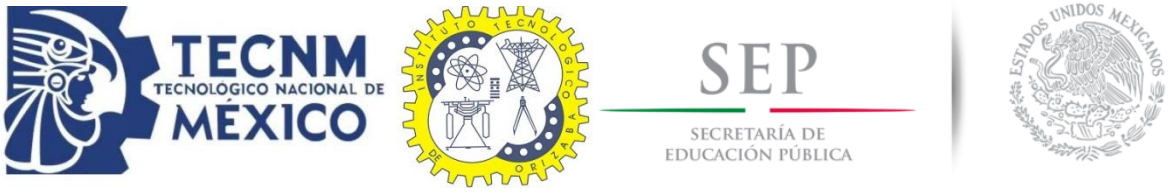

Instituto Tecnológico de Orizaba

División de Estudios de Posgrado e Investigación

Maestría en Sistemas Computacionales

TESIS

Desarrollo de un generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML

PRESENTADO POR:

Selene Estévez Gámez M08011262

PARA OBTENER EL GRADO DE:

Maestro en Sistemas Computacionales

DIRECTOR:

M.C.E. Beatriz Alejandra Olivares Zepahua

Orizaba, Veracruz, México 2018

Agradecimientos

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Tabla de acrónimos

Términos	Conceptos
AJAX	*"Asyncronous JavaScript And XML",* JavaScript asíncrono y XML
ANTLR	*"ANother Tool for Language Recognition"*, Otra Herramienta para el Reconocimiento de Lenguaje
API	*"Application Programming Interface"*, Interfaz de Programación de Aplicaciones
AST	*"Abstract Syntax Tree"*, Árbol de Sintaxis Abstracto
ATL	*"Active Template Library"*, Biblioteca de Plantillas Activa
BPMN	*"Business Process Model and Notation"*, Modelo y Notación de Procesos de Negocio
BSD	*"Berkeley Software Distribution"*, Distribución de Software Berkeley
CASE	*"Computer Aided Software Engineering"*, Ingeniería de Software Asistida por Computadora
CDDL	*"Common Development and Distribution License"*, Licencia Común de Desarrollo y Distribución
CIM	*"Computer Integrated Manufacturing"*, Manufactura Integrada por Computadora
CPC	*"Cross Platform Code"*, Código de plataforma cruzada
CRUD	*"Create, Read, Update and Delete"*, Crear, Leer, Actualizar y Borrar
CSS	*"Cascading Style Sheets"*, Hojas de Estilo en Cascada
CSV	*"Comma-Separated Values",* Valores Separados por Comas
DOM	*"Document Object Model"*, Modelo de Objetos del Documento
EBNF	*"Extended Backus–Naur Form"*, Notación de Backus-Naur Extendida
EJB	*"Enterprise JavaBeans"*, JavaBean Empresarial
EMF	*"Eclipse Modeling Framework",* Marco de modelado de Eclipse
FSM	*"Framework Specific Model"*, Modelo específico del marco de trabajo
GLR	*"Generalized LR"*, LR Generalizado

Términos	Conceptos
GNU	*"GNU's Not Unix"*, GNU No es Unix
GPL	*"GNU General Public License"*, Licencia Pública General de GNU
GTK	*"GIMP Toolkit"*, Biblioteca GIMP
GUIs	"*Graphical User Interface*", Interfaz Gráfica de Usuario
HTML	"*HyperText Markup Language"*, Lenguaje de Marcado para Hipertextos
HTTP	*"Hypertext Transfer Protocol"*, Protocolo de Transferencia de Hipertexto
ID	Identificador
IDE	*"Integrated Development Environment"*, Entorno de Desarrollo Integrado
IFML	*"Interaction Flow Modeling Language"*, Lenguaje de Modelado de Flujos de Interacción
IIS	*"Internet Information Services"*, Servicios de Información de Internet
JAXB	*"Java Architecture for XML Binding"*, Arquitectura Java para Vinculación XML
JDBC	*"Java Database Connectivity"*, Conectividad a Base de Datos de Java
JSF	*"JavaServer Faces"*
JSON	*"JavaScript Object Notation"*, Notación de Objetos JavaScript
JSP	*"JavaServer Pages"*
LL	"*Left-to-right*" y "*Leftmost derivation*", De izquierda a derecha y Derivación a la izquierda
LR	"*Left-to-right*" y "*Rightmost derivation*", De izquierda a derecha y Derivación a la derecha
M2T	"*Model to Text*", Modelo a Texto
MDA	*"Model Driven Architecture"*, Arquitectura dirigida por modelos
MDD	"*Model Driven Development*", Desarrollo Dirigido por Modelos
MDE	*"Model-driven Development Environment"*, Entorno de desarrollo dirigido por el modelo

Términos	Conceptos
MDWE	*"Model Driven Web Engineering"*, Ingeniería de la Web dirigida por modelos
MIT	*"Massachusetts Institute of Technology"*, Instituto Tecnológico de Massachusetts
MOF	*"Meta-Object Facility"*, Mecanismo de Meta-Objeto
MVC	*"Model-View-Controller"*, Modelo-Vista-Controlador
MVP	*"Model-View-Presenter"*, Modelo-Vista-Presentador
NC	*"Native Code"*, Código nativo
ODBC	*"Open DataBase Connectivity*", Conectividad abierta de bases de datos
ODM	*"Ontology Definition MetaModel"*, Metamodelo de Definición de la ontología
OMG	"*Object Management Group*", Grupo de Administración de Objetos
OWL2	*"Web Ontology Language"*, Lenguaje de Ontología Web
PCN	*"Place Chart Nets"*, Gráfico de Redes de Sitio
PEG	*"Parsing Expression Grammar"*, Gramática de Expresión de Análisis Sintáctico
PHP	*"Hypertext Preprocessor"*, Preprocesador de Hipertexto
PIM	*"Platform Independent Models",* Modelos independientes de la plataforma
PSM	*"Platform Specific Model"*, Modelo específico de plataforma
QVT	*"Query/View/Transformation"*, Consulta/Vista/Transformación
RESTful	*"Representational State Transfer" (REST)*, Transferencia de Estado Representacional
RIAs	"*Rich Internet Applications"*, Aplicaciones Enriquecidas de Internet
SAX	*"Simple API for XML"*, API Simple para XML
SFR	*"Special Function Registers"*, Registros de Funciones Especiales
SoaML	*"Service Oriented Architecture Modeling Language",* Lenguaje de Modelado de Arquitectura Orientada a Servicios

Términos	Conceptos
SQL	*"Structured Query Language"*, Lenguaje Estructurado de Consulta
SWF	*"Small Web Format",* Formato Web Pequeño
SysML	*"Systems Modeling Language"*, Lenguaje de Modelado de Sistemas
UI	*"User Interface"*, Interfaz de Usuario
TI	Tecnología de la Información
UML	*"Unified Modeling Language"*, Lenguaje Unificado de Modelado
UUID	*"Universally Unique Identifier"*, Identificador Único Universal
W3C	*"The World Wide Web Consortium"*, El Consorcio WWW
XMI	*"XML Metadata Interchange"*, XML de Intercambio de Metadatos
XML	*"eXtensible Markup Language"*, Lenguaje de marcado extensible

Resumen

El proceso de desarrollo de Aplicaciones Enriquecidas de Internet es cada vez más exigente pues es necesario cumplir con requerimientos como lo son interfaces de usuario vistosas y provistas de diversas funcionalidades, así como el uso de combinaciones de lenguajes de programación, lo cual origina que las aplicaciones se dividan en numerosos archivos y al encontrarse tan particionadas se vuelven más complejas de realizar.

Por el motivo expuesto en el párrafo anterior, el objetivo de la investigación es la generación de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando el estándar IFML, el cual abstrae la complejidad de las interfaces de usuario interactivas mediante el modelado del flujo de interacción de la aplicación, y conjuntándolo con el modelado del dominio, el cual se ve representado como una variación al diagrama de clases del también estándar UML.

El generador de aplicaciones propuesto obtiene esqueletos de RIAs en las principales combinaciones de lenguajes de programación actualmente empleadas para su desarrollo, las cuales son JavaServer Faces, estándar de Oracle para el desarrollo de interfaces de usuario del lado del servidor, enriquecido con PrimeFaces y la combinación de PHP, ampliamente utilizado en el mercado, junto con jQuery, dichas aplicaciones acatan las características estipuladas por el usuario en cuanto a distribución de su contenido, temas, y navegabilidad de acuerdo a los modelos IFML y UML dados también por el usuario.

Para la construcción de la herramienta, la metodología en espiral dirigió el proceso de desarrollo; como fuente del modelo navegacional en IFML se empleó la herramienta WebRatio, la cual proporciona una descripción de sus modelos en lenguaje XML; para la obtención de los diagramas de clases UML se utilizó su descripción de acuerdo a la especificación del estándar XMI, y por último, como instrumento para el reconocimiento de lenguajes se usó ANTLR4, el cual es un potente generador de analizadores sintácticos ampliamente utilizado para construir lenguajes, herramientas y marcos de trabajo.

Abstract

The process of developing Enriched Internet Applications is increasingly demanding as it is necessary to satisfy requirements such as user-friendly interfaces and provided with various functionalities, as well as the use of combinations of programming languages, which causes applications to divided into numerous files and when they are so partitioned they become more complicated make.

For the aforementioned, the objective of the research is the generation of Enriched Internet Applications modeled under the MVC architectural standard using the IFML standard, which abstracts the complexity of interactive user interfaces describing the interaction flow of the application, and combining it with the modeling of the domain, which is represented as a variation to the class diagram of the also UML standard.

The proposed application generator will obtains RIAs skeletons in the main combinations of programming languages currently used for their development, which are JavaServer Faces, Oracle standard for the development of server-side user interfaces, enriched with PrimeFaces and the combination of PHP, widely used in the market, together with jQuery, the generated applications have characteristics stipulated by the user in terms of distribution of content, themes, and navigability according to the IFML and UML models also given by the user.

For the construction of the tool, the spiral methodology guides the development process; as a source of the navigational model in IFML, the WebRatio tool is utilized, which provides a description of its models in XML language; To obtain the UML class diagrams, its information is handled according to the XMI standard specification, and finally, as an instrument for the recognition of languages, ANTLR4 is employed, which is a powerful generator of used syntactic analyzers to build languages, tools, and frameworks.

Introducción

Las RIAs cuentan con diversas características, entre las cuales se encuentran velocidad, atractivo visual, capacidad, funcionalidad y rendimiento, por mencionar algunas, lo cual conlleva a que en el desarrollo de este tipo de aplicaciones se considere una vasta cantidad de detalles sobre elementos que se ejecutan en ambientes distintos tanto en tecnología como en capacidad, haciendo indispensable el uso de técnicas de Ingeniería de Software para asegurar que las aplicaciones cumplen con las expectativas del usuario.

