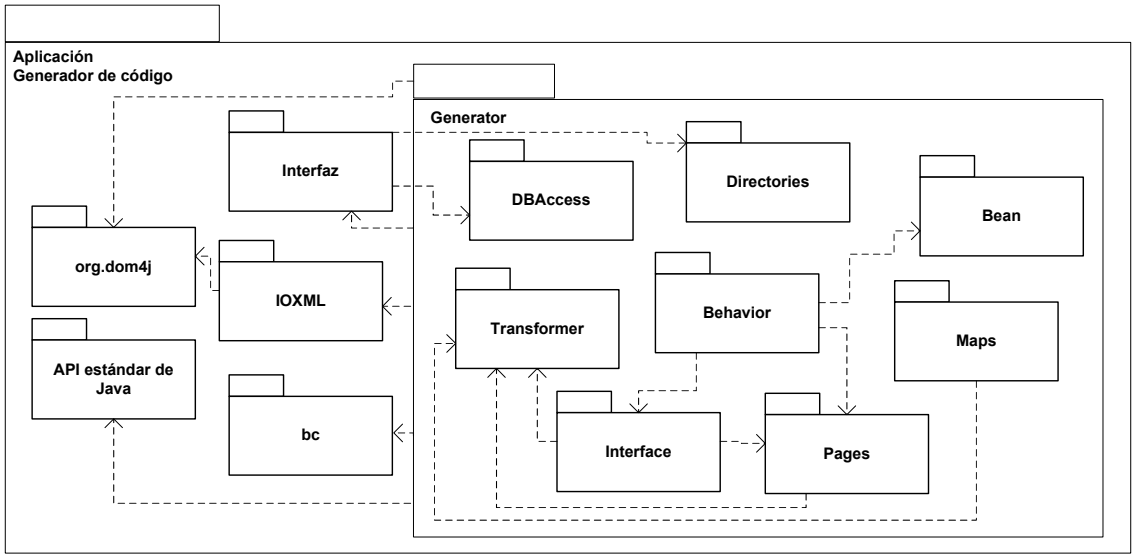

Figura 5.7: Diagrama de paquetes

El paquete *Generator* contiene a otros paquetes que implementan el desarrollo de la interfaz del sistema Web, la creación de *beans*, la generación de directorios, el acceso a la Base de Datos, la creación de páginas y todo el comportamiento del sistema. Este paquete implementa la mayor parte de la funcionalidad del generador de código. En el paquete *Interfaz* se encuentra el pequeño API desarrollado para generar el código mediante una interfaz amigable. En el paquete *IOXML* están las clases necesarias para la lectura de documentos XML y en el paquete *bc* se encuentran las entidades que maneja el generador. El paquete *org.dom4j* contiene las clases necesarias para la manipulación de documentos XML y por supuesto usamos la API estándar de Java para la programación del generador. En las siguientes secciones se detalla cada paquete, sus clases y como es que interactúan.

5.4.1. Entidades de la aplicación

El generador de código tiene una serie de entidades que permiten almacenar la información proveniente de las descripciones. Cada entidad es trasladada a una clase de la aplicación y cada una de éstas contiene todos los atributos de la entidad con sus respectivos métodos para asignación y recuperación de información. El paquete correspondiente a este conjunto de clases es *bc* (*bussines class*) de la figura 5.7. Se observa que tiene una relación con el paquete *Generator* pues la funcionalidad del generador de código se auxilia de las entidades del sistema para crear objetos que contengan la información proveniente de las descripciones. La figura 5.8 muestra el diagrama de clases de las entidades.

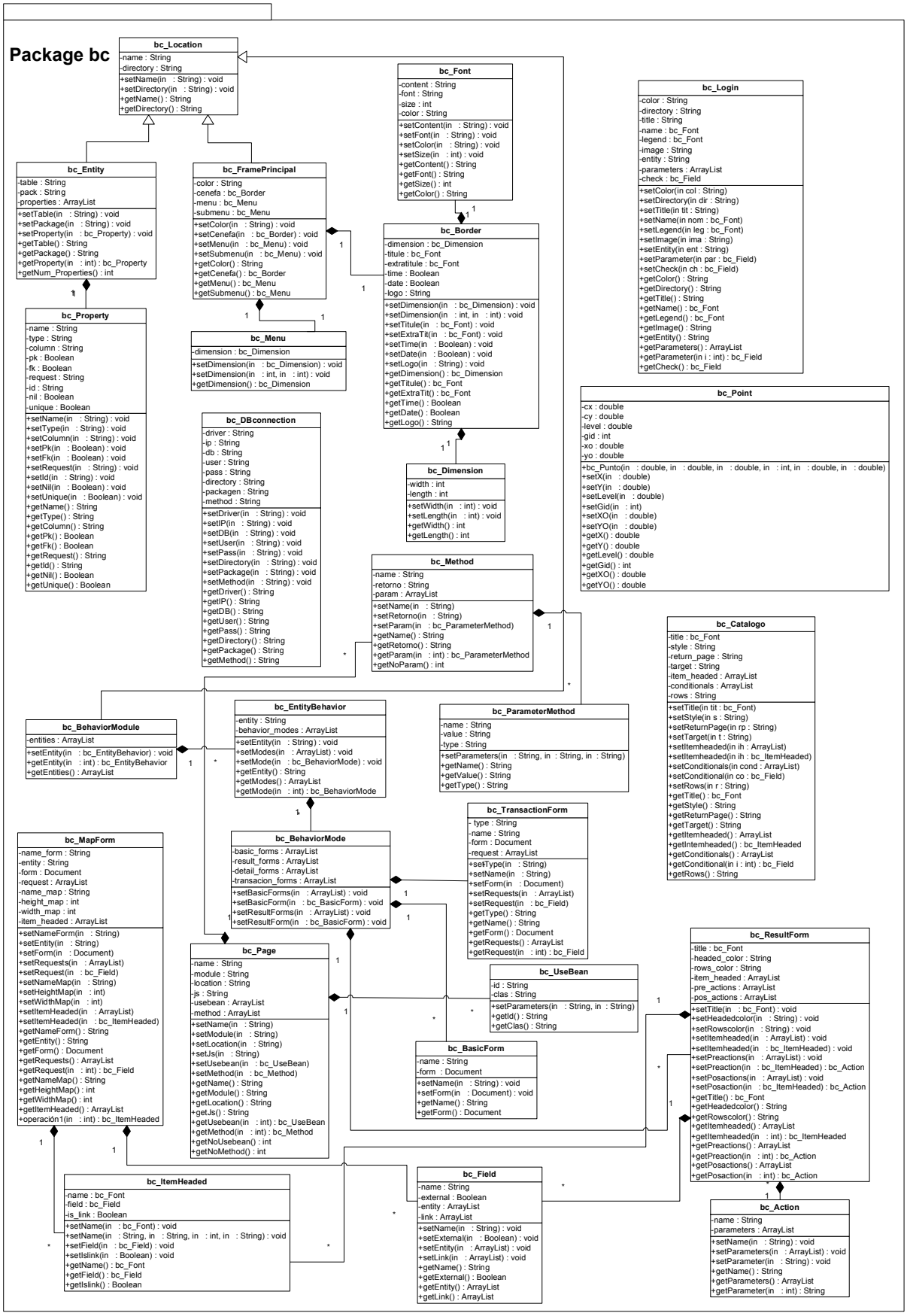

Figura 5.8: Entidades del generador de código

Cada una de las entidades prácticamente corresponde a una de las descripciones de la sección 4.3.1 del capítulo 4. Por ejemplo, la clase *bc_Entity* hereda de la clase *bc_Location*, con la finalidad de especificar mediante esta última el nombre del archivo que se generará y la localización física del mismo. La clase *bc_Entity* tiene como variables los atributos que aparecen en la descripción de la figura 4.16 del capítulo 4 y a su vez se compone de la clase *bc_Property* que contiene los atributos del elemento *Propiedad* de la misma figura. De esta manera, se crearon cada una de las clases que conforman las entidades y su correspondiente asociación con las descripciones.

Algunas clases se componen de otras, por lo que la clase *bc_BehaviorMode* está compuesta de las clases *bc_TransactionForm*, *bc_BasicForm*, *bc_ResultForm*, *bc_DetailForm* y *bc_MapForm* que constituyen las *formas* para modelar el comportamiento del sistema. De esta forma, se creó una clase por cada entidad identificada en las descripciones.

5.4.2. Acceso a Base de Datos

El paquete *DBAccess* contiene las clases necesarias para la creación de código que realice una conexión a la Base de Datos (figura 5.9). Para lograrlo, la clase *CG_ConnectionDB* contiene el método para la recuperación de la información de la descripción y creación del *bean* que realiza la conexión. La clase *CG_DBAccess* almacena la información de la descripción de entidades en objetos de tipo *bc_Entity* para su uso en la clase *CG_Query* y en otras clases de otros paquetes.

La clase *CG_Query* se encarga de la generación de consultas y transacciones en la Base de Datos. Esta clase esta muy relacionada con la clase *CG_Behavior* del paquete *Generator*, pues esta última le indica a la clase *CG_Query* la operación de la cual se desea generar código. En las siguientes subsecciones se detalla cada una de dichas operaciones.

5.4.2.1. Generación de transacciones

Con los métodos de la clase *CG_Query*, el generador de código puede crear *beans* de transacciones para que sean ejecutadas desde el cliente. También genera *beans* de llamadas a procedimientos almacenados y genera los procedimientos almacenados en sí. De esta manera, las transacciones se ejecutan en la Base de Datos y el proceso es más seguro. A partir de las descripciones podemos generar 4 procesos: consulta, inserción, eliminación y actualización.

1. Consulta. Antes de realizar cualquier operación de modificación de información en la Base de Datos se realizan consultas. La generación de consultas se hace a partir de *formas básicas*. Cada *forma básica* que se define contiene una serie de *parámetros* que definen los *campos* que están involucrados en la *forma*. Aquí intervienen las clases *bc_BasicForm* y un arreglo de objetos de tipo *bc_Field* que equivalen a los parámetros de la *forma básica*. Cada definición de *campo* (*bc_Field*) tiene la entidad de la cual proviene con lo que el generador puede instanciar objetos de tipo *Table* con información acerca de sus relaciones con otras entidades (Objeto *Tabla* mostrado en la figura 5.12).

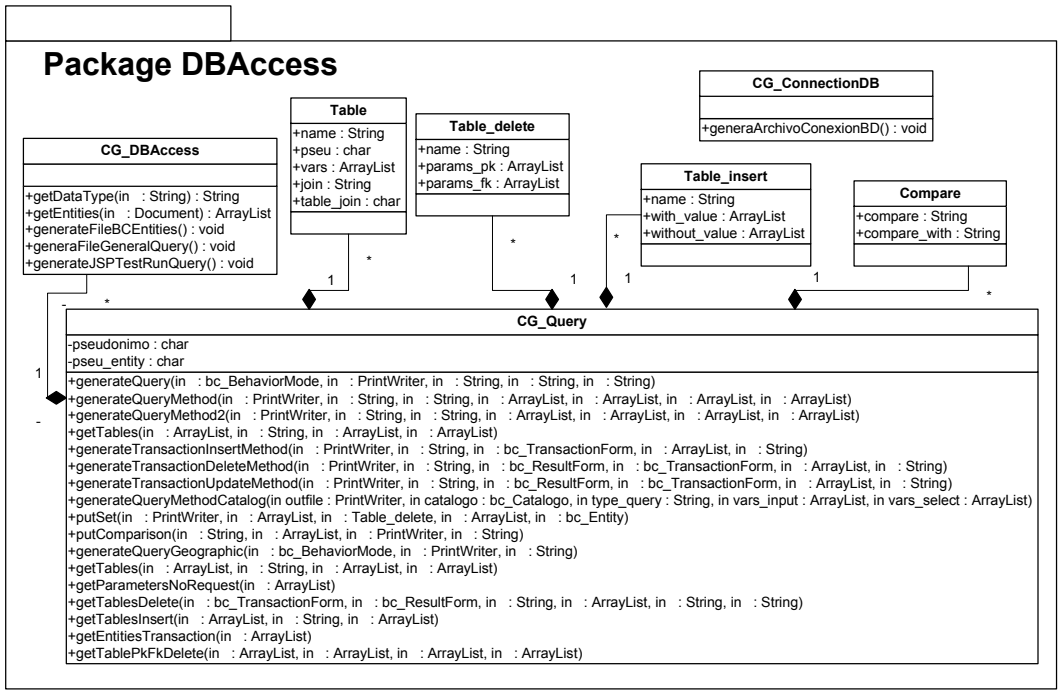

Figura 5.9: Clases para el acceso a Base de datos y generación de consultas

Sin embargo, existen *parámetros* con *campos* que no tienen definida ninguna entidad. Estos casos pertenecen a variables de comparación que auxilian la generación de las consultas. Una variable de comparación debe estar asociada a un *campo* del conjunto de *parámetros* de la *forma básica*. Dependiendo del tipo de dato de cada *campo*, se aplicará de una u otra forma la variable de comparación mediante el método *putComparison* de la clase *CG_Query*. Este método tiene definidas las siguientes variables de comparación:

- Contiene. Para esta variable de comparación se utliza la cláusula *like* con dos comodines de la forma *variable like ‘ % valor % ‘*. Esta forma puede ser aplicada a campos de tipo *char, character, text* e *integer*.

- Empieza con. Para esta variable de comparación se utiliza la cláusula *like* con un comodín al final de la forma *variable like ‘valor % ‘*. Esta forma puede ser aplicada a campos de tipo *char, character, text* e *integer*.

- Termina con. En este caso la cláusula *like* se utiliza con un comodín al inicio de la forma *variable like ‘ %valor ‘*. Esta forma puede ser aplicada a campos de tipo *char, character, text* e *integer*.

- Igual a. Para esta variable de comparación utilizamos el operador =. Para campos de tipo *char, character* y *text* es necesario utilizar las comillas simples de la forma *variable = ‘valor ‘*. Para valores numéricos se utiliza la forma *variable = valor*. De aquí la importancia del tipo de dato que se maneje.

- Diferente de. Para esta variable de comparación utilizamos el operador $<>$. Para campos de tipo *char, character* y *text* es necesario utilizar las comillas simples de la forma *variable* $<>$ *'valor'*. Para valores numéricos se utiliza de la forma *variable* $<>$ *valor*.

- Menor a. Para esta variable de comparación utilizamos el operador $<$. Para campos de tipo *char, character* y *text* es necesario utilizar las comillas simples de la forma *variable* $<$ *'valor'*. Para valores numéricos se utiliza la forma *variable* $<$ *valor*.

- Mayor a. Para esta variable de comparación utilizamos el operador $>$. Para campos de tipo *char, character* y *text* es necesario utilizar las comillas simples de la forma *variable* $>$ *'valor'*. Para valores numéricos se utiliza la forma *variable* $>$ *valor*.

Finalmente para formar la instrucción de código SQL se procede como sigue:

1. *Select*. Para la parte de *select* se toman los *elementos encabezado* (*bc_ItemHeaded*) de la *forma de resultados* que sigue a la *forma básica* de la cual se parte para formar la consulta. Como se explicó en la sección 4.3.1.11 del capítulo 4, la *forma de resultados* representa la información que deseamos consultar y sus *elementos encabezado* contienen campos que son precisamente los que se colocaran en la parte del *select*.
2. *From*. Para la parte del *from* se toman las tablas y sus seudónimos a partir de los objetos tipo *Table* que el generador ha instanciado.
3. *Where*. Para la parte del *where* se toman en cuenta todos los *campos* definidos en la sección de *parámetros* de la *forma básica* incluyendo por supuesto las variables de comparación.

El método que crea la instrucción SQL es *generateQuery* de la clase *CG_Query* mientras que el método *generateQueryMethod* crea a su vez un método en el *bean* de consultas que devuelve una cadena con la consulta formada. De esta manera, queda completamente definida una consulta que será utilizada para buscar entidades sobre las cuales el usuario desea realizar una transacción.

2. Inserción. Para generar el código SQL de esta transacción es necesaria la información de un modo de comportamiento de tipo *Inserción*. La clase *bc_BehaviorMode* cuenta con una *forma básica* y una *forma de transacción*. En los parámetros que se definen en la *forma básica* está la definición completa de los campos que desean insertarse, entonces se procede de la siguiente forma:

1. Se localizan las tablas involucradas generando un arreglo de objetos tipo *Table_insert*.
2. En cada objeto se tiene el nombre de la tabla, los campos de los cuales existen datos y de los cuales no.

3. De esta manera, se puede formar el código SQL de cada tabla del arreglo, auxiliándose de los objetos de tipo *bc_Entity* para verificar el tipo de dato de cada campo.

En la figura 5.10 se muestra gráficamente este procedimiento.

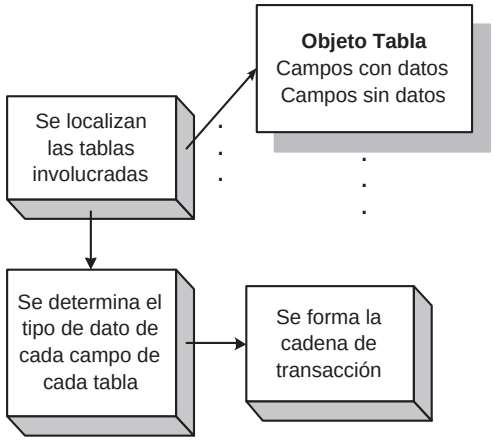

Figura 5.10: Proceso de inserción

3. Eliminación. En el caso de una eliminación, se necesita la información de un modo de comportamiento de tipo *eliminación*. La clase *bc_BehaviorMode* cuenta con una *forma básica* para realizar una búsqueda de la entidad que se desea eliminar, una *forma de resultados* para desplegar el resultado de la consulta y una *forma de detalle* para visualizar el detalle de la entidad seleccionada en la *forma* anterior y finalmente una *forma de transacción*.

Para realizar una elminación partimos de que en la *forma de detalle* ya se tiene selecionada una entidad la cual será eliminada. Esta entidad tiene un campo que lo identifica de manera única (su llave primaria), por lo que para el proceso de eliminación de la figura 5.11 se procede de la siguiente manera:

1. Se recibe el parámetro o campo que identifica a la entidad que se va a eliminar.
2. Dado que el campo que identifica a la entidad es una llave primaria, se busca dicho campo en un arreglo de objetos de tipo *bc_Entity* para conocer la tabla (entidad) de la cual se ha de eliminar un registro.
3. Ya que se tiene la entidad identificada se comienza a buscar las relaciones que tenga con otras entidades. Esto es necesario porque no sólo se elimina un registro de la tabla localizada sino tambien un registro por cada tabla con la cual tenga relación, de lo contrario se pueden provocar inconsistencias en la Base de Datos. En este paso se obtiene un arreglo

de objetos tipo *Table_delete*, cada objeto contendrá información acerca de llaves primarias y llaves foráneas correspondientes a cada entidad.

4. Se recorre el arreglo de objetos formando en cada paso una cadena de código SQL que forma una transacción. Cada objeto contiene sus llaves primarias y foráneas, por lo que se puede ir formando los *joins* de cada transacción. La forma de lograrlo es verificar si la llave primaria de un objeto es la llave foránea de otro y así sucesivamente. Se formarán tantas transacciones SQL como objetos tipo *Tabla_delete* existan en el arreglo.

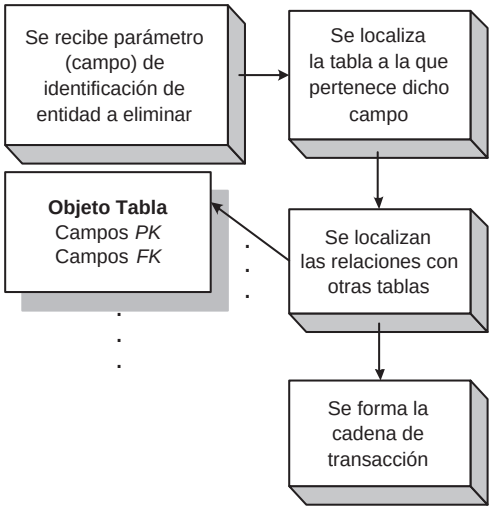

Figura 5.11: Proceso de eliminación

4. Actualización. Para una actualización, se necesita la descripción de un modo de comportamiento de tipo *actualización*. La clase *bc_BehaviorMode* cuenta con una *forma básica* para realizar una búsqueda de la entidad que se desea actualizar, una *forma de resultados* para desplegar el resultado de la consulta y una *forma de detalle* para visualizar el detalle de la entidad seleccionada en la *forma* anterior y finalmente una *forma de transacción*. Dentro de la *forma de transacción* hay una *forma básica* que se utiliza para desplegar los datos existentes y que se desean modificar.

Para realizar una actualización partimos de que en la *forma de detalle* ya se tiene seleccionada una entidad la cual será actualizada. Esta entidad tiene un campo que lo identifica de manera única (su llave primaria), por lo tanto el proceso de actualización de la figura 5.12 procede de la siguiente manera:

1. Se localizan las tablas involucradas para generar una consulta que devuelva los datos existentes que van a ser modificados. Para la localización de tablas, la *forma básica* contenida en la *forma de transacción* contiene *parámetros* que indican los detalles

de cada campo y las entidades (tablas) de las cuales provienen. Se obtiene un arreglo de objetos tipo *Table_delete*, cada objeto con la información necesaria para conocer las relaciones con los otros objetos del arreglo (*joins*).

2. Localizadas las tablas, se genera la consulta para desplegar los datos existentes que van a ser modificados por el usuario.

3. A partir del *bean* de generación de documentos XML, se forma un documento que tenga la estructura de la *forma básica* pero que además contenga los resultados de la consulta.

4. Los pasos 1, 2 y 3 del proceso de eliminación se aplican para localizar las tablas involucradas en el proceso de actualización.

5. Se recorre el arreglo de objetos generado para formar, de la misma manera que en el proceso de eliminación, las transacciones correspondientes a la actualización (cambiando las instrucciones SQL *delete* por *update* y añadiendo la claúsula *set*).

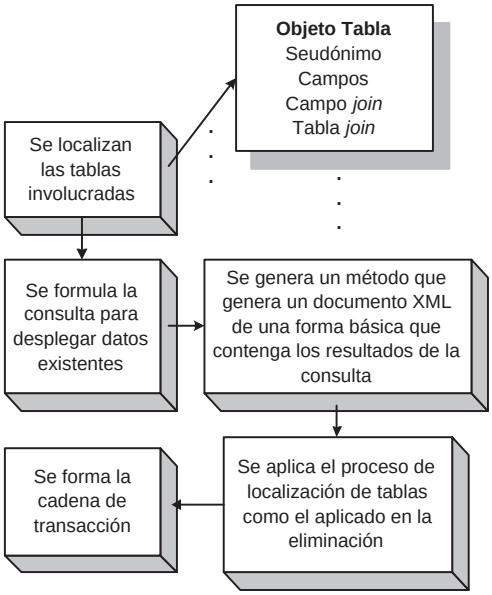

Figura 5.12: Proceso de actualización

Los métodos de la clase *CG_Query* generan los arreglos de objetos tipo *Table*, *Table_insert* y *Table_delete* así como la obtención de llaves primarias y foráneas. También crean los métodos de generación de consultas y su procedimiento almacenado correspondiente. Las clases de este paquete tienen una relación de composición con clases del paquete *bc*.

5.4.3. Entrada/Salida de documentos XML

La clase contenida en el paquete de la figura 5.13 se creo con la finalidad de leer un documento de descripción XML desde un archivo. Como resultado, se puede obtener la descripción como una clase *Document* o bien como un tipo *String*. La aplicación generalmente hace uso de esta clase para obtener un objeto de tipo *Document*. Posteriormente lo recorre para acceder a la información de la descripción. Esta clase es utilizada por las clases del paquete *bc* para el vaciado de información de descripciones a objetos de las entidades de la aplicación.

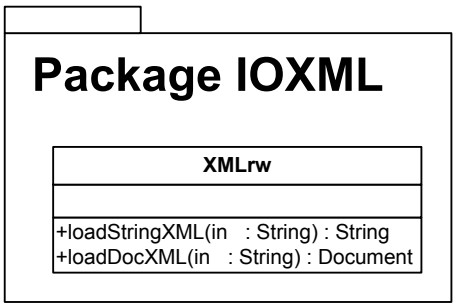

Figura 5.13: Clase para la lectura de un documento XML

5.4.4. Generación de la interfaz

El paquete mostrado en la figura 5.14 contiene las clases para la generación de las interfaces principales existentes en el sistema. La clase *CG_Interface* tiene métodos que crean archivos JSP con código HTML. Este código es el resultado de la transformación de un documento de descripción mediante un estilo. La clase lee el documento de descripción de interfaces y vacía la información a objetos de tipo *bc_FramePrincipal* que a su vez contiene objetos *bc_Border* y *bc_Menu* que conforman una interfaz del sistema.

Otra clase contenida en este paquete es *CG_XML* que tiene la tarea de generar métodos en los *beans* de documentos XML. Generalmente, el contenido de una *forma de resultados* y una *forma de detalle* depende del resultado de una consulta realizada por lo que es necesario contar con un *bean* que cree un documento XML con la información correspondiente y éste pueda ser transformado dentro del JSP del que se invoca el *bean*.

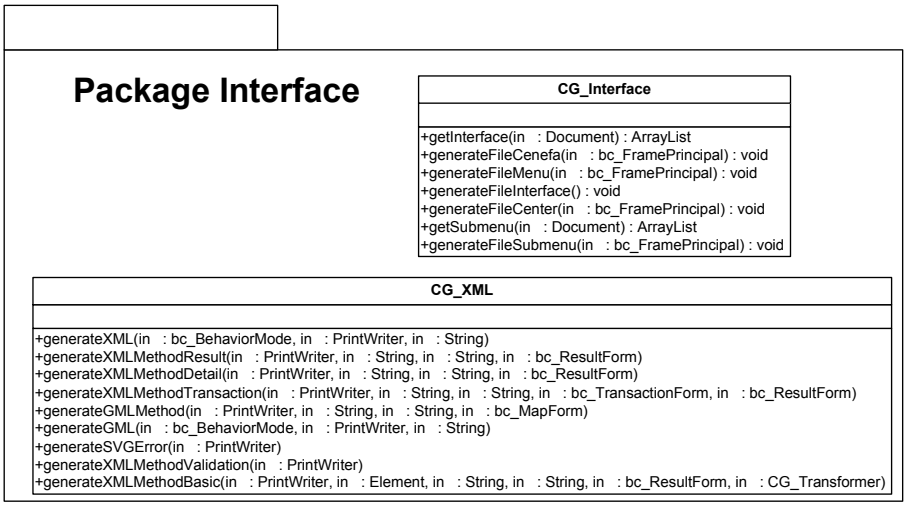

Figura 5.14: Clases para la creación de archivos de interfaz y documentos XML

5.4.5. Mapeo XML a código (Transformaciones)

En la sección 5.4.4 se ha mencionado el uso de estilos para realizar transformaciones. Una de las funcionalidades que provee el generador de código, es la creación de páginas capaces de visualizar mapas, así que vale la pena explicar con más detalle las transformaciones mediante estilos.