Con base en lo anterior, el presente proyecto de tesis tiene como objetivo principal el desarrollo de un generador de esqueletos de RIAs modeladas bajo el patrón arquitectónico MVC usando como entrada la descripción de modelos IFML y diagramas de clases UML en los lenguajes XML y XMI, respectivamente.

Con la intención de brindar al lector un claro entendimiento del proyecto de tesis, el presente documento se encuentra compuesto por cinco capítulos.

Dentro del capítulo uno titulado “Antecedentes” es descrito el marco teórico, planteamiento del problema, objetivos generales y específicos y la justificación que llevó a la elaboración del presente trabajo. El capítulo dos, “Estado de la práctica”, expone resúmenes de investigaciones ya desarrolladas que cuentan con un grado de similitud a la propuesta ofrecida dentro de este documento y se muestra un análisis comparativo que asiste al lector para hacer una clara distinción entre las investigaciones mostradas y el presente trabajo. El tercer capítulo “Aplicación de la metodología”, contiene la narrativa del desarrollo de la aplicación. El cuarto capítulo, “Resultados”, presenta el caso de estudio planteado junto con la herramienta desarrollada en operación y el funcionamiento de las aplicaciones entregadas en cada una de las tecnologías disponibles. En el último capítulo, “Conclusiones y recomendaciones”, se tienen las conclusiones y recomendaciones de este trabajo.

Capítulo 1. Antecedentes

En este capítulo se presenta el marco teórico del proyecto, se describen los conceptos básicos dentro del contexto del proyecto de tesis, así como también se presenta la problemática existente que origina el desarrollo del presente trabajo, los objetivos generales, específicos y la justificación del mismo.

1.1 Marco teórico

En este apartado se explican los conceptos más importantes relacionados con el trabajo presentado.

1.1.1 Aplicación Web

Algunas definiciones de aplicación Web son: a) Un sistema que permite a sus usuarios la ejecución de la lógica de negocios por medio de un visualizador Web, y b) Un sitio en donde la interacción de un usuario (entrada de datos y navegación en la página) afecta el estado del negocio.

La distinción entre un sitio Web y una aplicación Web recae en la capacidad que tiene un usuario para interactuar con en el servidor, si no existe una lógica de negocio en un servidor, el sistema no es una aplicación Web. Para la mayoría de las aplicaciones Web, excepto las más sencillas, el usuario debe introducir más información que la simple solicitud de navegación, esto normalmente se realiza por medio de campos de entrada de algún tipo en formularios [1].
Entre las características típicamente halladas en las aplicaciones Web tradicionales se tiene que:

- Se basan en una arquitectura o infraestructura Web existente.
- Cuentan con independencia de la plataforma y del lenguaje de programación.
- Promueven la reutilización de la aplicación más allá del visualizador Web.
- Habilitan su composición con otras aplicaciones Web o de escritorio.
- Exigen claridad semántica en el contenido intercambiado durante su uso [2].

1.1.2 Aplicación Enriquecida de Internet

Macromedia define Aplicación Enriquecida de Internet como la combinación de las mejores características en las interfaces de usuario para aplicaciones de escritorio junto con las mejores características de las aplicaciones Web y con una comunicación multimedia interactiva (incorporación de audio y video interactivo bidireccional) para obtener aplicaciones que proporcionan una experiencia de usuario intuitiva, responsiva y efectiva.

Al mencionar las mejores características de interfaz de usuario en las aplicaciones de escritorio se hace alusión a su capacidad de proporcionar una interacción en la verificación de formatos y validaciones, tiempos de respuesta rápidos sin recarga de página, comportamientos arrastrar y soltar y su capacidad de trabajar con o sin conexión a Internet.

Entre las mejores características de las aplicaciones Web se incluyen la capacidad de un despliegue instantáneo, disponibilidad entre plataformas, la realización de descargas progresivas para la recuperación de contenido y datos, su diseño compartido entre páginas y la amplitud de la adopción de los estándares de Internet [3].

Como parte de las características generales de las RIAs se encuentran:

1. **Interacción:** Especificar el comportamiento a los usuarios activos.
2. **Multimedia:** Soportar la representación de gráficos, audio, video, *streaming* y multimedia en vivo.
3. **Continuidad visual:** Evitar la recarga de página y experiencias de parpadeo.
4. **Sincronización:** Suministrar una representación activa (relacionada con la interacción del usuario) y pasiva (relacionada con comportamientos predefinidos).
5. **Desarrollo de N-Niveles:** Proporcionar separación en capas.
6. **Recuperación dinámica de los datos:** Transferencia de datos parcial desde/hacia el servidor en tiempo de ejecución.
7. **Respuestas paralelas de orígenes diferentes:** Recuperar datos de uno o más orígenes simultáneamente (de forma síncrona o asíncrona).

8. **Personalización:** Extensiones para internalización y localización, acceso a múltiples dispositivos, entre otros [4].

1.1.3 Arquitectura de software

En esencia, la arquitectura de un sistema de software se define como "el conjunto de decisiones de diseño principales tomadas sobre el sistema" [5].

Dichas decisiones abarcan todos los aspectos del sistema en desarrollo, como lo son:

- Decisiones de diseño relacionadas con la estructura del sistema.
- Decisiones de diseño relacionadas con el comportamiento funcional.
- Decisiones de diseño relacionadas con la interacción entre el usuario final y el sistema o bien entre los elementos del sistema.
- Decisiones de diseño relacionadas con las propiedades no funcionales del sistema.
- Decisiones de diseño relacionadas con la implementación del sistema.

Otra definición es que una arquitectura de software es la descripción y relaciones existentes entre los subsistemas y componentes computacionales de un sistema de software [6].

Y otro enfoque de arquitectura de software es que se trata de la forma en que los componentes de un sistema se encuentran organizados, su forma de interactuar entre ellos y con el contexto, siguiendo la aplicación de normas y principios de calidad que impulsan a que el sistema se encuentre preparado para su propia evolución [7].

Cabe hacer mención de que en un arquitectura de software convergen tres elementos esenciales: a) Los modelos que definen la estructura, topología y dinámica del sistema; b) La trazabilidad de dichos modelos con los requisitos establecidos que debe cumplir el software, y c) Las reglas, principios y justificaciones que rigen la arquitectura y que sustentan las decisiones tomadas [8].

1.1.4 Patrón arquitectónico

Se define patrón arquitectónico como "una colección con nombre de decisiones de diseño arquitectónico que son aplicables a un problema de diseño recurrente, con una parametrización que tiene en cuenta el contexto de desarrollo de software diferente al que aparece ese problema" [5].

Un patrón arquitectónico proporciona un conjunto de decisiones de diseño específicas, que se comprueban como eficaces, que organizan determinadas clases del sistema o subsistemas específicos. Entre sus características se encuentran que su alcance es el de un problema de diseño específico, son elementos arquitectónicos que cuentan con una parametrización para ser empleados como fragmentos específicos de un diseño y que en un sistema es posible la aparición de múltiples patrones [5].

1.1.5 MVC

El patrón arquitectónico Modelo-Vista-Controlador marca una clara separación entre los datos y la lógica del negocio de una aplicación de la interfaz de usuario y del módulo responsable de la gestión de eventos y las comunicaciones. En otras palabras, se definen tres componentes distintos, los cuales son el modelo, la vista y el controlador, estableciendo que de un lado se encuentran los elementos para la representación de la información y en otro lado los elementos para la interacción con el usuario [7].

De forma concreta, el papel que desempeña cada componente que integra MVC es:

- **Modelo:** Es el encargado de realizar la representación los datos y reglas del negocio, y maneja un registro de las vistas y de los controladores existentes en el sistema.
- **Vista:** Responsable de presentar la información procedente del modelo de tal forma que se muestre la interacción con el usuario. Adicionalmente cuenta con un registro del controlador asociado y suministra el servicio de actualización que es usado tanto por el controlador como por el modelo.

- **Controlador:** Responde a los eventos que implican cambios en el modelo y la vista, realizando la gestión de las interacciones del usuario [9].

1.1.6 Modelo 2

Modelo 2 es una arquitectura Web recomendada como una buena práctica para el diseño de la capa de presentación en aplicaciones Web, en la cual las solicitudes del usuario se envían hacia componentes en el servidor que ejecutan la lógica de negocio y que, con base en el resultado obtenido por ellos, redirigen las solicitudes a páginas que muestran los resultados de las peticiones. Cabe mencionar que dentro de esta arquitectura las páginas del servidor sólo se emplean con fines de presentación.

Bajo la arquitectura de Modelo 2 los *Servlets* controlan el flujo de la aplicación Web y delegan la lógica empresarial a componentes externos como *JavaBeans* o EJBs (*JavaBeans* Empresariales), y las tareas de generación de código para visualizadores Web se quedan en páginas JSP, con lo cual se logra una reutilización de los componentes de la presentación en el caso de que diversos componentes generen la misma página de salida bajo determinadas condiciones, lo cual cumple con el enfoque de Modelo 2 de ofrecer contenido dinámico [10].

1.1.7 XML

Es una especificación de propósito general para crear lenguajes de marcado personalizados, se clasifica como un lenguaje extensible, ya que permite a sus usuarios definir sus propios elementos y su objetivo principal es ayudar a los sistemas de información a compartir datos estructurados, particularmente a través de Internet, y que es usado tanto para codificar documentos como para la señalización de datos [11]. En la Figura 1.1 se muestra la estructura general de un documento XML.

```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

Figura 1.1 Ejemplo de descripción XML

1.1.8 XMI

XML de Intercambio de Metadatos (XMI) es un formato de intercambio XML concebido para proveer una forma de intercambio de modelos UML entre distintas herramientas de modelado que define los siguientes aspectos implicados en la descripción de objetos en XML:

- La representación de objetos en términos de elementos y atributos XML.
- Los mecanismos estándar para vincular objetos dentro sí mismo o entre otros archivos.
- La validación de documentos XMI utilizando esquemas XML.
- La identidad del objeto, que permite hacer referencia a los objetos desde otros objetos en términos de ID y UUID.

XMI describe soluciones a los puntos anteriormente mencionados mediante la especificación de reglas de producción EBNF para crear documentos XML y esquemas que comparten objetos de forma coherente [12].

En la Figura 1.2. se muestra una parte de la estructura que tiene un documento XMI.