Una hoja de estilo es un documento XML cuyo contenido es un lenguaje de programación mediante etiquetas que transforma un documento XML en otro documento XML o incluso en un documento que no tenga formato XML. De esta manera podemos separar la presentación del contenido de una forma efectiva.

Las hojas de estilo que contiene el generador de código están programadas de tal forma, que permiten cambiar muchos parámetros de diseño de cada página. En las descripciones se trata de parametrizar lo mejor posible para que la hoja de estilo haga su trabajo y se pueda obtener el diseño requerido. En el generador de código existen tres tipos de hojas de estilo:

- La que transforma un documento XML en un documento HTML. Éste tipo de hojas de estilo se utilizan principalmente para transformar *formas básicas*, *interfaces principales*, *menús* y *submenús*.

- La que transforma un documento XML en un documento SVG. Éste tipo de hojas de estilo se utilizan principalmente para transformar *formas de mapa*.

- La que transforma un documento XML en un documento XHTML.Éste tipo de hojas de estilo se utilizan principalmente para transformar *formas de mapa*.

De esta manera la hoja de estilo lee el documento XML y comienza a recorrer sus etiquetas. La hoja de estilo está programada para colocar el código correspondiente cada vez que encuentra una etiqueta específica. El funcionamiento de una hoja de estilo es sencillo, así que es tarea del desarrollador hacer una buena programación de la hoja para obtener buenos resultados.

Por ejemplo, para la transformación de una *forma básica*, en la descripción se define elemento por elemento, es decir, un elemento con etiquetas como *tipo, nombre, etiqueta*, etc., en donde *tipo* contiene información acerca de que tipo de elemento HTML se requiere (una etiqueta, una caja de texto, un radio botón, un botón de envío, una lista desplegable, un botón de carga de archivo). Entonces la hoja de estilo se programa para que cada vez que encuentre una etiqueta *tipo* se coloque código HTML del tipo de elemento que se especifico en dicha etiqueta. Así puede definirse completamente el elemento HTML pues a parte de la etiqueta *tipo*, están las etiquetas *nombre y etiqueta* que permiten darle un identificador y un nombre al elemento. De aquí la importancia de hacer un buen diseño tanto de la descripción como una buena programación de la hoja de estilo. En la figura 5.15 se muestra un fragmento de descripción de una *forma básica* y en la figura 5.16 se muestra la correspondiente tranformación en código HTML.

```
- <br>
  - <item>
      <item_type>label</item_type>
      <item_label>Nombre oficial:</item_label>
      <item_name>label:</item_name>
    </item>
  - <item>
      <item_type>text</item_type>
      <item_name>rtn</item_name>
    </item>
- </br>
```

Figura 5.15: Fragmento de descripción XML

```
<tr height="33">
  <td height="20" align="right">
    Nombre oficial:
  </td>
  <td>
    <input type="text" name="rtn" id="rtn" value=""/>
  </td>
</tr>
```

Figura 5.16: Código HTML transformado mediante una hoja de estilo

En la figura 5.17 se muestra la hoja de estilo que transforma una *forma básica*, es decir, el fragmento de código de la figura 5.15. Se definen las etiquetas correspondientes al código HTML y se coloca un título extraído de la descripción si es que esta etiqueta existe con el nombre de *título*. Posteriormente se define una tabla, el nombre de un formulario y a partir

de ahí, los elementos que componen el formulario. Por cada etiqueta con el nombre *br* que se encuentre en la descripción se define una columna. A su vez, por cada etiqueta con el nombre *item* que se encuentre en la descripción que tenga como padre la etiqueta *br* se define un *case* que determina qué tipo de elemento se desea codificar (texto, etiqueta, radio botón, lista desplegable, etc.). Dependiendo del tipo de elemento a codificar, la hoja de estilo selecciona las demás etiquetas de la descripción, por ejemplo para un elemento *input* es necesario la selección de las etiquetas *item_name* e *item_value* para dejarlo bien definido.

```
<xsl:stylesheet version="1.0">
  <xsl:template match="basic_form">
    <style type="text/CSS">
      body,td,th{ font-family:verdana,arial,sans-serif; font-size:11px; font-color:black}
    </style>
    <!-- ******************************** -->
    <HTML>
      <body bgcolor="#FFFFFF">
        <H3>
          <font color="black" face="Arial,Helvetica,Geneva,Swiss,SunSans-Regular">
            <xsl:value-of select="titulo"/>
          </font>
        </H3>
        <table border="0" cellspacing="2" cellpadding="1" align="center">
          <form name="enviar" action="capturas.jsp" method="post" enctype="multipart/form-data">
            <xsl:for-each select="hidden"></xsl:for-each>
            <xsl:for-each select="br">
              <tr height="33">
                <xsl:for-each select="item">
                  <xsl:choose>
                    <xsl:when test="item_type='label'">
                      <td align="right" height="20">
                        <xsl:value-of select="item_label"/>
                      </td>
                    </xsl:when>
                    <xsl:when test="item_type='label_ast'"></xsl:when>
                    <xsl:when test="item_type='espacio'"></xsl:when>
                    <xsl:when test="item_lista='SI'"></xsl:when>
                    <xsl:when test="item_separa='SI'"></xsl:when>
                    <xsl:when test="item_type='radio'"></xsl:when>
                    <xsl:when test="item_type='radio2'"></xsl:when>
                    <xsl:when test="item_type='text'">
                      <td>
                        <xsl:element name="input">
                          <xsl:attribute name="type">
                            <xsl:value-of select="item_type"/>
                          </xsl:attribute>
                          <xsl:attribute name="name">
                            <xsl:value-of select="item_name"/>
                          </xsl:attribute>
                          <xsl:attribute name="id">
                            <xsl:value-of select="item_name"/>
                          </xsl:attribute>
                          <xsl:attribute name="value">
                            <xsl:value-of select="item_value"/>
                          </xsl:attribute>
                          <xsl:if test="item_active">
                            <xsl:attribute name="disabled"/>
                          </xsl:if>
                        </xsl:element>
                        <!--<xsl:value-of select="item_value"/>-->
                      </td>
                    </xsl:when>
                    <xsl:when test="item_type='file'"></xsl:when>
                    <xsl:otherwise></xsl:otherwise>
                  </xsl:choose>
                </xsl:for-each>
              </tr>
            </xsl:for-each>
          </form>
        </table>
      </body>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Figura 5.17: Hoja de estilo

Existen diferentes modos de hacer transformaciones para lograr obtener el código deseado. Para cualquier modo, es necesario como primer paso la lectura de un documento XML y posteriormente cualquiera de las siguientes formas:

1. se realiza directamente la transformación mediante una hoja de estilo, se indica el nombre y ubicación del archivo de descripción (con extensión xml) y el nombre y ubicación de la hoja de estilo (archivo con extensión xsl). Para realizar esto, se invoca a un método de la clase *CG_Transformer* del paquete *Tranformer* (figura 5.19) y como resultado se obtiene código HTML que será embebido dentro de una página JSP,

2. se vacía la información leída en la descripción a objetos instanciados de clases del generador de código. Algunos de éstos objetos necesitan una transformación por lo que a partir de los objetos de tipo *Document* se puede realizar la transformación de las siguientes maneras:

   a) se realiza directamente la transformación mediante una hoja de estilo, a diferencia del punto 1 aquí se invoca a un método de la clase *CG_Transformer* indicando el objeto tipo *Document* y el nombre y la ubicación de la hoja de estilo con la que se realizará la transformación. Como resultado se devuelve el código HTML que también será embebido en una página JSP,

   b) para los objetos que no necesitan una transformación, la aplicación genera código a partir de la información que contengan los objetos. Este código también será embebido dentro de una página JSP, pero para su generación no ha necesitado de ninguna hoja de estilo,

   c) A partir de objetos tipo *Document* la aplicación genera *beans* que crean documentos XML (generalmente con resultados de consultas) y también genera páginas JSP que invocan métodos de los *beans* generados indicando el estilo que se usará para hacer la transformación. En este caso en la página JSP sólo queda indicado el método que se invoca y el estilo que transforma el documento que devolverá el método, sin embargo, la transformación se realiza en tiempo de ejecución, es decir, cuando la aplicación Web este en funcionamiento. Esto se hace así debido a que los documentos que el método del *bean* devuelve no son estáticos, dependen de una consulta que el usuario realice mientras navega en la aplicación Web.

En cada transformación se verifica la existencia de caracteres con acentos para indicar su correspondiente código HTML. En la figura 5.18 se muestra un diagrama con las diferentes formas de realizar una transformación siguiendo el procedimiento explicado anteriormente.

En el caso de la visualización de mapas desde la aplicación Web es necesario seguir un procedimiento para transformar un documento XML en código SVG. En la figura 5.20 se muestra dicho procedimiento y a continuación la explicación del mismo.

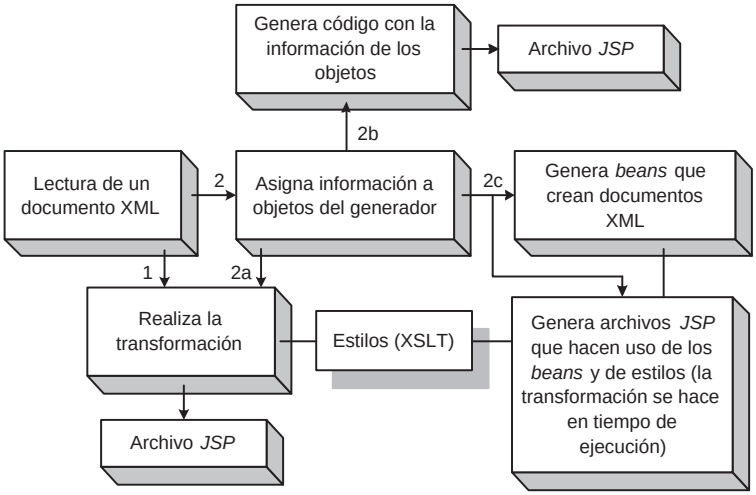

Figura 5.18: Modos de realizar una transformación

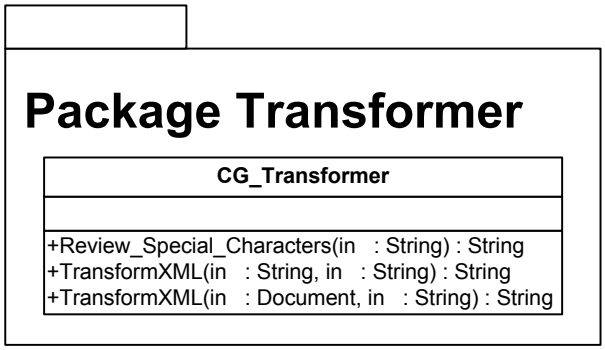

Figura 5.19: Clase para transformaciones de documentos XML

Como se explicó en la sección de descripciones, existe una *forma de mapa* que a su vez contiene una sección de forma donde se podrá realizar una consulta geográfica, una sección de mapa donde se visualizará el mismo, y una sección de datos donde se desplegarán datos nominales de las entidades graficadas en el mapa. Partiendo de la consulta geográfica que el usuario puede realizar, el procedimiento mostrado en la figura 5.20 para visualizar el mapa es el siguiente:

1. se obtiene un objeto geométrico (*PG_Geometry* del conjunto de clases de Postgis) resultado de la consulta,
2. se hace una correspondencia (*casting*) con un tipo de objeto geométrico,
3. se transforma el objeto geométrico a un arreglo de objetos tipo *Point*. Por ejemplo, si el objeto geométrico que se obtuvo es un *MultiLineString* entonces se desglosa a un arreglo de objetos de tipo *LineString*, luego cada objeto del arreglo se desglosa a un arreglo de objetos tipo *Point* y

así se procede para cada objeto geométrico. En la tabla 5.1 se muestra las transformaciones de objetos geométricos,

4. dado el arreglo de objetos tipo *Point* se transforma a un conjunto de coordenadas que representan las coordenadas que describen una entidad geométrica,
5. cada objeto resultado de la consulta representa una entidad geométrica, por lo que los pasos del 1 al 4 se realizan por cada objeto geométrico que resulte de la consulta,
6. una vez formadas las coordenadas de todas las entidades, se forma un documento GML, que es un documento XML pero con referencia especial a esquemas geométricos. Este documento GML es producto de la invocación de un método del *bean* de generación de documentos XML. En la figura 5.21 se muestra el esquema geométrico que el generador de código utiliza. Para formar los documentos GML se hace referencia al esquema *features* del estándar GML 3.1.1 que a su vez contiene otros esquemas para la completa definición de propiedades geométricas,
7. después, mediante un estilo, se transforma el documento GML a código SVG de la forma explicada en el punto 2c de transformaciones. Este código se embebe en una página JSP y para que la página pueda funcionar correctamente el tipo de contenido del JSP debe ser de tipo XHTML.

La sección donde se despliegan datos nominales y la sección de consulta se generan a partir de un documento XML y una hoja de estilo dando como resultado código XHTML. Se generan dos páginas JSP, una con el formulario para la consulta geográfica y otra para desplegar los datos nominales de la entidad, ambas se invocan desde la página JSP donde se embebe el código SVG.

Objeto original	**1er paso**	**2do paso**	**3er paso**
MultiLineString	LineString	Point	
MultiPolygon	Polygon	LinearRing	Point
LineString	Point		
Polygon	LineRing	Point	

Tabla 5.1: Proceso de transformación de objetos geométricos

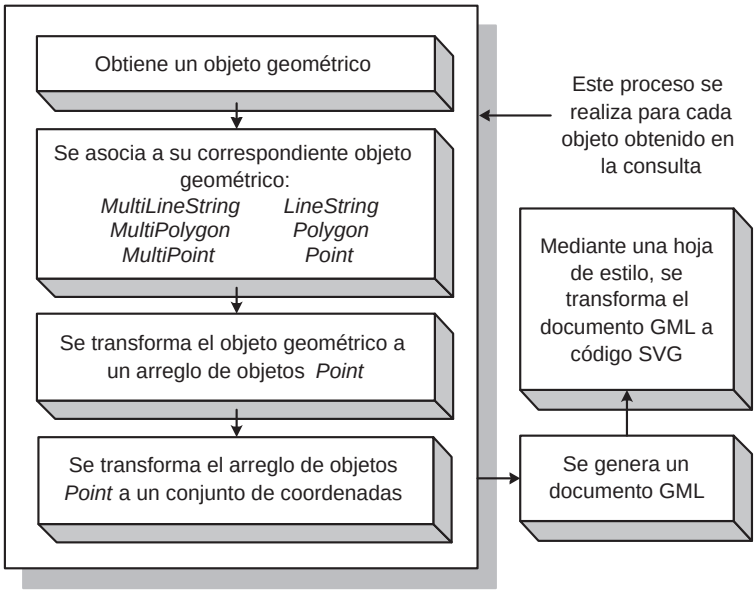

Figura 5.20: Proceso para una transformación geométrica

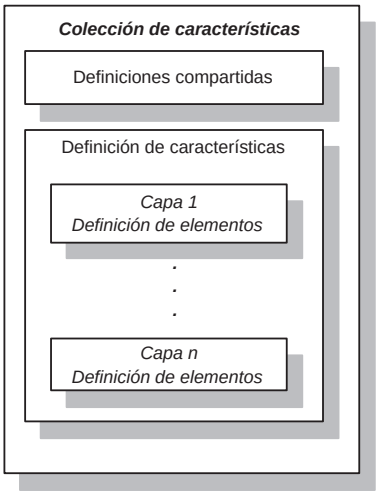

Figura 5.21: Esquema geométrico (Estándar GML 3.1)

5.4.6. Creación de directorios

El paquete *Directories* contiene la clase que implementa los métodos para la generación del árbol de directorios donde se va a almacenar el código creado. La figura 5.22 muestra el diagrama de clase de este paquete.

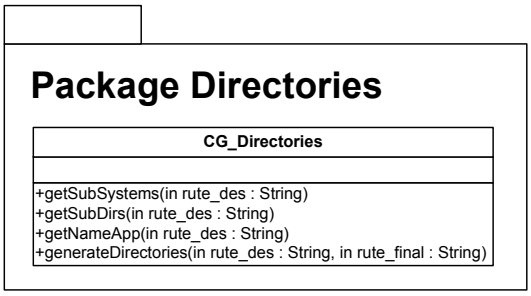

Figura 5.22: Diagrama de clase del paquete *Directories*

5.4.7. Generación de páginas