```
<packagedElement xmi:type="uml:Class" xmi:id="c001" name="Empleado">
    <ownedAttribute xmi:id="a001" name="nombre"/>
</packagedElement>
<packagedElement xmi:type="uml:PrimitiveType" xmi:id="t001" name="String"/>
<packagedElement xmi:type="uml:Class" xmi:id="c002" name="Departamento">
    <ownedAttribute xmi:id="a002" name="nombre" type="t001"/>
</packagedElement>
<packagedElement xmi:type="uml:Association" xmi:id="as001" name="trabajaEn"
memberEnd="e001 e002">
<ownedEnd xmi:id="e001" type="c002" association="as001"/>
<ownedEnd xmi:id="e002" name="" type="c001" association="as001">
<upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="un001" value=""/>
</ownedEnd>
</packagedElement>
```

Figura 1.2 Ejemplo de descripción XMI

1.1.9 UML

Es el Leguaje Unificado de Modelado del OMG que especifica, visualiza y documenta modelos de sistemas de software, así como también se emplea para el modelado en general y el modelado de sistemas que no son de software.

La especificación de UML 2.0 cuenta con trece diagramas estándar agrupados en tres tipos que son:

- **Diagramas estructurales:**
    1. Diagrama de clases.
    2. Diagrama de componentes.
    3. Diagrama de despliegue.
    4. Diagrama de objetos.
    5. Diagrama de paquetes.
    6. Diagrama de estructura compuesta.
- **Diagramas de comportamiento:**
    7. Diagrama de actividades.
    8. Diagrama de casos de uso.
    9. Diagrama de máquina de estados.

- **Diagramas de interacción:**
    10. Diagrama general de interacciones.
    11. Diagrama de comunicación.
    12. Diagrama de secuencia.
    13. Diagrama de tiempos.

Con el uso de combinaciones de los diagramas enlistados, es posible modelar cualquier tipo de aplicación ejecutable independientemente del tipo de composición en el hardware requerido, el sistema operativo necesario y el lenguaje de programación empleado para su realización [13].

Cabe mencionar que los modelos UML 2 se serializan en XMI 2 de acuerdo con las reglas especificadas por la especificación de mapeo MOF 2 XMI como lo establece el OMG [14].

1.1.10 Diagrama de clases

El diagrama de clases es un tipo de diagrama UML que muestra los objetos requeridos en la aplicación y las relaciones entre ellos, y dado que proporciona información detallada sobre las propiedades e interfaces de las clases, se considera como el modelo principal para la construcción de la aplicación y el resto de diagramas se toman como modelos complementarios [15].

En otras palabras, el diagrama de clases es un diagrama de estructura UML que muestra la estructura del sistema diseñado a nivel de clases e interfaces, muestra sus características, restricciones y relaciones: asociaciones, generalizaciones, dependencias, entre otros, un ejemplo de la representación del diagrama de clases se observa en la Figura 1.3 [16].

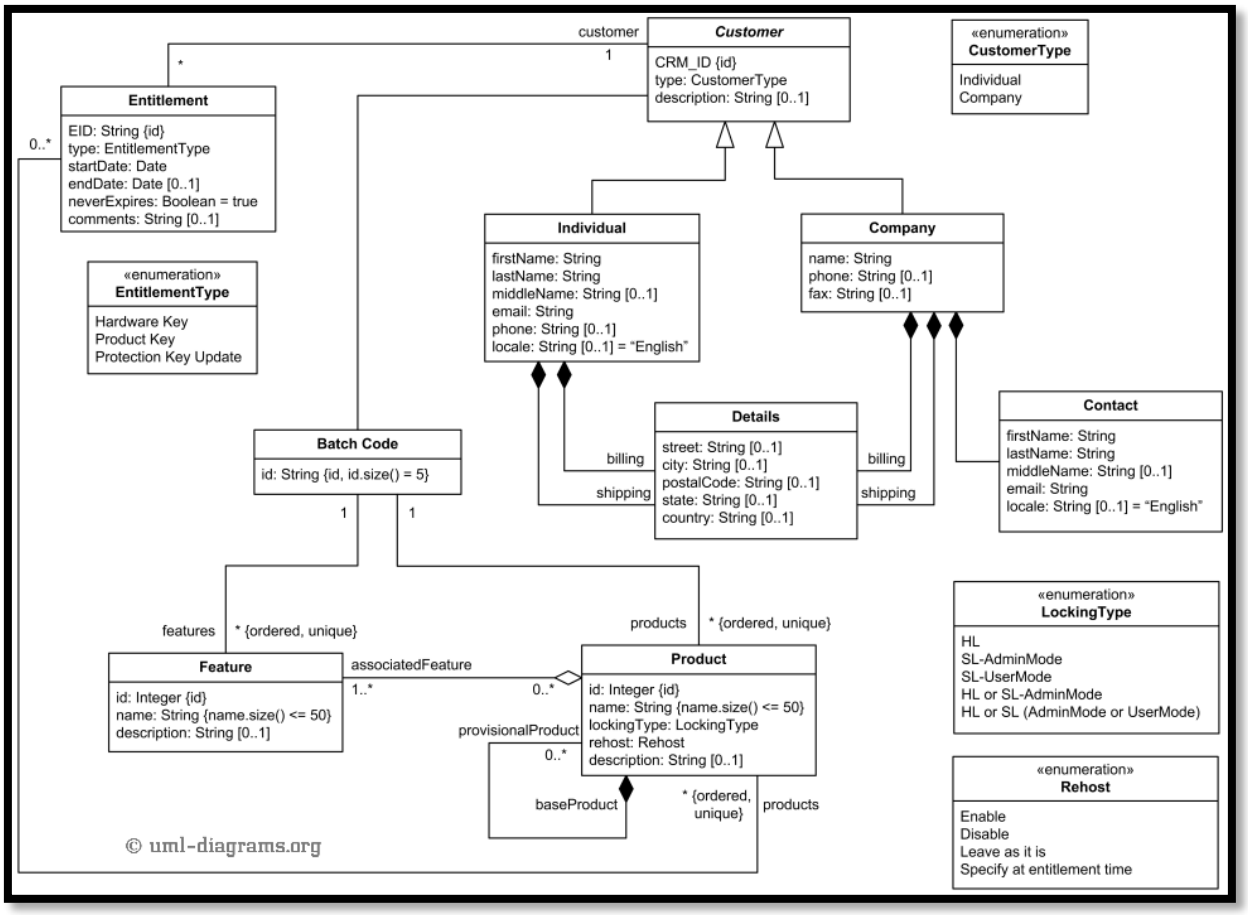

Figura 1.3 Ejemplo de diagrama de clases en UML 2.0

1.1.11 IFML

El Lenguaje de Modelado de Flujo de Interacción es un lenguaje de modelado gráfico cuyo foco principal se centra en la estructura y el comportamiento de la aplicación tal y como lo percibe el usuario final, en otras palabras, cubre el contenido del *front-end* y los mecanismos de interacción con el usuario disponibles en la interfaz de usuario.

Los aspectos de la interfaz cubiertos por IFML son la estructura de la vista, el contenido de la vista, eventos, transiciones de eventos y la vinculación de parámetros, todos condensados dentro del Diagrama de Flujo de Interacción.

IFML se diseñó para expresar el contenido, la interacción con el usuario y el comportamiento del *front-end* de las aplicaciones que pertenecen a los siguientes dominios:

- Aplicaciones Web tradicionales basadas en HTML+ HTTP.
- RIAs soportadas por el estándar HTML5.
- Aplicaciones móviles.
- Aplicaciones cliente-servidor.
- Aplicaciones de escritorio.
- Interfaces usuario-máquina integradas para el control de las aplicaciones.
- Aplicaciones multicanales y contextualizadas.

Dentro del estándar se considera el diagrama de dominio, el cual se ve representado como una variación al diagrama de clases de UML, en el que se realiza la especificación de los activos de información relevantes que constituyen el modelo de dominio de la aplicación modelada, y su importancia recae en que la interfaz hace referencia a objetos que proporcionan contenido a publicarse en el *front-end* de la aplicación, además, los eventos desencadenados dentro de la interfaz pueden provocar la ejecución de operaciones que actualizan objetos y cambian el estado de la interfaz [17].

1.1.12 JSF

JavaServer Faces es un marco de trabajo Web de Java que establece el estándar para construir interfaces de usuario del lado del servidor, introducido por Sun Microsystems para la simplificación del desarrollo de aplicaciones Web, como una evolución del marco de trabajo de JavaServer Pages apegado a la arquitectura MVC empleando archivos XML para la construcción de la vista y clases escritas con Java para la lógica de la aplicación.

El marco de trabajo soporta el uso tecnologías como AJAX para facilitar el desarrollo de contenido dinámico y cuenta con integración a bases de datos por medio de la API JDBC, EJBs o tecnologías RESTful para el trabajo en el *back-end* [18].

1.1.13 PrimeFaces

Es una biblioteca de código abierto integrada por componentes enriquecidos para JSF que proporcionan una mayor funcionalidad que la otorgada por los componentes estándar de JSF, y adicionalmente cuenta con temas integrados que hacen uso de jQuery UI como base para permitir al desarrollador la creación o adaptación de temas sin mayor dificultad [19].
Entre sus características principales se encuentran:

- Cuenta con un abundante conjunto de componentes (*HtmlEditor*, autocompletado y gráficos, por mencionar algunos).
- Incorpora AJAX basado en el estándar de las APIs JSF AJAX.
- Su configuración es nula y no necesita de dependencias externas, el único requisito es incorporar la biblioteca de PrimeFaces a una aplicación de JSF estándar.
- Soporte por medio del marco de trabajo *Atmosphere* al paradigma *push*.
- Cuenta con un conjunto de herramientas de interfaz de usuario para el desarrollo de aplicaciones de dispositivos móviles.
- Dispone de numerosos temas incorporados por medio del marco de trabajo *Skinning*.
- Incluye soporte para herramientas de diseño de temas visuales.
- Posee una amplia documentación [20].

1.1.14 PHP

PHP es un lenguaje interpretado del lado del servidor para la creación de contenido HTML, es ejecutable en los principales sistemas operativos Unix, plataformas Windows y MacOS X, que se usa con los servidores Web líderes en el mercado como lo son Apache, Microsoft IIS y Oracle iPlanet, por mencionar algunos, además de que dispone de un amplio soporte para bases de datos (incluyendo MySQL, PostgreSQL, Oracle, Sybase y bases de datos compatibles con ODBC, entre otras).

Se utiliza principalmente de tres formas:

1. **Como *script* del lado del servidor:** En la creación de contenido Web dinámico.
2. **Como *script* de línea de comando:** Para la ejecución de *scripts* de línea de comando con la finalidad de realizar la administración del sistema, como lo son realizar copias de seguridad o análisis de registros, entre otras tareas.
3. **Para generar aplicaciones de GUI del lado del cliente:** Mediante el uso de PHP-GTK le es posible escribir aplicaciones multiplataforma GUI [21].

Cabe mencionar que entre los lenguajes de programación del lado del servidor, PHP se encuentra consolidado con un 82.9% de uso entre los sitios Web disponibles en la red [22].

1.1.15 JSON

JSON es un formato ligero de intercambio de datos, cuya característica es que para los humanos es simple leerlo y escribirlo, a la par que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, pero es un formato de texto que es completamente independiente del lenguaje, aunque utiliza convenciones que son ampliamente conocidas por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, entre otros, por estas propiedades JSON es un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras, la primera como colección de pares de nombre/valor que varios lenguajes definen como un objeto, registro, estructura, diccionario, tabla *hash*, lista de claves o un arreglo asociativo, y la segunda estructura como una lista ordenada de valores, que en la mayoría de los lenguajes se implementa como un arreglo, vector, lista o secuencia, la representación gráfica de estas estructuras se encuentra representada en la Figura 1.4.

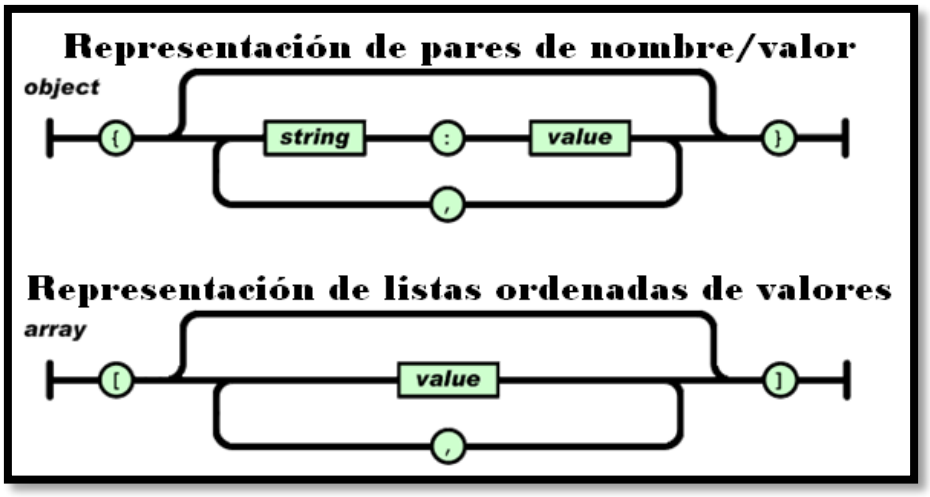

Figura 1.4 Representación gráfica de las estructuras que constituyen JSON

Estas estructuras son consideradas como universales ya que virtualmente todos los lenguajes de programación las soportan de una forma u otra, por lo cual es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se encuentre basado en esas estructuras [23].

1.1.16 jQuery

Es un marco de trabajo de JavaScript de código abierto del lado del cliente, dirigido a la interacción existente entre el DOM, JavaScript, AJAX y HTML, con el objetivo de simplificar los comandos comúnmente encontrados en JavaScript, obedeciendo su principio fundamental de que, mediante una escritura reducida, es posible realizar más trabajo en menos tiempo y código.

Además de su enfoque hacia un desarrollo intuitivo y conciso, jQuery también se enfoca en el conjunto de elementos para la gestión del DOM, permitiendo un acceso fácil y claro a ellos, dispone un conjunto común de funciones a través de las cuales se asegura la compatibilidad a través de todos los visualizadores Web, proporciona cuantiosos *plug-ins* dedicados a tareas concretas (mostrar un calendario, validaciones, exportación de tablas, por mencionar algunos), y se encuentra sustentado por una amplia comunidad de desarrolladores [24].