La invocación a los métodos de la clase *CG_Pages* para la creación de JSPs se hace desde la clase *CG_Behavior*, pues en ésta última se hace el recorrido del arreglo de objetos tipo *bc_BehaviorMode* que contiene uno o más tipos de *formas* y los *catálogos* del sistema. Así también se genera el archivo para autenticación de usuario. La figura 5.23 muestra el paquete *Pages* y las clases para generación de páginas, validación de las mismas y para la creación de una hoja de estilo para la visualización del mapa.

En los métodos de la clase *CG_Pages* (*generateFileTransactionForm*) se hace el llamado a otros métodos de la misma clase (*makeCallInsert, makeCallDelete* y *makeCallUpdate*) para crear el código de la invocación a métodos de *beans* que se han creado a lo largo del proceso de generación de código del comportamiento del sistema.

5.4.7.1. Validación de páginas generadas

Para que el sistema Web generado sea consistente se añadieron validaciones a las páginas que contienen formularios que el usuario deberá llenar con información. Estas páginas son generadas a partir de *formas básicas*, por lo que la definición de campos que contiene esta descripcón permite conocer la entidad (tabla) a la que pertenece cada campo. Así, la descripción de entidades indica las propiedades de cada campo (tipo de dato, longitud y si puede o no ser nulo). Teniendo esta información, el generador puede insertar código de validación *JavaScript* en cada página mediante los métodos de la clase *CG_Validation* de la figura 5.23. Los tipos de validación que se generan son:

- Campos no vacíos. En el caso de formularios para inserción, es necesario que el usuario capture los campos requeridos, así se evita un error en el momento que se este realizando la transacción en la Base de Datos.

- Longitud de la información. Principalmente para campos de texto, es necesario validar que no sobrepase la longitud especificada en cada campo.

- Tipos de datos. Es importante validar los tipos de datos que pueden capturarse en los formularios. Estas validaciones corresponden a permitir sólo numeros para campos numéricos, sólo letras para campos de texto, y en el caso de fechas validar que tengan un formato adecuado.

- Extensión de archivos. Especialmente para realizar la captura de un objeto geométrico, el generador define que puede hacerse mediante la carga de un archivo con extensión *.txt*. Debido a lo anterior, es necesario hacer la validación de extensión de archivos.

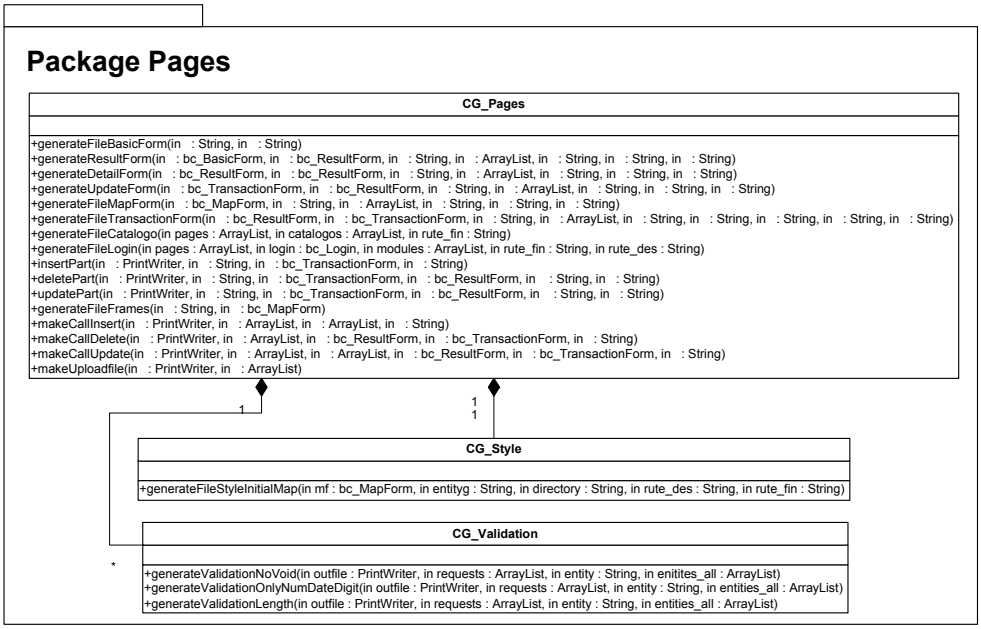

Figura 5.23: Diagrama de clase del paquete *Pages*

5.4.8. Manejo de beans

La figura 5.24 muestra la clase que implementa los métodos para la apertura y cierre de un archivo Java que corresponde a un *bean*. Cuando se obtiene el identificador de dicho archivo, éste se va pasando como parámetro a otros métodos de otras clases para escribir sobre ese archivo. De esta forma, los métodos de las clases *CG_XML* y *CG_Query* reciben el identificador del archivo y escriben código correspondiente a métodos que serán utilizados en los archivos JSP generados por las clases del paquete *Pages*.

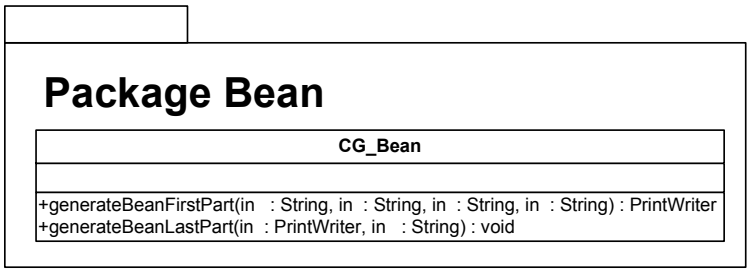

Figura 5.24: Diagrama de clase del paquete *Beans*

5.4.9. Funcionamiento del generador

En la figura 5.25 se muestra la forma en la que el generador de código va realizando cada actividad.

1. El primer paso es generar el *bean* de conexión a Base de Datos. Se generan métodos para apertura de conexión, cierre de conexión y ejecución de consultas.

2. El siguiente paso es generar los *beans* de entidades identificados en el sistema que se está generando. Cada *bean* con sus métodos de asignación y lectura de propiedades.

3. Posteriormente, se generan las páginas JSP de las interfaces principales (una interfaz principal por cada módulo identificado en el sistema que se está generando).

4. Después se generan las páginas JSP para la autenticación de usuario.

5. Como se muestra en la figura 5.25, se entra a un ciclo que está representado por el cuadro más grande, este ciclo se realiza por cada módulo de comportamiento que haya resultado de la descripción de comportamiento. En este proceso se realiza:

   - Generación de páginas JSP de todas las *formas* contenidas en el módulo
   - Generación de *beans* de consultas y transacciones a la Base de Datos.
   - Generación de un archivo de texto de procedimientos almacenados
   - Generación de *beans* para la creación de documentos XML y GML

6. El último paso es la generación del código relacionado con Catálogos (páginas JSP, *bean* de consulta y de generación de documentos XML).

El paso 5 y 6 corresponde al método de la clase *CG_Behavior* de la figura 5.26. La clase *CG_Forms* permite vaciar la información de la descripción de comportamiento a objetos de tipo *bc_BasicForm* para su procesamiento en el método de la clase *CG_Behavior*. Desde esta clase se invocan métodos de generación de páginas, transformaciones, *beans* de generación de XML, *beans* de consulta situados en las clases de los paquetes que se han ido explicando en las secciones anteriores de este capítulo.

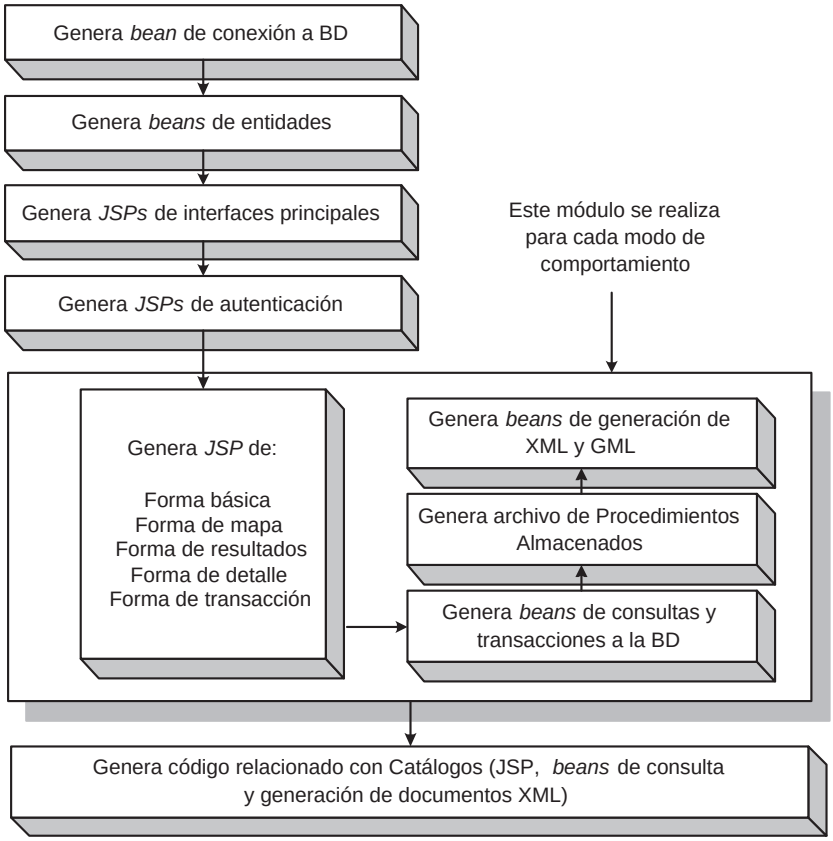

Figura 5.25: Proceso de generación de código

Para finalizar este capítulo de implementación, a continuación se explica la aplicación que puede generarse y un pequeño ambiente gráfico para el generador de código.

5.4.10. Implementación del IDE

A lo largo del presente capítulo se ha explicado cada una de las acciones que realiza el generador de código para crear el código de una aplicación Web. En las secciones anteriores se ha explicado el tipo de código que se genera (figura 4.32) y la distribución del código en carpetas (figura 4.33) Para que ésto sea una tarea sencilla se desarrolló una interfaz que permite generar el sistema Web de una manera amigable.

El pequeño IDE está desarrollado completamente en *Swing* [11] para la programación de las acciones de los componentes gráficos. Como se muestra en la figura 5.27 la interfaz se compone de un menú, botones, una lista desplegable, un área de texto y una barra de progreso. Las acciones de los componentes pueden tambien realizarse desde el menú de la interfaz. Cada componente gráfico tiene asociada una actividad como se muestra en la figura 5.25.

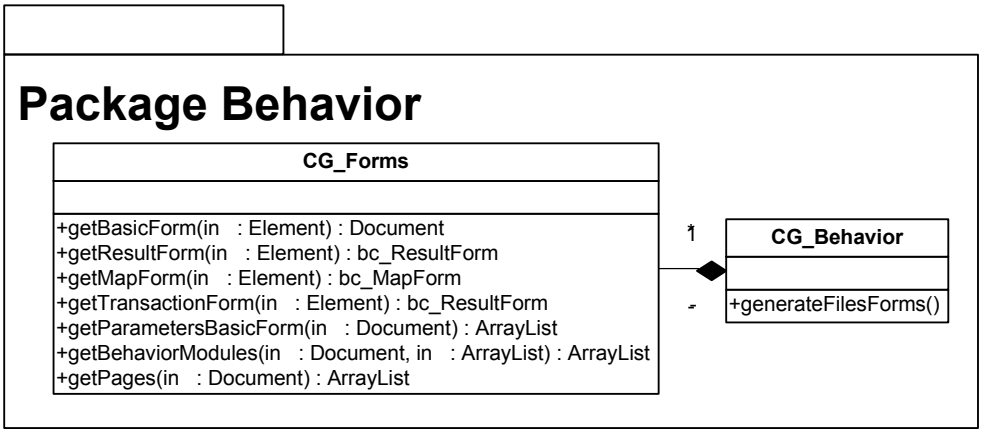

Figura 5.26: Diagrama de clase del paquete *Behavior*

Cada componente realiza una acción determinada:

1. Se selecciona la ubicación donde se almacenará el código generado.
2. Se selecciona la carpeta que contiene todas las descripciones. Una vez seleccionada, se creará el árbol de directorios para alojar el código generado (figura 4.33)
3. Se selecciona la carpeta que contiene las imágenes que el sistema Web generado va a utilizar
4. Se generan los *beans* de conexión a Base de datos
5. Se generan las páginas que conforman las interfaces principales
6. Finalmente en la lista desplegable se podrá elegir entre los módulos del sistema para así generar el comportamiento de un subsistema o bien generar todos los módulos y así generar el sistema completo
7. La caja de texto va describiendo lo que se genera cada que el usuario realiza una acción
8. De igual manera la barra de progreso indica el porcentaje de código generado
9. El usuario puede seleccionar salir de la aplicación

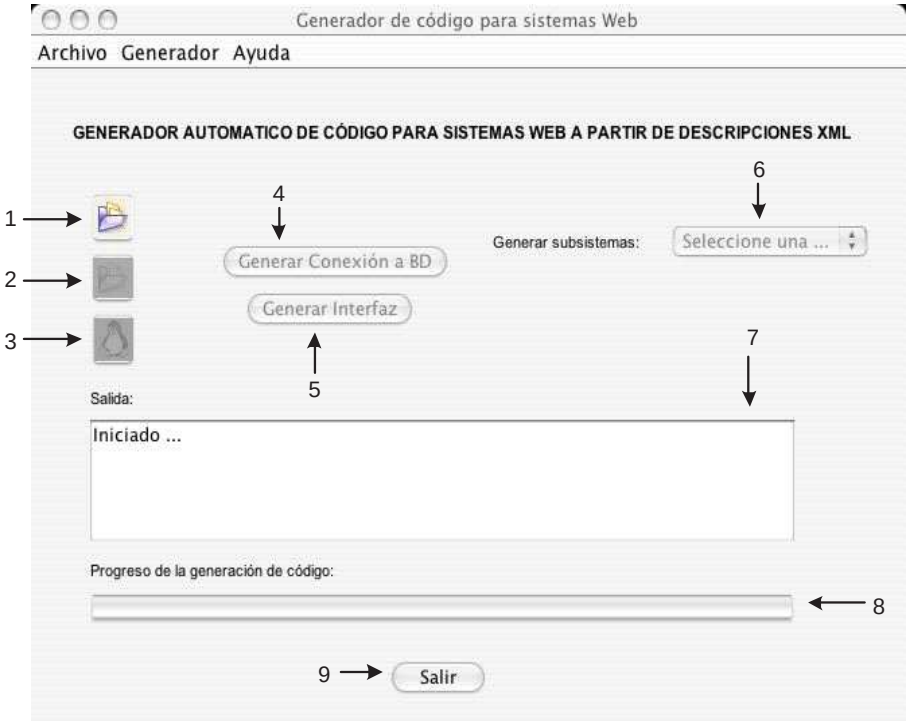

Figura 5.27: Interfaz gráfica del generador de código

5.5. Conclusiones

La explicación de cada uno de los módulos que integran aplicación permite que se pueda entender mejor la forma en la que está implementada la misma. Cada paquete definido tiene relación con otros paquetes de la aplicación para realizar una función específica. Se observó la relación que existe entre las descripciones y los módulos de la aplicación para lograr la generación del código. El resultado se puede visualizar eligiendo un caso de estudio específico que permita abarcar todas las características del generador de código para comprobar su potencial.

Capítulo 6

Caso de estudio

Para comprobar la funcionalidad del generador de código se eligió un caso de estudio que cubriera la parte geográfica que se puede manejar en el sitio Web generado. El caso de estudio elegido es la generación del sitio Web de la capa de hidrología de Colima. En las siguientes secciones se explica de manera general como está organizada la información de Colima y cómo podemos trasladar esa información a descripciones para utilizar el generador de código.

6.1. Requerimientos

El requerimiento global es la realización de un sitio Web para el Sistema de Información Geográfica de la capa de hidrología del estado de Colima. A continuación se lista de manera muy general los requerimientos del sistema.

1. Autenticación de usuario
2. El sistema debe estar dividido en subsistemas
3. Cada subsistema debe contar con una interfaz principal
4. Cada interfaz principal debe contar con un menú principal y un submenú por cada menú definido
5. Debe realizar transacciones básicas (inserción, eliminación y actualización) sobre cada entidad identificada en el sistema
6. Cada operación de consulta o transacción puede ser ejecutada desde el cliente o bien mediante procedimientos almacenados
7. Consulta de datos nominales de las entidades involucradas
    a) Búsqueda de información por distintos parámetros
    b) Visualización de resultados de búsquedas

   c) Selección de un resultado para visualizar sus datos nominales

   d) Visualización del detalle de datos nominales

8. Consulta de datos geográficos de las entidades involucradas

   a) Búsqueda de información por distintos parámetros (geográficos)

   b) Visualización de entidades geográficas resultado de la búsqueda

   c) Selección de una entidad para visualizar sus datos nominales

En las secciones posteriores se describen con más detalle los requerimientos y su asociación con las descripciones para generar el sitio Web requerido.

6.2. Traslación de requerimientos a descripciones