1.1.17 Traductor

De acuerdo a Aho [25], se define un traductor como un programa con la capacidad de leer un programa en un lenguaje, nombrado lenguaje de origen, y traducirlo en un programa equivalente en otro lenguaje, denominado lenguaje objetivo, la representación gráfica de esta definición se encuentra en la Figura 1.5.

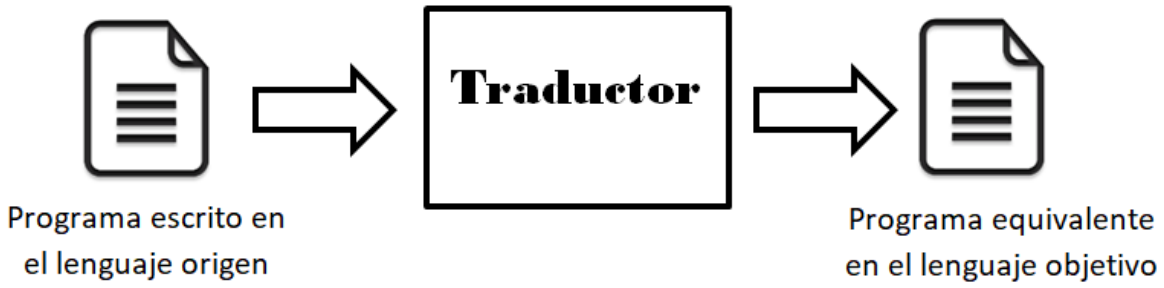

Figura 1.5 Descripción gráfica de traductor

De acuerdo al nivel del lenguaje de entrada que recibe el traductor se encuentran los conceptos de compilador y transpilador, el primero toma como entrada un lenguaje de alto nivel y realiza su traducción a lenguaje de bajo nivel, mientras que un transpilador, también nombrado traductor de fuente a fuente, es un traductor que tanto en la parte de entrada como de salida toma lenguajes de alto nivel, como se aprecia en la Figura 1.6.

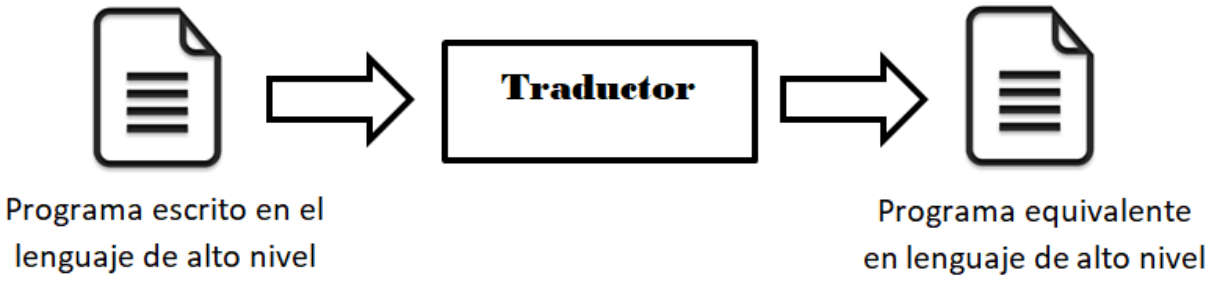

Figura 1.6 Descripción gráfica de transpilador

Como se muestra en la Figura 1.7, el proceso de traducción consta de varias fases divisibles en dos etapas principales, la primera de ellas corresponde al procesamiento de la fuente en donde se realiza la lectura del lenguaje origen, la obtención de sus componentes léxicos, la agrupación

de dichos componentes léxicos en elementos gramaticales y la realización de la revisión semántica, mientras que la segunda etapa corresponde a la generación del código objetivo equivalente al código fuente, para lo cual se genera una representación intermedia del programa fuente, en algunos casos se lleva a cabo una optimización del código intermedio y por último la generación del código objetivo.

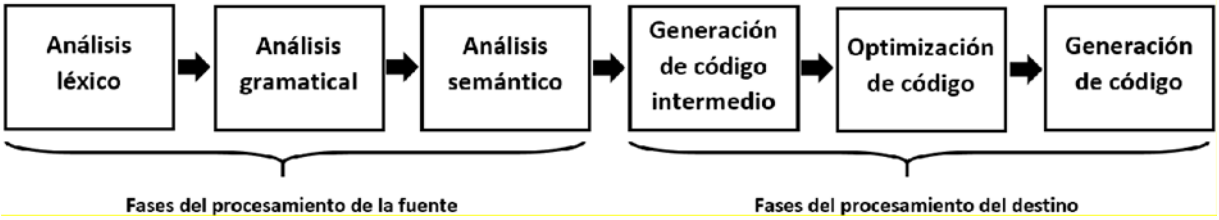

Figura 1.7 Fases del proceso de traducción

1.1.18 ANTLR

ANTLR es una herramienta empleada para la lectura, procesamiento, ejecución o traducción de archivos de textos estructurados o binarios, el cual se usa para grandes proyectos que involucran el análisis de más de dos billones de consultas (Twitter) o hasta proyectos a pequeña escala como lectores de archivos de configuración, el programa obtenido derivado del uso de ANTLR recibe el nombre de aplicación de lenguaje.

El funcionamiento de ANTLR consiste en que a partir de una gramática (descripción formal de un lenguaje) genera un analizador sintáctico para dicha gramática y le permite construir árboles de análisis sintáctico, que son la representación de las coincidencias entre la gramática y los datos de entrada, así como también le es posible la generación automática de *walkers*, que son estructuras usadas para visitar los nodos de los árboles sintácticos para la ejecución de códigos específicos de la aplicación.

Cuenta con la capacidad de aceptar cualquier gramática dada (exceptuando el caso de recursividad izquierda de forma indirecta) sin tener conflictos gramaticales o de ambigüedad. Su funcionamiento consiste en el uso de la tecnología de análisis *Adaptive LL(*)* o también

conocida como *All (*)*, la cual realiza el análisis de la gramática dinámicamente en tiempo de ejecución en lugar de hacerlo de forma estática previo a la ejecución del analizador sintáctico generado, y al tener acceso a las sentencias de entrada, le es posible reconocer de forma apropiada las sentencias a través de la gramática [26].

La Figura 1.8 muestra las fases de las aplicaciones de lenguaje, en donde se observa como un traductor lee una entrada y emite una salida, combinando esencialmente los elementos de un lector y un generador [27].

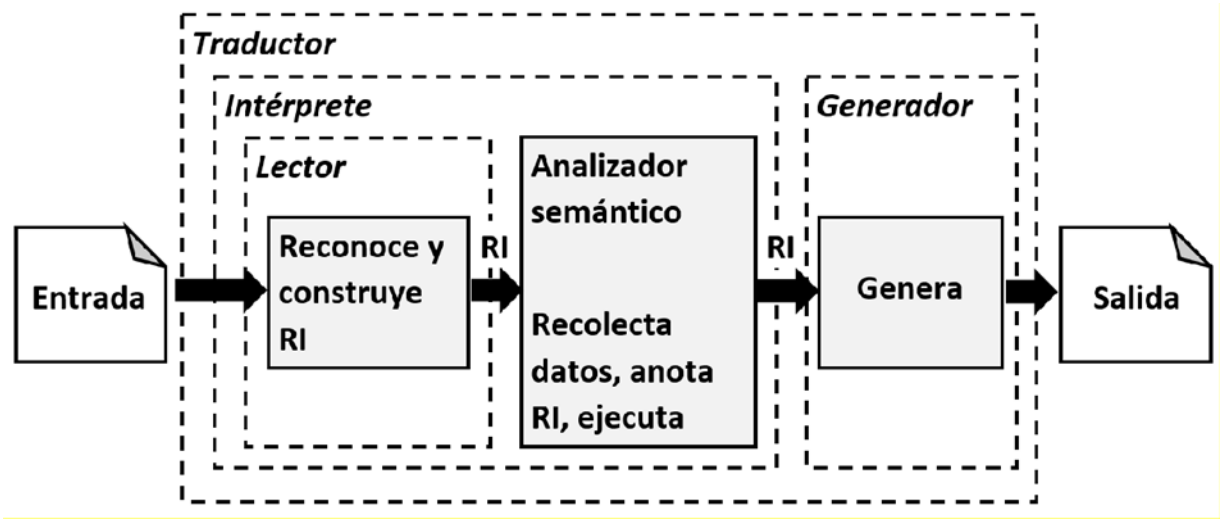

Figura 1.8 Fases de las aplicaciones de lenguajes

1.1.19 WebRatio Web Platform

WebRatio Web Platform es un entorno de desarrollo que proporciona el uso del estándar IFML para modelar el flujo de interacción entre el usuario y la aplicación Web a desarrollar (correspondiente al *front-end*), así como también incorpora elementos para el modelado de dominio (*back-end*).

Algunos de los aspectos en que se centra la herramienta y que son relevantes en este proyecto:

- **Modelado del dominio:** Soporta el diseño del modelo de dominio haciendo uso de las características de los diagramas de clases de UML.

- **Diseño del *front-end*:** Permite el diseño el modelo navegacional con una implementación particular de los elementos especificados por IFML [17].

1.1.20 Layout

La distribución de contenido (*layout*) de una página Web es la forma en que el texto y otros elementos se ajustan a las páginas de un documento [28]. Las aplicaciones Web cuentan con un diseño común que proporciona al usuario una experiencia consistente a medida que se navega de página en página, dentro de este diseño se encuentran elementos comunes en cualquier interfaz de usuario, como lo son el encabezado de la aplicación, el menú de navegación y pie de página.

Las estructuras HTML comunes, como scripts y hojas de estilo, utilizadas frecuentemente dentro de varias páginas pertenecientes a una aplicación Web pueden ser definidas en un archivo de distribución para posteriormente ser referenciado por cualquier vista dentro de la aplicación ayudando a la reducción de código duplicado [29].

1.1.21 Estilos

Los estilos son elementos que homogeneizan la apariencia de las páginas Web definiendo clases de estilo personalizadas o redefiniendo etiquetas HTML, esto por medio del uso de hojas de estilo en cascada [30]. Las hojas de estilo en cascada son elementos indispensables en el diseño Web. Entre sus cualidades más importantes se encuentra la reducción del peso global de una página Web, su independencia con respecto al código, y su flexibilidad para aportar estilos para diferentes medios, como pantallas de computadoras de escritorio o dispositivos móviles, por mencionar algunos [31].

1.2 Planteamiento del problema

Las capacidades de cómputo en el cliente son muy variadas y van en constante aumento gracias a los avances tecnológicos y a la masificación del uso de dispositivos como computadoras de escritorio, laptops o dispositivos móviles como *tablets* o teléfonos inteligentes, por lo cual se genera que los usuarios sean cada vez más exigentes en lo que se refiere a sus expectativas sobre

atractivo y respuesta de las aplicaciones. De lo anterior deriva la existencia de las RIAs, que combinan lo mejor de las aplicaciones Web tradicionales con lo mejor de las aplicaciones autónomas como el software de ofimática instalado en computadoras de escritorio.

Como consecuencia de las características en funcionalidad, rendimiento, velocidad, atractivo y capacidad que se exige a las RIAs, aunado a las complejidades inherentes de las aplicaciones distribuidas, este tipo de aplicaciones está obligado a considerar gran cantidad de detalles sobre elementos que se ejecutan en ambientes distintos tanto en tecnología como en capacidad, haciendo indispensable el uso de técnicas de Ingeniería de Software para asegurar que las aplicaciones cumplen con las expectativas del usuario.

En este trabajo se propone desarrollar un generador de Aplicaciones Enriquecidas de Internet con la capacidad de obtener los esqueletos de la aplicación y que considere más de un lenguaje de salida, relación entre modelos y manejo de temas, entre otros.

1.3 Objetivo general y específicos

Este apartado contiene el objetivo general y el conjunto de objetivos específicos a los cuales atiende el presente proyecto de tesis.

1.3.1 Objetivo general

Desarrollar un generador de Aplicaciones Enriquecidas de Internet a partir de modelos realizados con UML e IFML basados en un patrón arquitectónico MVC y entregando como salida una de dos posibles combinaciones: a) JSF + PrimeFaces y b) PHP + jQuery.

1.3.2 Objetivos específicos

1. Analizar las características de IFML para identificar los elementos a implementar en el generador.
2. Analizar las herramientas que soportan IFML para elegir la que mejor se adapte a la generación de código final.

3. Analizar diagramas y herramientas de modelación UML para elegir la forma de relacionarlos con el modelo en IFML y la generación de código final esperada.
4. Plantear la arquitectura, independiente del lenguaje, de las Aplicaciones Enriquecidas de Internet a soportar, basándose en el patrón arquitectónico MVC.
5. Realizar la(s) gramática(s) para los intérpretes a) IFML + UML –> JSF + PrimeFaces y b) IFML + UML –> PHP + jQuery
6. Realizar el generador de Aplicaciones Enriquecidas de Internet a partir de las gramáticas realizadas.
7. Realizar los casos de estudio (al menos uno por cada combinación).

1.4 Justificación

Actualmente existen cuantiosos lenguajes de programación y marcos de trabajo, tanto para el lado del cliente como para el lado del servidor, cada uno con ventajas y desventajas; si bien muchos cuentan con ambientes específicos de desarrollo, de uso libre o con licencia, es muy poco común que tengan una relación directa con estándares de modelado como IFML.

Entre los lenguajes de programación más utilizados en aplicaciones Web y/o enriquecidas se encuentran, del lado del servidor PHP y JSF, mientras que del lado del cliente se considera a jQuery como el estándar de facto y a PrimeFaces como el mejor marco de trabajo para utilizar con JSF. Tanto jQuery como PrimeFaces utilizan el concepto de temas para el manejo de estilos y colores, además de ser susceptibles a implementar diversas distribuciones entendibles como plantillas; ese concepto, a primera vista, no está soportado en herramientas como WebRatio.

De lo anterior se deriva la necesidad de contar con un generador de Aplicaciones Enriquecidas de Internet que considere los elementos mencionados con anterioridad en el planteamiento del problema. El generador de aplicaciones obtendrá los esqueletos de la aplicación enriquecida: vistas (en HTML o equivalente), control (*Beans* Administrados en el caso de JSF, archivos PHP que devuelven JSON en el otro caso) y modelo (clases en Java o PHP según corresponda); se consideran esqueletos porque el modelo no incluirá el código específico SQL para la base de datos, aunque el resto de los elementos sí estará completo.

Capítulo 2. Estado de la práctica

En esta sección se presenta el análisis del estado de la práctica, es decir, de trabajos que se relacionan con la investigación a desarrollar en este proyecto de tesis.

2.1 Trabajos relacionados

En [32] se expuso la problemática en empresas dedicadas a la producción de aplicaciones móviles, en donde es necesario que los tiempos de desarrollo sean breves aunque no se cuente con suficiente personal con las habilidades necesarias. La solución propuesta es la *"suite"* de herramientas llamada WebRatio Mobile Platform para proporcionar un desarrollo impulsado por modelos para aplicaciones móviles, basado en las especificaciones proporcionadas por el desarrollador; la herramienta, WebRatio, proporciona la comprobación de modelos y la generación de código completo. El código generado consiste en aplicaciones móviles multiplataforma listas para desplegar, basadas en marcos de trabajo de código abierto *Apache Cordova*, dentro de la distribución *PhoneGap*.

En [33] se mencionó que en la última década, los sistemas se equiparon con interfaces gráficas de usuario sofisticadas, en donde las potentes funcionalidades de interacción implementadas en la parte superior de una variedad de tecnologías y plataformas, presentan límites que se han vuelto menos distinguibles entre aplicaciones cliente-servidor, aplicaciones Web, RIAs y aplicaciones móviles, por consiguiente, las investigaciones en desarrollo de software se centran en modelos abstractos de las interfaces de usuario. La propuesta de [33] se centró en RIAs, debido a que son aplicaciones complejas, y su desarrollo requiere diseño e implementación, que consumen tiempo y las herramientas disponibles están especializadas en el diseño manual. Como resultado, se presenta un enfoque que deriva GUIs de RIAs mediante un Desarrollo Dirigido por Modelos con la combinación de la ontología OWL2 e IFML para generar automáticamente la interfaz de usuario en una plataforma específica de acuerdo con las especificaciones de los modelos.

La problemática señalada por [34] es que la navegación en páginas Web, incluso las que cuentan con un diseño responsivo, en equipos móviles resulta incómoda, ya que la navegación móvil y

la Web difieren en aspectos como el tamaño de la pantalla, diferentes contextos físicos y modales, entre otros. Por lo cual es más común que páginas Web existentes (por ejemplo, Facebook, Amazon, EBay, YouTube, por mencionar algunas) introduzcan a una aplicación móvil dedicada. Para solventar esta problemática en [34] presentaron un enfoque capaz de construir modelos a partir de los patrones de uso de un sitio Web provenientes de sesiones de su uso en la Web y en aplicaciones para dispositivos móviles, que se transforma en un árbol de navegación y del cual se genera un modelo de navegación orientado a dispositivos móviles.

El trabajo de [35] reconoció que se han realizado diversos trabajos para la descripción y generación de GUIs integrando ontologías, así como también surgieron enfoques definidos por el OMG (ODM, MDA, IFML) para la realización de dicha tarea. Con el propósito de aprovechar las ventajas de los modelos de interacción junto con ontologías formales, en [35] realizaron la propuesta de un enfoque MDE para aplicaciones Web que combina ontologías OWL2, para inducir los elementos de la interfaz y sus características, e IFML, para capturar las interacciones y acciones relacionadas con los conceptos definidos en el modelo lógico de la GUI, que produce de forma automática un metamodelo en HTML5.

En [36] se destacó que el diseño del *front-end* para aplicaciones móviles sigue siendo una tarea manual para los desarrolladores, por lo que se corren altos riesgos de errores, incoherencias e ineficiencias durante su proceso de desarrollo, por esta razón plantean como solución un enfoque basado en modelos bajo el estándar IFML de la OMG junto con una extensión a ese lenguaje de modelado adaptada a aplicaciones móviles. Como resultado del trabajo en [36], se obtuvo un generador de código automático para aplicaciones móviles que parte de dos posibles niveles, el primero, comenzando desde el modelo de dominio (diagrama Entidad-Relación que describe el modelo de datos de la aplicación) o el segundo a partir de un modelo IFML. De esa forma, el generador analiza el modelo de entrada de la aplicación, lo serializa como un archivo XML que contiene la información sobre los datos de la aplicación y la interacción, y finalmente, produce los archivos JavaScript, CSS y HTML de una aplicación multiplataforma.

Como parte del estudio del proyecto de investigación de *Automobile European*, para encontrar la mejor estrategia de generación de código de aplicaciones para dispositivos móviles para la empresa de desarrollo de software WebRatio, con el objetivo de maximizar sus posibilidades de tener éxito al diseñar y desarrollar una herramienta MDD dirigida al mercado de aplicaciones para dispositivos móviles, en [37] se presentó un estudio comparativo realizado para identificar las mejores ventajas y desventajas entre varios enfoques de generación de código automático. En el escrito se incluyeron generadores de código para plataformas nativas (Android e iOS) y generadores para marcos de trabajo multiplataforma como *PhoneGap* y *AppCelerator Titanium*, se empleó MDA como marco de referencia para las estrategias de generación de código: a) PIM a NC; b) PIM a PSM a NC; c) PSM a NC; d) PIM a CPC; e) PIM a FSM a CPC, y clasificaron el esfuerzo necesario para cada enfoque de generación de código presentado. Además, el estudio de los distintos enfoques de generación de código analizados por [37] concluyó en que uno no es mejor que otro en términos absolutos, sin embargo, para el caso específico de la empresa WebRatio, se realizó la recomendación de aplicar el enfoque PIM a CPC, sirviéndose *PhoneGap* como marco de desarrollo multiplataforma, por dar prioridad a la multiplataforma por sobre el alto rendimiento, la mayor parte de sus clientes son usuarios empresariales que privilegian una interfaz de usuario práctica y utilizable, y su equipo de desarrollo cuenta con habilidades y experiencia en el modelado de aplicaciones y tecnologías Web estándar. Y cabe mencionar que el producto WebRatio Mobile Platform de la empresa WebRatio implementa la estrategia de generación de código recomendada por [37].

De acuerdo con el crecimiento y añadidura de características del *back-end* o *front-end* de un sistema y posibles errores en alguna de sus fases de desarrollo anteriores, es necesario preparar pruebas para verificar el funcionamiento de la aplicación según sus especificaciones. En [38] se señaló que dichas pruebas requieren de tiempo para su preparación y ejecución, y al ser una tarea realizada de forma manual, no se garantiza que su precisión no se vea afectada cada vez que se agrega una nueva característica a la aplicación o un problema se corrige y cambia el código base de la aplicación. Ya que IFML es una notación creada para describir el contenido, la interacción del usuario y el comportamiento de control en el *front-end* de una aplicación, para

[38] fue posible realizar el esbozo de un generador automatizado de casos de prueba para el *front-end* de aplicaciones Web, combinando un enfoque basado en modelos e IFML.

Debido a la amplia gama de pantallas de dispositivos y plataformas de codificación en el desarrollo de aplicaciones móviles, es una parte crítica el tener la capacidad de evolucionar y evaluar nuevas versiones de una aplicación. En atención a estas necesidades en [39] presentaron la herramienta en línea para el desarrollo rápido de prototipos de aplicaciones Web y de dispositivos móviles nombrada IFMLEdit.org, la cual permite generar prototipos realistas para ser fácilmente convertidos en aplicaciones implementadas por medio de un enfoque MDD que sigue el estándar IFML de la OMG. La herramienta permite: a) Edición de modelos IFML; b) Editor de propiedades vinculadas a la base de datos; c) Generación de código totalmente funcional para la arquitectura Web y móvil, con una emulación en un visualizador Web para permitir probar el prototipo de la aplicación Web o móvil sin instalar ningún servidor Web y también en ausencia de conexión a Internet; d) Personalización de la base de datos de *Mock-up*; e) La descarga del prototipo para su refinamiento y producción de la aplicación final.

Se señala por [40], que en el desarrollo de aplicaciones Web o móviles es común que los desarrolladores deban realizar implementaciones de operaciones CRUD, en el mercado se encuentran disponibles diversos marcos de trabajo que cuentan con elementos para optimizar operaciones CRUD (por ejemplo: Ruby on Rails, Django, Catalyst, entre otros), mientras que en el dominio del enfoque MDWE no se encuentra una herramienta impulsada por modelos que proporcione de forma automática una implementación de operaciones CRUD, en vista de tal carencia y aunado a la demanda de un alto nivel de productividad en los desarrollos de proyectos de *software*, en [40] desarrollaron AutoCRUD, un complemento creado para la herramienta WebRatio con la finalidad de posibilitar la especificación IFML automatizada de operaciones CRUD brindando optimización en operaciones repetitivas y recurrentes. Para enfatizar la importancia de la optimización del esfuerzo de desarrollo en un proyecto de *software* llevaron a cabo una colaboración con la empresa Homeria Open Solutions, S.L , socio oficial de WebRatio, realizaron una evaluación para analizar el ahorro de costos obtenido incorporando la herramienta AutoCRUD en el proceso de desarrollo de proyectos reales implementados por la