El conjunto de requerimientos debe ser trasladado a su correspondiente descripción para que el generador de código tome como entrada estas descripciones y genere el sitio Web requerido. Cada requerimiento listado puede ser ubicado en una de las descripciones que se explicaron en el capítulo anterior. A continuación se asocian las descripciones con el correspondiente requerimiento.

6.2.1. Modelo de datos

Colima se encuentra en la región oeste de la República Mexicana. Limita al norte, al este y al oeste con Jalisco, al sureste con Michoacán y al sur con el Océano Pacífico. Su capital es la ciudad de Colima. El estado tiene un área de 5,433 km. que representa el 0.3 % de la superficie del país y es uno de los estados más pequeños del país.

El modelo de datos está constituido por 9 capas geográficas, cada una de ellas cuenta con distintas entidades que se diferencían por sus atributos, así como por la capa geográfica y la capa de nivel a la que pertenecen:

- Altimetrá y Datos de Elevación
- Hidrografía e Infraestructura Hidráulica
- Localidades y rasgos urbanos
- Límites
- Instalaciones diversas e industriales
- Tanques de almacenamiento, conductos y líneas de transmisión
- Comunicación y transporte

- Elementos de referencia topográfica
- Áreas protegidas y de interés

El caso de estudio se enfoca en la capa de **Hidrografía e Infraestructura Hidráulica** que a su vez se divide en cartas y entidades geográficas:

- Rasgos Hidrográficos Puntuales
  - Manantial
  - Rápido
  - Salto de agua
  - Tanque de agua
- Corrientes y vías de conducción de agua
  - Acueducto
  - Bordo
  - Canal
  - Corriente de agua
  - Muro de contención
  - Presa
  - Rápido
  - Salto de agua
  - Separador
- Cuerpos de agua
  - Canal
  - Cuerpo de agua
  - Estanque
  - Salina
  - Tanque de agua

Por lo tanto, cada carta representa un módulo en el sistema, así que existirán tres módulos. La Base de Datos de la capa de hidrología ya está creada en PostgreSQLy reside en un servidor con una IP fija, por lo que de aquí se desprende la descripción de *Conexión al servidor de Base de Datos*. Con esta información se pueden realizar algunas de las descripciones. La primera descripción que puede obtenerse es la de *entidades*. Se traslada cada propiedad de las entidades a un archivo XML de la forma explicada en la sección 4.3.1 del capítulo 4. Además se define un nombre de aplicación y un nombre para cada carpeta que contendrá el código de cada módulo y así puede crearse la descripción de *directorios*.

6.3. Funcionamiento del generador de código

Una vez que se han asociado los requerimientos a cada una de las descripciones, se procede a la generación del sistema Web como se indicó en la sección 5.4.10 del capítulo 5. Como resultado se puede visualizar el sitio Web en un navegador con soporte para SVG.

6.3.1. Autenticación de usuario

El sistema Web que se requiere necesita la autenticación del usuario. Existe un usuario y una contraseña diferente para acceder a cada módulo del sistema. Por lo tanto este requerimiento se asocia a la descripción de *autenticación*. Además se elige un logotipo y el nombre para que identifique la aplicación Web. ATIC es el nombre de la aplicación Web que es una palabra nahuatl que significa *agua limpia o agua clara*. Se eligió este nombre debido a la relación con la capa de hidrología para la cual esta diseñada la aplicación Web.

En la figura 6.1 se muestra la página de autenticación, donde dependiendo del usuario que se introduzca, se ingresa a un módulo diferente. En este caso se ingresa al módulo de *Corrientes y Vías de Conducción de Agua*.

Figura 6.1: Página de autenticación de usuario

6.3.2. Interfaz

Debido a que se han identificado los tres módulos se puede elaborar la descripción de *directorios* y a partir de aquí las descripciones de *interfaz*.

- En cada interfaz se define un color de página diferente para identificar a cada módulo.

- En la descripción de *encabezado de página* se elige una imagen, un logotipo y una leyenda que describa al módulo.
- La descripción de *menú* en cada descripción de *interfaz* se va formando de modo que cada entidad sea una opción del menú.
- Al final de cada menú se agrega una opción más que no es precisamente una entidad, esta opción es la de mapa que en la sección de 5.4.5 del capítulo 5 se explica.
- Cada una de estas opciones de menú despliegan un submenú (excluyendo la opción de mapa).
- Las opciones de cada submenú son precisamente los *modos de comportamiento* descritos en la sección 4.3.1.11 del capítulo 4 que serán las operaciones que pueden realizarse sobre cada entidad del sistema.
- Los parámetros de diseño (color, tipo de letra, color de fondo) de cada opción de menú y submenú serán parecidos a los de la interfaz que los contenga para que la página se vea homogénea. En este punto, se asocian las descripciones de *menú* y *submenú* a los requerimientos especificados.

En la figura 6.2 se muestra la interfaz principal de el módulo *Corrientes y Vías de Conducción de Agua*, cada opción del menú corresponde a cada entidad contenida en el módulo.

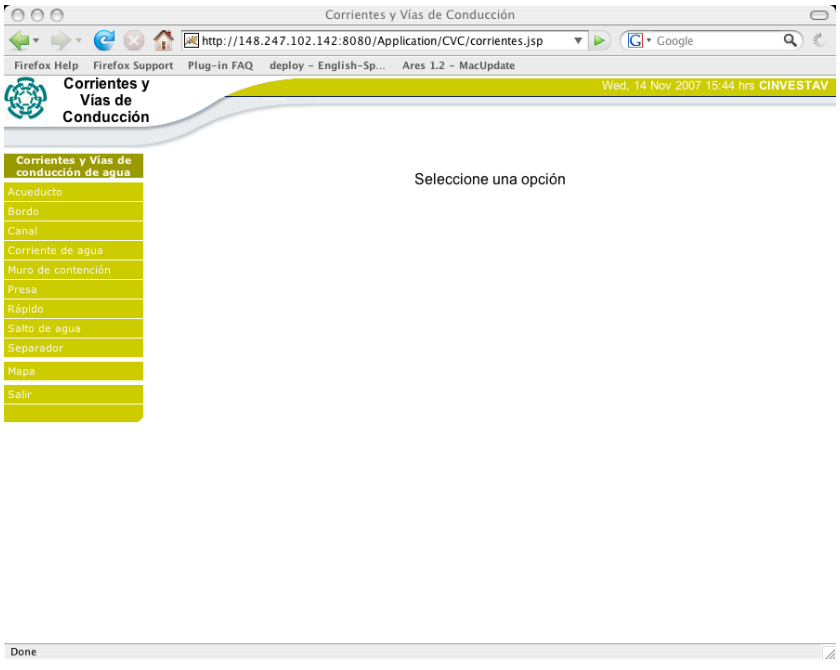

Figura 6.2: Interfaz principal del módulo *Corrientes y Vías de Conducción de Agua*

Si el usuario elige el menú *Acueducto* se despliega el submenú con las operaciones que pueden realizarse sobre esa entidad. En la figura 6.3 se muestra el submenú.

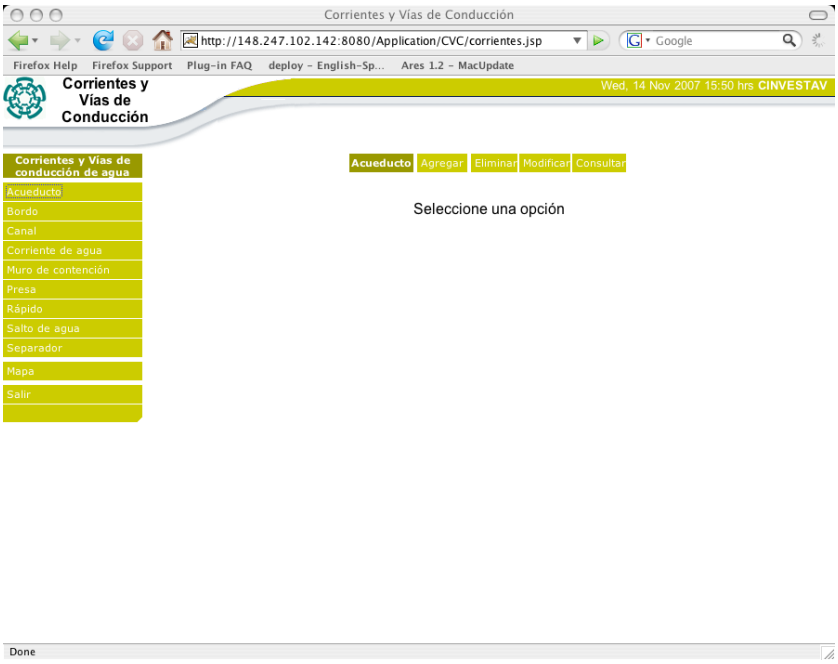

Figura 6.3: Submenú de la opción *Acueducto*

6.3.3. Catálogos

El sistema sólo contiene un catálogo de capas. La información que se muestra en el catálogo debe ser filtrada de acuerdo a la entidad de la que se desee información de capas y de acuerdo a la entidad geográfica relacionada con la entidad manejada. Para mostrar la información de un catálogo, desde un formulario se abre una nueva página en el navegador y cuando el usuario elige un dato, la ventana se cierra y el dato queda registrado en un cuadro de texto del formulario de donde se invocó el catálogo.

Si el número de registros del catálogo es muy extenso, entonces la información se muestra por paginación, es decir, se muestra un número determinado de registros y se puede avanzar o retroceder para visualizar los demás registros. Todos los detalles para cubrir este requerimiento están registrados en la descripción de *Catálogos*.

Dentro de un formulario se puede activar el catálogo para elegir una capa. La figura 6.4 muestra el despliegue del catálogo.

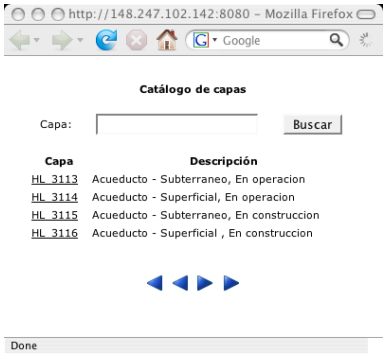

Figura 6.4: Catálogo de capas

6.3.4. Comportamiento

Para cubrir este punto, la descripción de *Comportamiento* contiene la información necesaria para generar el código correspondiente. Como se mencionó anteriormente, cada submenú corresponde a un *modo de comportamiento* o una operación que podrá realizarse sobre la entidad a la que está asociada el submenú. Entonces, dependiendo del *modo de comportamiento* se definirán las *formas* para determinar la navegación, formularios y mapas.

Para el sitio Web requerido, cada opción de menú (entidad) tiene un submenú con los cuatro *modos de comportamiento* (excepto la opción de mapa). En cada *modo de comportamiento* se definen *formas* que establecen la funcionalidad del sitio. Todas las páginas que se despliegan se visualizan en la sección de contenido de cada interfaz. Para elegir cada *modo de comportamiento* se selecciona el submenú correspondiente. La descripción de *comportamiento* está clasificada para cada módulo identificado, en este caso, el módulo *Corrientes y Vías de conducción de agua* contiene nueve entidades, por lo que en la descripción se definen los siguientes modos por cada entidad:

6.3.4.1. Modo de comportamiento: consulta

Dentro de este modo se especifican las consultas nominales y las geográficas. Se inicia con una página que contenga un formulario para la búsqueda de datos nominales para la entidad en cuestión. Esto se logra mediante la descripción de una *forma básica*. Los parámetros de búsqueda de información son básicamente el nombre de la entidad, el identificador geográfico y la capa a la que pertenece.

Por cada parámetro de búsqueda se define una etiqueta, una variable de comparación (lista desplegable) y una caja de texto para que el usuario capture la información. Al final se incluye un botón que activa la búsqueda de la información y la invocación a una nueva página con los resultados.

En la figura 6.5 se muestra el formulario para la búsqueda de entidades.

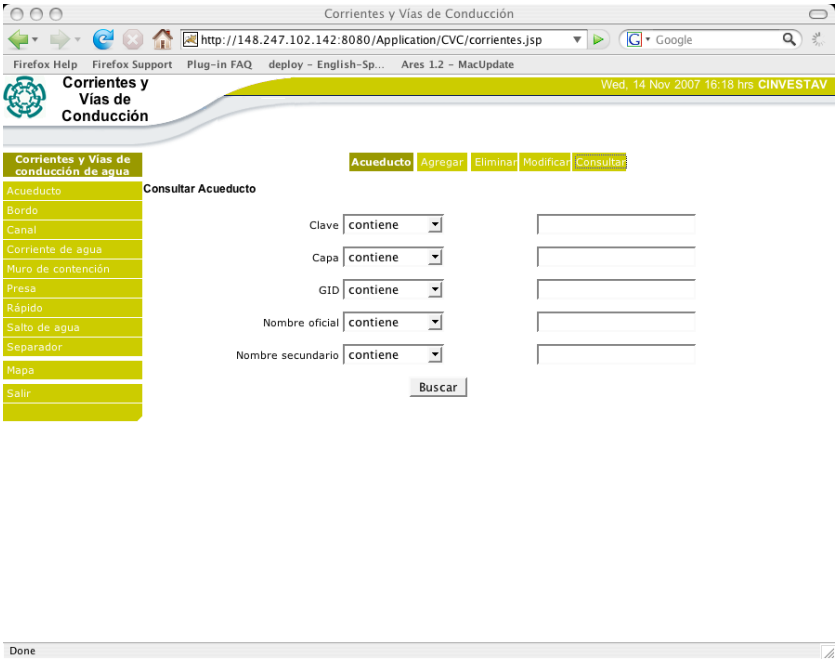

Figura 6.5: Consulta sobre la entidad *Acueducto*

Para la página de resultados la descripción correspondiente es la de *forma de resultados*. En una tabla se despliega la información de algunos campos de las entidades que resultaron de la búsqueda. Cada dato de una de las columnas de la tabla es una liga (post accion) que invoca a una página con el detalle de información de una entidad específica.

Para la página de detalle de información la descripción que le corresponde es la de *forma de detalle*. De manera similar que en la *forma de resultados*, dentro de una tabla se despliega la información de todos los campos (o los que se hayan especificado en la descripción según el requerimiento) para conocer toda la información nominal de esa entidad. Esta es la última página de este *modo de comportamiento*.

La figura 6.6 muestra el resultado de la búsqueda. Se puede visualizar una liga en la primera columna de información para elegir algún resultado y así visualizar el detalle de la información como se muestra en la figura 6.7.

Dentro de este *modo de comportamiento* se incluye también la consulta geográfica y su descripción corresponde a una *forma de mapa*. En la sección de datos se define un formulario para la búsqueda de datos geográficos y visualizarlos en la sección de mapa. El primer parámetro de búsqueda es la o las entidades que se desean visualizar en el mapa, ésto se hace mediante una lista desplegable de selección múltiple.

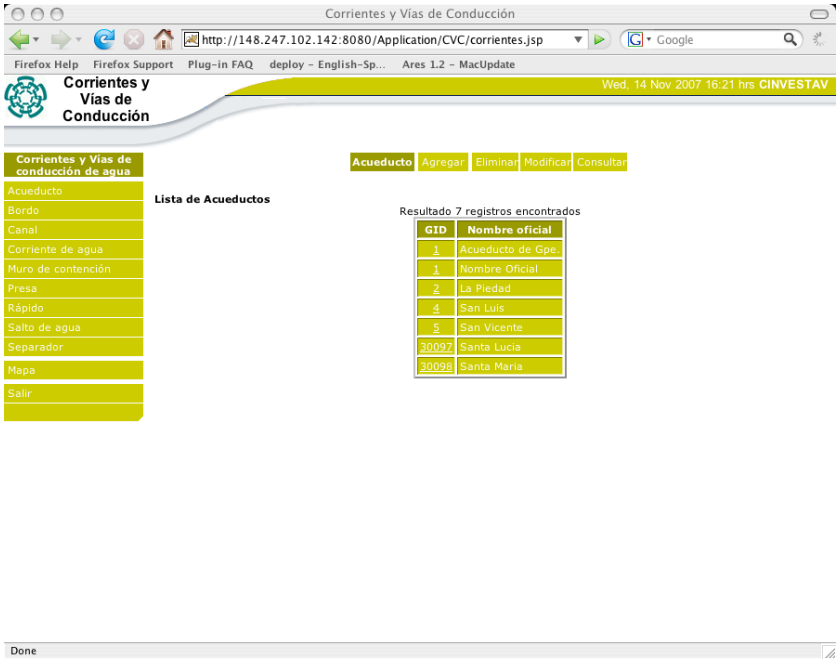

Figura 6.6: Lista de acueductos resultado de la búsqueda