empresa, y señalaron importantes evidencias de la ganancia de optimización obtenida por la herramienta, pero también de su escalabilidad, ya que los resultados son incluso mejores en medida de cuanto más grande es el proyecto.

Pese a la existencia de marcos de trabajo y herramientas para simplificar el proceso de creación de RIAs (por ejemplo Ajax o JavaFX, entre otros.), el desarrollo de este tipo de aplicaciones demanda un alto consumo en tiempo dado que las herramientas disponibles se orientan más hacia el diseño manual, por esta razón, en [41] presentaron un enfoque dirigido por componentes para la generación de RIAs empleando el estándar IFML a través de un modelo simplificado, respetando IFML, se realiza la transformación al metamodelo MVC para RIAs con el cual el generador de código produce los gráficos de RIAs teniendo en cuenta los eventos y las interacciones del usuario además de que respeta el patrón MVP.

En la MDA existen dos tipos de transformaciones, las transformaciones que van de CIM a PIM y las trasformaciones de PIM a PSM. La transformación de modelos en el nivel CIM a modelos en el nivel PIM rara vez se utilizan en trabajos de investigación por tratarse de dos niveles diferentes, y los que se encuentran disponibles emplean BPMN para crear el nivel CIM, por lo cual, en [42] realizaron la propuesta de una metodología para una transformación semiautomática de CIM a PIM. El punto principal del trabajo es que establecieron un conjunto de reglas bien definidas que estructuran el CIM a fin de facilitar la transformación hacia el PIM, dichas reglas son también las que permiten la transformación de los modelos PIM resultantes a SoaML e IFML, lo cual tiene como propósito una eventual transformación a modelos PSM.

En el desarrollo de sistemas Web, las tareas de diseño para el *front-end* suelen ser realizadas mediante enfoques MDWE, en el cual para llegar a la generación de código fuente completo, es necesaria la representación de modelos decorados manualmente con una semántica que represente las acciones de los usuarios, los flujos de la pantalla y la lógica de negocio. Como se menciona con anterioridad, los modelos de transformación admiten modelos altamente detallados para generar piezas funcionales de una aplicación, las cuales después de recibir un refinamiento, por parte de los diseñadores, proporcionan un prototipo ejecutable final que es

evaluado por los usuarios. El proceso de generación del prototipo descrito requiere de meses de trabajo, por lo que el esfuerzo invertido para detallar modelos es visto como un motivo para evitar la adopción de algunos enfoques MDWE, para dar solución a la problemática de la incapacidad de ejecutar tareas en un corto periodo de tiempo al seguir un enfoque MDWE, en [43] desarrollaron una metodología orientada a MDWE, nombrada *MockupToME Method*, respaldada por técnicas de diseño automatizado estrictamente asociadas con patrones de casos de uso del tipo CRUD, con la cual logran ejecutar tareas en cortas escalas de tiempo, aunque se encuentra limitada a patrones de casos de uso del tipo CRUD, Lista, Filtrado y Reporte.

Diversos trabajos plantean el uso de técnicas impulsadas por modelos para la especificación de aplicaciones de *software*, interfaces e interacción con el usuario, sin embargo, no son abordadas específicamente las necesidades de RIAs en términos de aspectos gráficos y de conexión con las capas de aplicación respetando un patrón de diseño dado como MVC o MVP, por mencionar algunos. En [44] establecieron un enfoque basado en modelos centrados en el aspecto gráfico de la aplicación y en la abstracción de cualquier conocimiento técnico de la plataforma dentro del modelo, se presentó una extensión al estándar IFML, para el MDD enfocado a RIAs, simplificándolo y abstrayendo detalles técnicos, añaden componentes IFML que describen de manera eficiente los componentes gráficos de un aplicación enriquecida, describen un metamodelo PSM enfocado a RIAs adoptando el MVP como patrón arquitectónico central y propusieron un proceso de desarrollo impulsado por el modelo para generar de forma automática una RIA completamente operativa a partir de un modelo simple.

En [45] señalaron que la tendencia para las herramientas multiplataforma es la de permitir la creación y distribución de aplicaciones para múltiples destinos, además de entornos sin código o con bajo código con IDEs visuales disponibles en línea. Por lo cual presentan un enfoque MDD para la generación automática de aplicaciones Web multiplataforma a partir de especificaciones IFML, en dicho trabajo, la semántica de IFML se define por la asignación de diagramas IFML a *Place Chart Nets* (PCN), una variante de las redes de Petri caracterizado por primitivas de modularización compacta (en este trabajo es utilizada para la simulación de modelos). Desde el PCN y la especificación de la plataforma (Web o dispositivo móvil) se hace

entrega de un prototipo para Web y dispositivos móviles completamente funcional a través de la transformación de modelo a código.

En [46] mencionaron que disminuir el tiempo de desarrollo de un aplicación, mientras se preserva o aumenta la calidad del código fuente es una tarea difícil de lograr, pero con la aplicación de un enfoque MDD en combinación con la generación automática de código es posible lograr una disminución en medida de hora-mercado. La propuesta en [46] es el diseño de un sistema que transforma la especificación del cliente en una especificación de controles GUI mediante la ejecución de cuatro tareas: 1) Generación de clases del Modelo SFR; 2) Generación de clases SFR UI; 3) Generación de prueba SFR para las clases del modelo, y 4) Generación de clases UI de prueba SFR. Los autores realizaron cuantiosas pruebas para verificar la estabilidad y confiabilidad del sistema, verificando si el número de controles creados coincidía con el número de campos SFR que fueron obtenidos con JAXB del modelo de datos XML.

Actualmente, se cuenta con enfoques importantes (PEGs, GLR, LR, LL) que son empleados con éxito para analizar lenguajes de programación, así como también se dispone de avanzados generadores de *parsers* que se encuentran equipados con herramientas para el procesamiento de estructuras de datos resultantes de su reconocimiento, como por ejemplo Rascal y Stratego, por mencionar algunos, y ambos ofrecen un conjunto de herramientas para la transformación de programas, encontrándose la importancia de dichas transformaciones en estar destinadas a mejorar la confiabilidad, la productividad y el análisis del software. Las herramientas mencionadas con anterioridad tienen un enfoque directo al generar un AST desde un programa, dicho enfoque es que las reglas de producción gramatical se anotan manualmente por el usuario con la información sobre la construcción del árbol. Por ello, en [47] se propuso un enfoque en el cual los usuarios nunca deben escribir reglas o estrategias que requieran aprender nuevos lenguajes mediante un algoritmo que, dada la gramática de un lenguaje, algunos programas y sus correspondientes AST, es capaz de inferir automáticamente las anotaciones y generar la gramática anotada.

Existen diversos marcos de trabajo que permiten la implementación de formateadores de código para cualquier lenguaje específico de forma simple, pero la construcción de dichos formateadores resulta en una tarea complicada por el hecho de que cada experto da formato al código de forma distinta, lo cual resulta en innumerables variantes de formato o en opciones de configuración, aunado la problemática de que el tamaño de cada implementación va a ser distinto, puesto que el tamaño de la implementación varía con el tamaño de la gramática, siendo que a mayor número de reglas establecidas, mayor es el tamaño de la implementación. Para la resolución de las dificultades expuestas en la construcción de formateadores, en [48] se desarrolló un enfoque que automáticamente deriva formateadores para cualquier lenguaje dado sin la intervención de un experto en dicho lenguaje y un formateador de código nombrado CodeBuff, que sigue el enfoque antes mencionado, y cuyas pruebas de precisión y gramática son realizadas empleando Java, SQL y gramáticas ANTLR.

En [49] se señaló que en el área de desarrollo Web no se cuenta con un IDE que provea herramientas de desarrollo, depuración, refactorización y pruebas enfocadas en el lenguaje interpretado JavaScript, lo cual obstaculiza el seguimiento de enfoques como el del Desarrollo Impulsado por Pruebas, por esta razón se realiza el desarrollo de la herramienta PharoJS, la cual es un IDE enfocado en la creación y ejecución de pruebas en JavaScript, y en el momento en que se presentan errores en la prueba se permite al desarrollador realizar de forma simple y fluida las modificaciones pertinentes para posteriormente proseguir con la ejecución sin mayor complicación, cabe mencionar que las pruebas realizadas son desarrolladas una única vez y son ejecutadas bajo diferentes condiciones, y sólo la configuración y los recursos son modificados. PharoJS sigue tres pasos en su proceso de desarrollo: 1) Pruebas nativas en Pharo, para probar el procesamiento y los algoritmos; 2) Aplicación con un visualizador Web remoto, en donde se introducen las características requeridas por el visualizador Web, y 3) Traducción a JavaScript, en donde se asegura el funcionamiento de la aplicación en diversos visualizadores Web, y los códigos de ejecución dentro del visualizador Web y de Smalltalk son traducidos de forma precisa a JavaScript.

Debido al auge de las aplicaciones para dispositivos móviles y al éxito de HTML5 como estándar avalado por el W3C al ofrecer capacidades multimedia de forma nativa a visualizadores Web sin necesidad de integrar herramientas externas, el uso y soporte de Adobe Flash fue en decadencia, resultando en páginas Web con animaciones Flash que no reciben mantenimiento, y dado que la creación de nuevos elementos desde cero para reemplazar los elementos obsoletos Flash representa una inversión de esfuerzo y tiempo, en [50] se propuso un enfoque de transformación para facilitar el proceso de migración de archivos SWF a JavaScript/HTML5 mediante los siguientes pasos: a) Realiza la de compilación de s*prites*, imágenes y botones a formato PNG, crea matrices de transformación para cada *assets*, entre otras transformaciones, b) Construye el proyecto por medio de cuatro pasos internos: 1) Crea una clase de envoltura AS3 para cada *asset* e incorpora los elementos a ella mediante meta etiquetas integradas en AS3; 2) Es agregada la función setStage en la clase principal donde los *assets* son agregados a la lista de visualización; 3) Las funciones que no se encuentran implementadas por la API de transpilación Flash son eliminadas o sustituidas según el caso, y 4) Se realiza la implementación del envoltorio AS3 de cualquier biblioteca JavaScript requerida, y c) El proyecto obtenido AS3 obtenido en la fase de construcción se traduce a JavaScript.

2.2 Análisis comparativo

En la Tabla 2.1 se reúne un conjunto de siete artículos los cuales manifiestan el auge del modelado dirigido por modelos y que resaltan la necesidad de contar con herramientas que sigan dicho enfoque [27][32][37]-[40], así como también dos artículos centrados en el tema de la obtención de gramáticas [42][43], el cual también es un punto de interés dentro del presente trabajo.

Tabla 2.1 Análisis comparativo de artículos relacionados.

Artículo	Objetivo	Tecnologías	Resultados	Estado
[32]	Proponer la *suite* de herramientas *WebRatio Mobile Platform* para la especificación del modelo de dominio y del modelo de interacción de aplicaciones móviles siguiendo el desarrollo impulsado por modelos.	IFML WebRatio	Descripción de la experiencia en la extensión de IFML para el dominio móvil presentando la herramienta *WebRatio Mobile Platform* para generar aplicaciones móviles multiplataforma completamente funcionales.	Finalizado
[37]	Realizar un estudio comparativo para encontrar la mejor estrategia de generación de código móvil para el desarrollo de una aplicación MDD por la empresa de	UML IFML PhoneGap AppCelerator Titanium Eclipse Acceleo XML Java	Recomendación de la estrategia de generación de código PIM a CPC, utilizando PhoneGap como marco de desarrollo multiplataforma, para el desarrollo de la herramienta MDD *WebRatio Mobile Platform* de la empresa WebRatio.	Finalizado

Artículo	Objetivo	Tecnologías	Resultados	Estado
	desarrollo de software WebRatio.			
[42]	Proponer un enfoque de transformación del modelo para el control de la transformación de CIM a PIM basada en la Web de acuerdo con la MDA.	IFML SoaML UML 2	Establecimiento de un conjunto de reglas bien definidas para transformar de CIM a PIM a fin de facilitar la transformación posterior hacia PSM.	En progreso
[43]	Proponer una metodología para MDWE que mediante técnicas de diseño automatizadas ayuden a la producción de modelos mediante tareas de transformación y refinamiento y que permita el alto nivel de abstracción que se encuentra en los modelos de aplicación basados en MVC.	Metodología MockupToME Method MockupToME MockupToME DSL UML 2 WebDSL	El método MockupToME que acelera la especificación de modelos detallados basados en MVC dentro de iteraciones planeadas para durar un corto periodo de tiempo. Resumen de un informe de dos sistemas, uno desarrollado con la propuesta de diseño automatizado y el otro utilizando principalmente diseño manual.	Finalizado
[44]	Presentar un enfoque MDD para la generación automática	IFML	Editor en línea IFMLEdit.org basado en un enfoque MDD que a partir de	Finalizado

Artículo	Objetivo	Tecnologías	Resultados	Estado
	de aplicaciones Web multiplataforma a partir de especificaciones IFML.	Eclipse Modeling Framework QTV - Operational Eclipse Acceleo XML	especificaciones en IFML realiza la generación automática de prototipos funcionales multiplataforma para Web y dispositivos móviles.	
[45]	Presentar una herramienta MDD que permita la creación y distribución de aplicaciones móviles multiplataforma y Web.	IFML Place Chart Nets JSON Apache Cordova	Construcción de la versión preliminar de la herramienta IFMLEditing.org que soporta la transformación del mapeo semántico de IFML a PCN.	En progreso
[47]	Desarrollar ASTs de forma que las reglas de construcción AST sean inferidas a partir de una gramática libre de contexto de lenguajes de programación y un conjunto de pares: P y PAST, sea capaz de inferir automáticamente las anotaciones necesarias y	ANTLR	Desarrollaron un algoritmo capaz de inferir las reglas de construcción de un árbol sintáctico abstracto basadas en una Gramática Libre de Contexto, y un conjunto de pares: P y PAST. Realización de la implementación del algoritmo creado mediante una herramienta que genera la gramática	Finalizado

Artículo	Objetivo	Tecnologías	Resultados	Estado
	generar la gramática con anotaciones.		con anotaciones y el analizador correspondiente usando ANTLR.	
[48]	Desarrollar un enfoque que automáticamente derive formateadores para cualquier lenguaje dado sin la intervención de un experto en dicho lenguaje.	ANTLR Java SQL	El formateador de código CODEBUFF, el cual emplea aprendizaje automático para abstraer las reglas de formato a partir de un *"corpus"* representativo.	Finalizado

En la Tabla 2.2 se agrupan artículos referentes a la obtención de salidas productivas a partir de modelos, cabe mencionar que todos los trabajos citados en la tabla se encuentran finalizados.

Tabla 2.2 Comparativa de artículos relacionados sobre traductores de código que reciben como entrada algún tipo de modelo.

Artículo	Objetivo	Entrada	Salida	Resultados
[33]	Proponer un enfoque que derive la interfaz de usuario de RIAs a plataformas específicas a partir de la combinación de la ontología OWL2 e IFML.	Modelo IFML Modelo lógico que respeta la sintaxis ODM	Metamodelo Flex	Enfoque que a partir de los modelos IFML y de una ontología de GUI realiza una transformación QVT a un metamodelo Flex y mediante una transformación M2T con Acceleo es obtenida la interfaz de usuario.

Artículo	Objetivo	Entrada	Salida	Resultados
[34]	Su objetivo es desarrollar un enfoque capaz de construir un modelo a partir de los patrones de uso de un sitio Web procedentes de sesiones de uso Web y específicas de aplicaciones móviles.	Grafo navegacional extraído a partir de un reporte de *Web Analytics*	Modelo navegacional IFML	Enfoque en el cual a partir de minar el uso de una página Web se extrae un árbol navegacional que es transformado a un modelo en IFML.
[35]	Proponer un enfoque que a través de la combinación de ODM junto con IFML genere interfaces de usuario de aplicaciones Web.	Metamodelo MOF compuesto por un metamodelo de definición de ontología (ODM) y un metamodelo IFML	Metamodelo HTML5	Desarrollo del enfoque capaz de generar de forma automática interfaces de usuario de aplicaciones Web en HTML a partir de metamodelos ODM e IFML.
[36]	Proponer un enfoque basado en modelos para el desarrollo de aplicaciones móviles basado en el estándar IFML junto con una extensión al lenguaje específico para aplicaciones móviles.	Modelo IFML serializado como XMI	HTML5 CSS3 JavaScript	Herramienta generadora de código que por medio de un modelo en IFML realiza una transformación a código HTML5, CSS3 y JavaScript y mediante la plataforma en línea Build Phonegap obtiene una aplicación .apk

Artículo	Objetivo	Entrada	Salida	Resultados
				para Andrioid o .ipa en el caso de iOS.
[38]	Proponer un enfoque para la generación automática de escenarios de prueba de *front-end* para aplicaciones Web.	Modelo IFML	JavaScript	El desarrollo de reglas de trasformación de modelos IFML a JavaScript, para la obtención de escenarios de prueba de aplicaciones Web.
[39]	Enfoque MDD para la generación automática de aplicaciones multiplataforma Web y móviles a partir de las especificaciones IFML.	Modelo IFML	JSON	IFMLEditing.org, la cual es una herramienta MDD en línea para la especificación y prototipado rápido de aplicaciones de dispositivos móviles y Web a partir de modelos IFML.
[40]	Desarrollar un complemento para la herramienta WebRatio que permita la especificación IFML automática de operaciones CRUD y mostrar cómo repercute en la optimización del esfuerzo	Modelo IFML	No se indica	Complemento AutoCRUD para la herramienta WebRatio que realiza la especificación IFML automatizada de operaciones CRUD.

Artículo	Objetivo	Entrada	Salida	Resultados
	de desarrollo en proyectos reales y un análisis de la optimización del esfuerzo de desarrollo empleando el complemento a desarrollar.			
[41]	Desarrollar un enfoque para la generación impulsada por modelos de la GUI para RIAs empleando IFML.	Modelo IFML	Meta modelo de RIAs MVC	Propuesta del metamodelo RIAs MVC para Aplicaciones Enriquecidas que respeta el patrón MVC.

En la Tabla 2.3 se analizan artículos referentes a la transformación de código existente en un determinado lenguaje de entrada en otro de salida mediante el uso de traductores, igual que en el caso anterior todos los artículos se encuentran en estado finalizado.

Tabla 2.3 Análisis de artículos sobre traductores de código.

Artículo	Objetivo	Entrada	Salida	Resultados