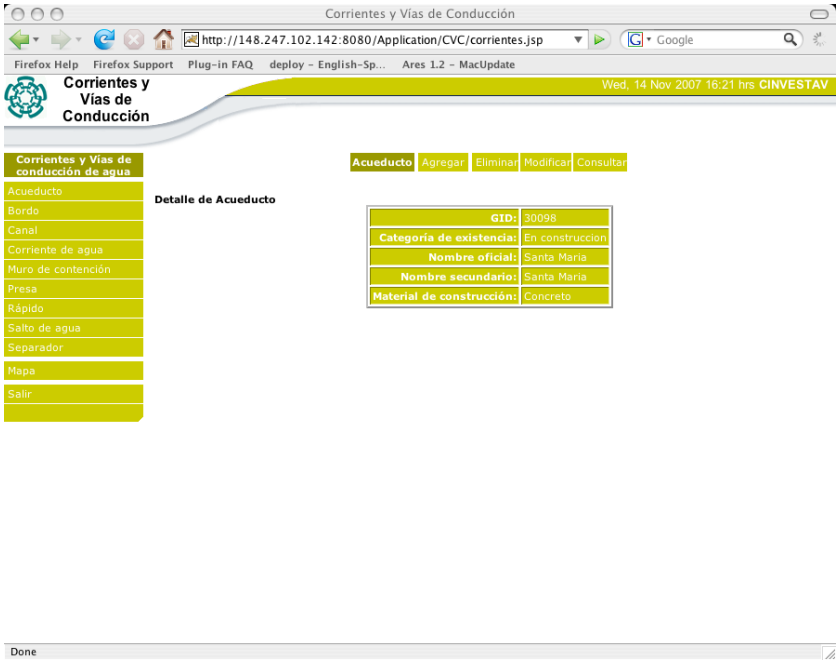

Figura 6.7: Detalle de información de la entidad *Acueducto*

En el caso del módulo de *Corrientes y vías de conducción de agua* las entidades geográficas son *MultiLineString* por lo que los demás parámetros de búsqueda son capa, región, longitud y distancia desde un punto. Al final del formulario se coloca un botón que invoca la visualización de los resultados en la sección de mapa.

En el caso del módulo *Cuerpos de agua* las entidades geográficas son *MultiPolygon* por lo que en vez de longitud y distancia, los parámetros de búsqueda pueden ser área y perímetro. Para el caso del módulo *Rasgos Hidrográficos Puntuales*, las entidades geográficas son *Point* así que los parámetros de búsqueda podrán ser longitud y distancia.

En la sección de mapa se visualizan los resultados de la consulta geográfica formada con los parámetros de búsqueda definidos en la sección de datos. En el mapa se puede hacer la selección con el *mouse* de una entidad y visualizar su identificador, además visualizar los datos nominales de la entidad seleccionada en la sección de datos resultado. Los datos nominales se despliegan en una tabla cuyos parámetros de diseño se especificaron en la descripción de *forma de mapa*. Estos parámetros deben ser similares a los especificados en la interfaz del módulo.

En la figura 6.8 se muestra el mapa de los acueductos y bordos existentes en este módulo. La figura 6.9 muestra la información nominal de una entidad seleccionada por el usuario (la entidad con identificador mostrado en el mapa).

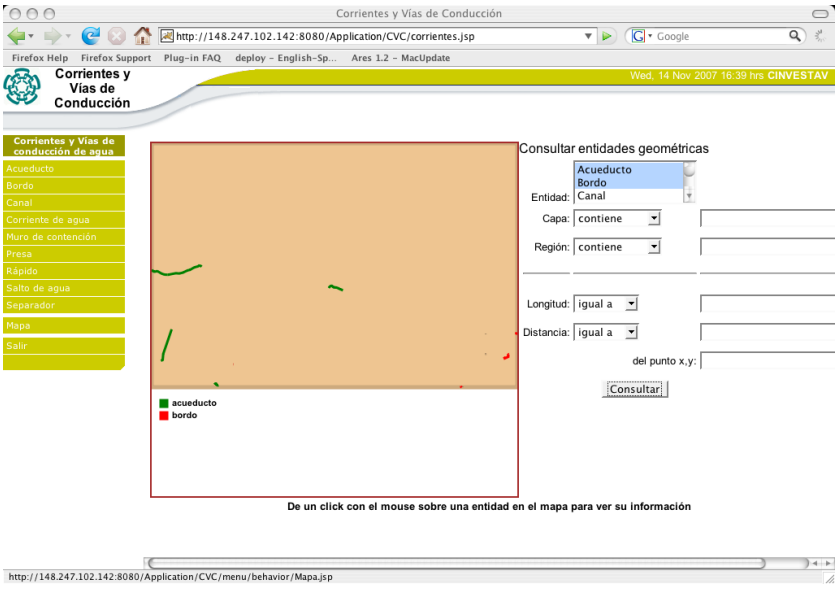

Figura 6.8: *Acueductos* y *Bordos* existentes

La figura 6.10 muestra el mapa y la información nominal de un canal para el módulo *Cuerpos de Agua*.

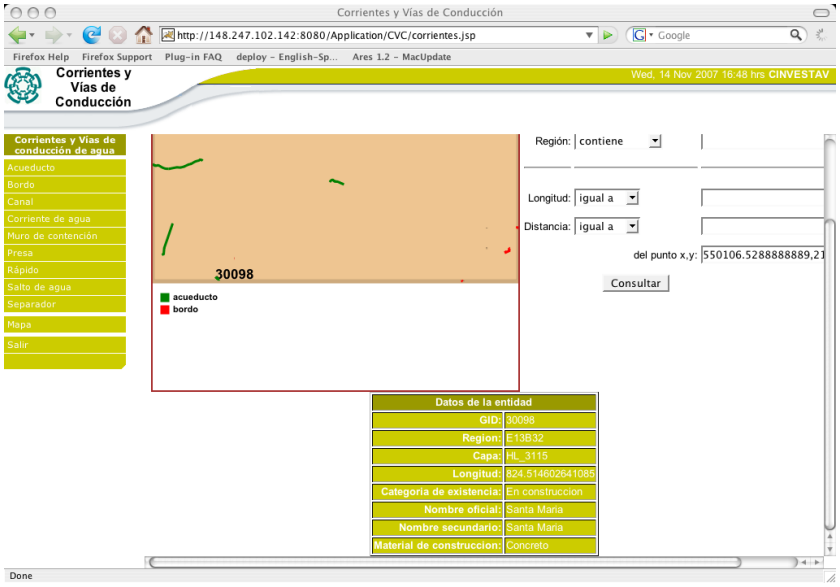

Figura 6.9: Información nominal del *Acueducto* seleccionado por el usuario

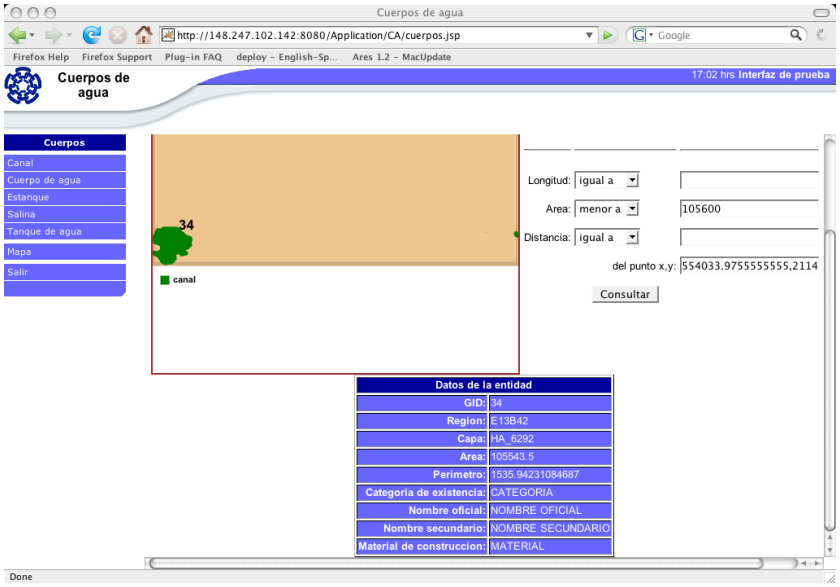

Figura 6.10: *Canales* existentes

6.3.4.2. Modo de comportamiento: agregar

En este modo se agregan datos tanto nominales como geográficos de la entidad en cuestión. La descripción correspondiente a este modo es la *transacción*. La primera página contiene un formulario para que el usuario capture datos. Para definir esta primera página se hace la descripción de la misma en una *forma básica*. Los elementos del formulario son precisamente los campos de la entidad. Cada elemento se va definiendo con etiquetas, cajas de texto, listas desplegables y botones. Para agregar la entidad geográfica se hace la carga de un archivo de texto que contiene las coordenadas de la entidad. En esta página se invoca el catálogo del sistema para indicar a qué capa pertenece la entidad. Al final del formulario se incluye un botón para realizar la inserción invocando una página que hace la transacción.

La figura 6.11 muestra el formulario con los datos capturados.

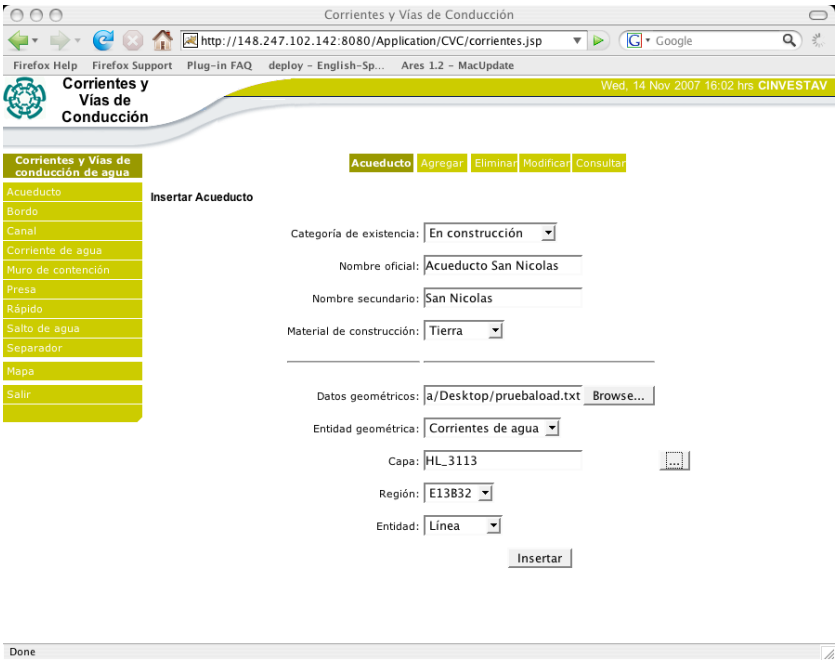

Figura 6.11: Captura del formulario para una inserción

Antes de invocar la siguiente página, se realiza una validacion para que los datos sean correctos en cuanto al tipo de dato y longitud, es decir, que el usuario no pueda introducir letras cuando se requieren números o viceversa y que la longitud de los mismos sea la adecuada. Con la validación se garantiza que a la hora de hacer la inserción no se produzcan errores de tipos de datos y longitudes. Una vez hecha la validación, se invoca la página que invoca al procedimiento almacenado correspondiente y mediante una ventana de aviso se indica si la entidad se ha insertado o no.

6.3.4.3. Modo de comportamiento: eliminar

En este modo se eliminan datos de una entidad. Este requerimiento está asociado a la descripción de una *forma de transacción*. Se debe elegir primero la entidad que se desea eliminar, por lo tanto, debe desplegarse una página con un formulario de búsqueda similar al que se despliega en el modo de consulta. Básicamente, se sigue un procedimiento similar al de consulta hasta la página donde se hace el detalle de la entidad. La diferencia es que en esta última página se agrega una post acción para invocar la página que realice la eliminación.

Antes de invocar la página para realizar la eliminación se despliega una ventana de confirmación y si la respuesta es afirmativa entonces se invoca al procedimiento almacenado correspondiente y se despliega una ventana de aviso para indicar que la entidad se ha eliminado.

La figura 6.12 muestra el punto en el que el usuario ha seleccionado la entidad que desea eliminar.

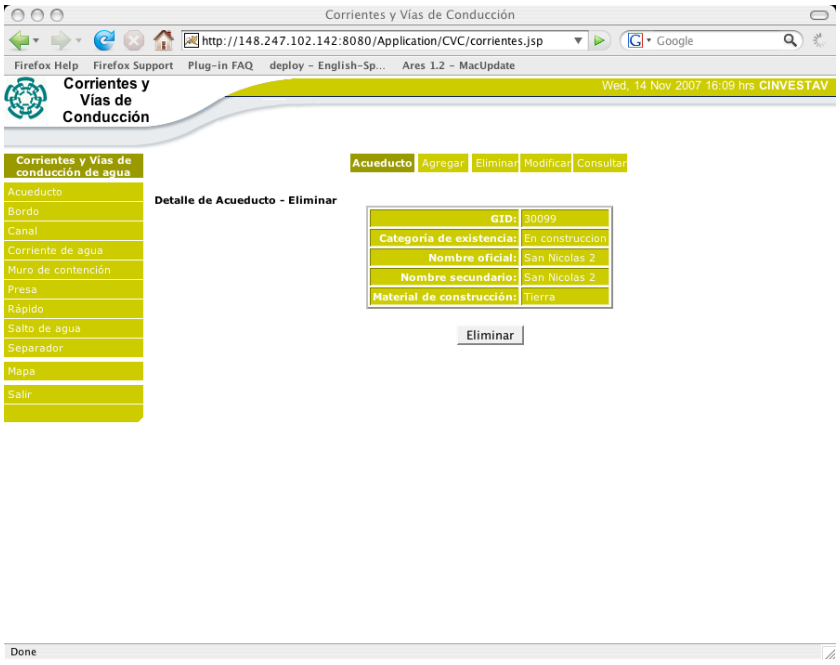

Figura 6.12: Eliminación de una entidad

6.3.4.4. Modo de comportamiento: actualizar

Para realizar la actualización de una entidad es necesario hacer la búsqueda de la entidad que se desea actualizar. Se sigue un procedimiento similar al del modo de consulta hasta

llegar a la página de detalle de la entidad. En la página de detalle se agrega una pos acción para invocar la página que realiza la actualización. Debido a que ya se tiene conocimiento de qué entidad va a ser modificada, la descripción de *transacción* en su modalidad de actualización contiene una *forma básica* que formará una página con un formulario similar al de la inserción pero con la información de la entidad para que el usuario la modifique. De la misma manera que en la inserción el último elemento del formulario es un botón que invoca la página que realizará la transacción.

Antes de hacer dicha invocación se hace la validación de la forma similar que en la inserción. La página siguiente invoca al procedimiento almacenado correspondiente y despliega una ventana de aviso para indicar que la información de la entidad se ha actualizado.

La figura 6.13 muestra el formulario de la entidad que se ha de actualizar con los datos existentes.

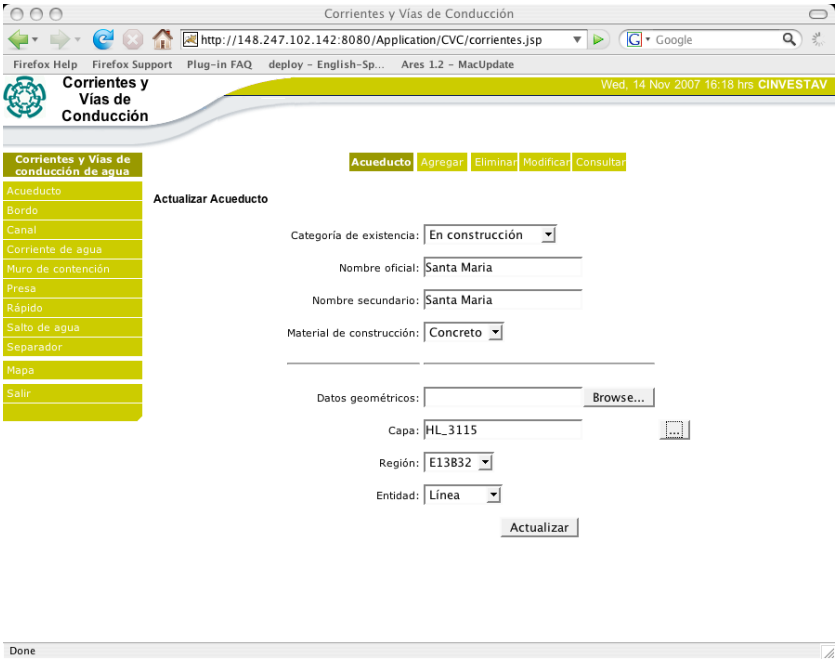

Figura 6.13: Actualización de un *Acueducto*

Para los módulos de *Cuerpos de Agua* y *Rasgos Hidrográficos Puntuales* la definición de los *Modos de Comportamiento* es similar.

Todas las páginas mencionadas deben estar registradas en la descripción de *Páginas* para indicar su ubicación en el árbol de directorios de la aplicación Web y los *beans* que ha de utilizar.

Cada opción del menú tiene una navegación similar a la mostrada en las figuras de esta sección. De la misma manera, los módulos *Rasgos Hidrográficos Puntuales* y *Cuerpos de Agua* tienen un funcionamiento parecido para cada una de sus entidades. Si algún requerimiento debe ser modificado, basta con editar las descripciones, hacer las modificaciones correspondientes y re generar el código.

6.4. Conclusiones

El caso de estudio elegido fue adecuado para mostrar las capacidades de la aplicación, La base de datos de Colima, contiene los elementos gráficos necesarios para mostrar operaciones geográficas dentro de la aplicación Web. Así, se puede visualizar la asociación entre los requerimientos iniciales y la aplicación final funcionando dentro de un servidor de aplicaciones Web. Esta asociación permite variar los requerimientos y obtener la visualización de los cambios realizados en una forma automática.

Capítulo 7

Conclusiones

La generación automática de código es una técnica que ofrece grandes ventajas para acelerar el desarrollo de un sistema, pues representa un ahorro de tiempo y dinero. Es muy común que las aplicaciones estén desarrolladas por diferentes personas, en diferentes tiempos y con una documentación escasa, lo que provoca un problema a la hora de aplicar mantenimiento a las aplicaciones. La definición de requerimientos no es constante, es decir, la especificación inicial puede ir cambiando conforme el sistema se va desarrollando. Incluso al finalizar el desarrollo se pueden hacer modificaciones a los requerimientos. A partir de aquí surge la necesidad de crear una herramienta que genere una aplicación Web que refleje el cambio de requerimientos de manera automática.

De acuerdo a los objetivos planteados inicialmente se propuso la construcción de una herramienta de generación automática de código para entornos Web. Especialmente para el manejo de datos complejos como lo son los datos geográficos, se propuso que los sistemas que se pudiesen generar con la herramienta estuvieran enfocados a Sistemas de Información Geográfica. Mediante descripciones basadas en XML y GML se realizó el diseño del sistema. Éstas descripciones son la entrada al generador de código que procesan la información contenida en las mismas y genera código ejecutable y código que ha de ser compilado. También genera procedimientos que han de ser cargados en la Base de Datos para el manejo de transacciones. Para probar la funcionalidad del generador de código se eligió la capa de hidrología del modelo de datos del estado de Colima. De esta manera, se genera un sistema Web con manejo de datos geográficos.

Para el primer ciclo de generación de código se obtiene el sistema Web funcionando dentro de un servidor de aplicaciones como lo es Tomcat. Sin embargo, como sucede en todos los desarrollos, pueden solicitarse cambios en los requerimientos iniciales, cambios en el modelo de datos o modificaciones a parámetros de diseño, tales como:

- Elementos de los formularios
- Colores, tipos y tamaño de letra en las interfaces principales, menús y submenús
- Despliegue de información de resultados

- Imágenes

Incluso se puede solicitar la adición de nuevos requerimientos. Para estas solicitudes de cambios, sólo es necesario hacer las modificaciones correspondientes en las descripciones y realizar un nuevo ciclo de generación de código. Este procedimiento se repite hasta que no existan más modificaciones, por lo tanto, es claro ver que el sitio Web final se obtendrá tras realizar varios ciclos de generación de código. Con lo anterior, se cumple el objetivo de cubrir la variabilidad en los requerimientos que se definen en todo sistema.

Otro beneficio y objetivo cubierto es la generación de subsistemas. Debido a que el comportamiento del sistema se divide en módulos, la generación de subsistemas se realiza de manera automática. Incluso si se desea la regeneración de código de uno de los subsistemas es posible realizarlo, sin tener que hacer un ciclo de generación del sistema completo.

Uno de los objetivos particulares de este trabajo es realizar la generación de código de modo que la codificación manual sea mínima. Con la estructura que tienen las descripciones y por lo tanto la posible estructura de las aplicaciones que pueden generarse mediante la herramienta, la codificación manual es nula.

Finalmente el Sistema de Información Geográfica que puede generarse mediante la herramienta será de utilidad para la toma de decisiones como se había planteado en un principio. La gestión de construcción o cancelación de acueductos, canales, tanques, etc. puede realizarse mediante el sitio Web, para posteriormente visualizar las nuevas entidades en el mapa y de ahí consultar sus datos. Este tipo de sistemas y en particular el sitio Web generado con el caso de estudio puede ser de utilidad para la Comisión Nacional de Agua de Colima o cualquier institución relacionada.

7.1. Discusión

Dentro de esta sección se muestra un comparativo donde se presentan las ventajas que proporciona el trabajo de tesis con las herramientas y los desarrollos existentes. Es importante mencionar que ninguna herramienta mostrada en la tabla 7.1 provee una generación completa de su código, esto implica que el desarrollador debe realizar codificación manual. El presente trabajo realiza una completa generación de código lo que representa una ventaja importante contra las herramientas existentes.

Herramienta	Características	Comparativo con la Tesis
XDoclet [31]	Uso de comentarios javadoc como fuente de descripción	Uso de XML para la generación de descripciones
SOUL [16]	Generación de aplicaciones básicas	Generación de aplicaciones con información compleja
Apps Java [17]	Enfocado principalmente a la generación de interfaz	Enfocado a generación de todas las capas de la aplicación
WGenerator [15]	Necesaria la programación manual en la lógica de negocio	Se busca que la programación manual sea mínima

Tabla 7.1: Herramientas existentes

7.2. Trabajo futuro

El trabajo que puede desarrollarse a partir de este proyecto es considerable. La aplicación que se ha desarrollado tiene una estructura definida y aunque tiene muchos parámetros que pueden ser configurados, la información de las descripciones tiene un formato predefinido. Modelar comportamiento no es una tarea sencilla, razón por la cual se estableció un modelo de descripciones para el generador de código. Considerando lo anterior, se pueden realizar varias tareas que den seguimiento al proyecto realizado, entre ellas se pueden mencionar:

- Generar más estructuras para modelar el comportamiento del sistema. Dichas estructuras pueden involucrar las relaciones entre los módulos que ya han sido definidos en este trabajo. Se podría agregar un módulo para el *pool* de conexiones a la base de datos. Con esto se puede generar código con un comportamiento más complejo.

- Agregar más estructuras a la interfaz, de modo que el diseño que se haga en las descripciones genere diseños de interfaces completamente diferentes. Es decir, agregar más plantillas para la generación de interfaces con niveles de menús más profundos.

- Posibilidad de integrar varias capas del modelo geográfico en el sistema. De este modo, se puede visualizar el mapa con mayor información.

- Agregar más funcionalidad al mapa. Crear más funcionalidades para interacción con el usuario. Selección y arrastre de elementos geográficos, selección de áreas sobre el mapa, etc.

- Agregar al modelo de datos de Colima, un modelo de datos de contaminantes para la generación de un Sistema de Información Geográfica orientado a la gestión y control de contaminación de aguas en el estado.

Bibliografía

[1] Douglas C. Schmidt, *Model-Driven Engineering*, Published by the IEEE Computer Society Vol. 39, Febrero 2006, págs. 25-31.

[2] Spencer Rugaber, Kurt Stirewalt, *Model-Driven Reverse Engineering*, Published by The IEEE Computer Society IEEE Software, Vol. 21, E.U., 2004, págs. 45-53.

[3] Krishnakumar Balasubramanian, Aniruddha Gokhale, Gabor Karsai, Janos Sztipanovits, and Sandeep Neema, *Developing Applications Using Model-Driven Design Environments*, Published by The IEEE Computer Society IEEE Software, Vol. 39, E.U., 2006, págs. 33-40.

[4] YE Qing, XU Ran-Bin, MA Xun, LIU Yan-Hong, *The application of Web Technology to Geographic Information System*, Proceedings of International Conferences on Info-Tech and Info-net, Beijing, 2001, págs. 267-272.

[5] Sergio Chapa, *Sistema de Información Geográfica para el Manejo Sustentable de Cuencas Hidrológicas: Fundamentos, Desarrollo y Enfoques*, Proyecto mixto Conacyt-Colima, 2006, págs. 19-22, 38.

[6] Stonebraker, M., Rowe, L.A., Hirohama, M., *The implementation of POSTGRES*,Transactions on Knowledge and Data Engineering, Vol 2, 1990, págs. 125-142.

[7] Jansle Vieira Rocha, *El Sistema de Informaciones Geográficas (SIG) en los contextos de planificación del medio físico y de las cuencas hidrográficas*, II Curso Internacional de ASPECTOS GEOLOGICOS DE PROTECCION AMBIENTAL, UNESCO - UNICAMP, 2002

[8] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, *Database Systems Concepts*, Ed. MacGraw-Hill, 2005.

[9] Alonso Rodríguez Zamora, *Publicación en Internet y tecnología XML*, Ed. Alfaomega, 2004, págs. 115-120.

[10] Danny Goodman, *JavaScript Bible*, Ed. Hungry Minds, 2001.

[11] Bruce Eckel, *Thinking in Java*, Ed. Prentice-Hall, 2002.

[12] Jaideep Roy, Anupama Ramanujan, *XML Schema Language: Taking XML to the Next Level*, IT Professional Vol. 3, 2001, págs. 37-40.

[13] Bartlett, R.G. Cook, M.W. , *XML security using XSLT*, Proceedings of the 36th Annual Hawaii International Conference on System Sciences, (HICSS'03), Australia, 2003, pág. 6.

[14] João Araújo Ribeiro, Oscar Luis Monteiro de Farias, Luis Alberto Oliveira Lima Roque, *A Syntactic and Lexicon Analyzer for the Geography Markup Language (GML)*, Proceedings of Geoscience and Remote Sensing Symposium (IGARSS'04), Brazil, 2004, págs. 2896-2899.

[15] Jia Zhang, Ugo Buy, *A Framework for the Efficient Production of Web Applications*, Proceedings of Eighth IEEE International Symposium on Computers and Communications (ISCC'03), 2003, págs. 419-424

[16] Tom Tourwé, Luk Stoops, Stijn Decneut, *Automated Support for Data Exchange via XML*, Proceedings of IEEE Fifth International Symposium on Multimedia Software Engineering (ISMSE'03), 2003, págs. 70-77.

[17] Branko Milosavljevi, Milan Vidakovi, Konjovi Zora, *Applications of java programming: Automatic code generation for database-oriented web applications*, Proceedings of The inaugural conference on the Principles and Practice of programming, 2002, págs. 59-64.

[18] Alan W. Biermann, Gérard Guiho, Yves Kodratoff, *An Overview of Automatic Program Construction Techniques*, MacMillan Publishing Company, New York, 1984, págs. 3-10.

[19] Sheng Ye, Feng Xuezhi, Shaotao Yuan, Li Juliang, *Visualization GML with SVG*, Proceedings of The IEEE International Geoscience and Remote Sensying Symposium (IGARSS'05), Vol. 5, pp. 3648 - 3651, China, 25-29 July 2005.

[20] W.T.M.S.B. Tennakoon, *Visualization of GML data using XSLT*, International Institute for Geo-Information Science and Earth Observation, Enschede, Holanda, 2003.

[21] Jack Herrington, *Code Generation in Action*, Ed. Manning, 2003, págs. 4-27, 62-96.

[22] Paul Ramsey, *PostGIS Manual*, Refractions Research Inc, 2005.

[23] The Jakarta Project, *Taglibs*, http://jakarta.apache.org/taglibs/doc/xtags-doc/intro.html, The Apache Software Foundation, 2004.

[24] James Clark, *XSL Transformations (XSLT) Version 1.0*, http://www.w3.org/TR/xslt, World Wide Web Consortium, 1999.

[25] Arnaud Le Hors, Ian Jacobs, *HTML 4.01 Specification*, http://http://www.w3.org/TR/REC-html40/, World Wide Web Consortium, 1999.

[26] Steven Pemberton, Daniel Austin, *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*, http://www.w3.org/TR/xhtml1/, World Wide Web Consortium, 2002.

[27] Chris Lilley, Chair, and Doug Schepers, *Scalable Vector Graphics, XML Graphics for the Web*, http://www.w3.org/Graphics/SVG/, World Wide Web Consortium, 2007.

[28] The PostgreSQL Global Development Group, *PostgreSQL*, http://www.postgresql.org/docs/7.4/interactive/index.html, The PostgreSQL Project, 1996-2007.

[29] Jason Hunter, *JDOM*, http://www.jdom.org/, The JDOM Project, 2000.

[30] Apache Maven Project, *dom4j*, http://www.dom4j.org/, Project Object Model, 2005.

[31] Jack Herrington, *Code-Generation Techniques for Java*, http://www.onjava.com/pub/a/onjava/2003/09/03/generation.html, o'reilly on java, 2003.

[32] Open Geospatial Consortium, *GML - the Geography Markup Language*, http://www.opengis.net/gml/, Especificación GML 3.1.1, 2004.

[33] ESRI Group, *ESRI, GIS and Mapping Software*, http://www.esri.com/software/arcgis/index.html, ArcGIS the complete GIS, 1995-2007.

[34] ESRI Group, *Information Technology Association of America*, http://www.itaa.org/, 2008.

[35] MapInfo Corporation, *MapInfo*, http://www.mapinfo.com/location/integration, 2007.

[36] Sun Microsystems, *NetBeans*, http://www.netbeans.org/, 2007.

[37] GRASS Development Team, *Geographic Resources Analysis Support System*, http://grass.itc.it/, 1999-2007.

[38] OMG Technology Committee, *Object Management Group*, http://www.omg.org/mda/, 2007.

[39] Noe Sierra Romero, Sergio V. Chapa Vergara, *Diseño de Interfaces Visuales*, http://www.cs.cinvestav.mx/CursoVis/prinvisual.html, 2005.

[40] Flor Radilla López, *Modelado de Datos para Bases de Datos Espaciales: Caso de Estudio SIG*, Tesis de Maestría, Centro de Investigación y Estudios Avanzados del IPN CINVESTAV, 2007

[41] Teresa Villegas Casas, *Evex Mac: Herramienta para el diseño de base de datos*, Tesis de Maestría, Centro de Investigación y Estudios Avanzados del IPN CINVESTAV, 2005