[46]	Proponer una técnica de desarrollo impulsada por el modelo que realice una generación automática de código fuente para convertir la	CSV	Registros SFR	Diseño de una herramienta que transforma la especificación del cliente, a través de distintas fases, en controles de la interfaz gráfica de usuario.

Artículo	Objetivo	Entrada	Salida	Resultados
	especificación en la implementación del producto.			
[49]	Proponer una herramienta de desarrollo, depuración, refactorización y pruebas para el lenguaje JavaScript.	Smalltalk	JavaScript	Desarrollo de la herramienta PharoJS, la cual es un IDE enfocado a la creación y ejecución de pruebas en JavaScript,
[50]	Proponer un enfoque de transformación para facilitar el proceso de migración de elementos Flash a JavaScript	Archivos SWF	JavaScript HTML5	Enfoque de transformación de archivos SWF en JavaScript/HTML5 por medio de procesos de decompilación y traducción.

Como resultado del análisis realizado en el estado de la práctica se concluye que, desde la adopción de IFML como estándar de la OMG en el año 2015, se cuenta con un fuerte interés en la creación de herramientas que obtengan código final a partir de modelos en virtud de la cantidad de trabajos identificados para la generación de interfaces de usuario para RIAs a partir de metamodelos IFML y/o con la combinación de IFML con otras tecnologías. También se observa que dentro de las herramientas analizadas la necesidad de obtener salidas útiles a partir de modelos se encuentra ampliamente satisfecha, pero es posible percatarse que la solución está limitada dado que no se suministran alternativas de personalización de temas o flexibilidad para la selección del lenguaje de salida, lo que en muchos casos obliga al desarrollador a editar gran parte del código generado.

De lo anterior se comprueba que el presente proyecto está alineado con las nuevas tendencias de la industria y la academia (uso y explotación de estándares como IFML, manejo de RIAs) y como diferencial de las investigaciones encontradas, busca incluir la personalización de temas y la elección de un lenguaje de salida, atacando la debilidad mencionada en el párrafo anterior.

2.3 Propuesta de solución

Con la finalidad de establecer el marco de trabajo más acorde a la propuesta de solución plateada en el proyecto de tesis, se llevó a cabo el análisis de las principales herramientas de modelado UML e IFML disponibles en el mercado, se investigaron herramientas para el reconocimiento de lenguajes, así como IDEs y metodologías de desarrollo de software.

Tomando en cuenta las características, documentación, plataformas, costo de licencia, ventajas y desventajas de las TI investigadas, en la Tabla 2.4 se muestra la alternativa de solución establecida para el desarrollo del presente proyecto de tesis.

Tabla 2.4 Alternativa de solución

Aspecto	Propuesta
Herramienta proveedora del modelo IFML	WebRatio Web Platform
Herramienta proveedora del modelo UML	Visual Paradigm
Herramienta para el reconocimiento de lenguaje	ANTLR4
IDE	NetBeans
Metodología de desarrollo	Espiral

Como herramienta para proveer el modelo navegacional en IFML se seleccionó WebRatio, que aunque no es la herramienta que provee el mayor sostén al estándar, puesto que de todas las tecnologías analizadas se posicionó en el segundo lugar, mantiene un alto nivel de soporte de IFML junto con sus elementos extendidos, además de que proporciona una descripción de dichos elementos en un formato XML, se descartó el uso del editor de código abierto de IFML en Eclipse, el cual fue la herramienta con el soporte más completo a los elementos principales y extendidos del estándar IFML por el motivo de que su único método de exportación de modelos es por medio de una imagen en formato JPG, y la otra herramienta analizada IFMLEdit.org aunque provee una descripción del modelo navegacional en formato JSON, después del análisis realizado se consideró que tiene un soporte muy limitado a los elementos del estándar IFML.

Para suministrar el modelo del diagrama de dominio todas las herramientas de modelado analizadas cumplieron en disponer de elementos para modelar diagramas de clases de acuerdo a la especificación UML 2.0, suministrar la exportación de estos modelos en el formato estándar XMI y contar con una amplia documentación, la característica que rompe con la homogeneidad entre las tres herramientas es el costo de su licencia, así que por este factor se seleccionó como la herramienta proveedora del modelo de dominio a Visual Paradigm en su edición modelador, cuya licencia tiene un costo de $72 USD por un periodo de tiempo de 12 meses, mientras que

Rational Software Modeler en su versión diseñador tiene un costo de $228 USD y Enterprise Architect en su versión de escritorio con un costo de $135 USD, ambos costos de las licencias cubren el uso de las herramientas durante un año. Cabe mencionar que se contempló la versión comunitaria de la herramienta Visual Paradigm, pero fue descartada por no tener habilitada la opción de exportación de los modelos realizados en formato XMI.

Una parte fundamental de la herramienta propuesta en el presente proyecto recae en la transformación de un lenguaje a otro, por lo cual, la selección del instrumento adecuado para llevar a cabo el reconocimiento de lenguajes es crucial, dentro de las herramientas analizadas todas toman como notación gramatical de entrada la notación EBNF, empleada para la expresión de gramáticas libres de contexto, poseen las características de ser multiplataforma y de código abierto, y aunque hay una distinción debido a que ANTLR en sus versiones tres y cuatro cuentan con un IDE mientras que Coco\R carece de uno, el factor determinante entre cada una de las herramientas es el algoritmo gramatical que emplean, Coco\R hace uso de un algoritmo LL(1), el cual permite construir de forma automática un analizador determinista descendente con tan solo examinar en cada momento el símbolo actual de la cadena de entrada (símbolo de pre análisis) para saber la producción a aplicar, ANTLR3 por su parte hace uso del algoritmo LL(*), el cual es una extensión al algoritmo LL(k) que permite que no se especifique el parámetro k de búsqueda en la cadena de entrada, sino que busca automáticamente tantos caracteres como sea necesario, hasta que se logra evitar la ambigüedad, y por último, ANTLR4 dispone del algoritmo ALL(*), el cual se basa en el uso de gramáticas libres de contexto y no solo en expresiones regulares, por lo que se reconocen como *tokens* sin contexto, como lo son comentarios anidados, adicionalmente tiene la capacidad de introducir o extraer *tokens* de acuerdo al contexto semántico, además de también ser adecuado para el análisis sin el uso de un escáner debido a su poder de reconocimiento, lo cual es útil para solventar problemas léxicos sensibles al contexto. Con base en la operatividad de cada algoritmo ya mencionado, se opta por seleccionar ANTRL4 que emplea el algoritmo ALL(*) como la herramienta de reconocimiento de lenguajes a ser usada en el presente proyecto.

En cuanto al entorno de desarrollo integrado se seleccionó NetBeans, ya que es considerado como el IDE oficial para el desarrollo en Java 8, y puesto que cuenta con las licencias CDDL y GNU, su uso es gratuito, y aunque el IDE Eclipse tampoco demanda pago por uso dada su licencia de código abierto EPL utilizada por la Fundación Eclipse, este no cuenta con complementos y módulos integrados en su instalación, lo cual hace necesario que el desarrollador configure el ambiente de desarrollo, situación que no se presenta en el caso de NetBeans, el cual ya es un IDE que contempla la inclusión del Entorno de Ejecución Java (JRE) y no requiere una instalación por separado, y se descartó el uso de IntelliJ en su versión comunitaria, que de igual forma no solicita un pago para uso, pero se encuentra muy limitado en los lenguajes que soporta en comparación a los otros IDEs contemplados.

Por último, como metodología de desarrollo se seleccionó la metodología de desarrollo en espiral, la cual entre sus principales ventajas cuenta con que evoluciona a medida que progresa el software a desarrollar, permite al desarrollador reaccionar ante riesgos en cada uno de los niveles de desarrollo de la aplicación, así como el uso de un enfoque de construcción de prototipos en cualquier etapa del proceso evolutivo, se descartó el marco de trabajo Scrum dado que gran parte del éxito de Scrum radica en la experiencia que aportan los profesionales de los equipos de trabajo y en este caso el peso del trabajo a desarrollar recae en un solo individuo, y la metodología de programación extrema se descartó porque requiere de un rígido ajuste a los principios de XP, lo cual hace que no sea adaptable al desarrollo del proyecto propuesto.

Capítulo 3. Aplicación de la metodología

En el presente capítulo se encuentra la descripción de la solución propuesta junto con la narrativa del seguimiento de la metodología de desarrollo seleccionada para la realización de la aplicación. Se tiene la definición de los requerimientos del usuario y del sistema junto con las restricciones para el proyecto, la especificación de la arquitectura de las aplicaciones a generar en cada una de las combinaciones de tecnologías soportadas, se puntualizan las alternativas de distribución de contenido ofrecidas al usuario. Adicional a esto, se presenta de forma detallada cada uno de los elementos que conforman la arquitectura del generador.

3.1 Descripción de la solución

Como se ha mencionado con anterioridad, las RIAs son aplicaciones complejas y cuyo desarrollo demanda un alto costo de tiempo para su diseño e implementación, por lo cual se propone la realización de un generador de RIAs que considere más de un lenguaje de salida e implemente las especificaciones de modelos UML e IFML y que, además, tome en cuenta el uso de plantillas y/o temas.

El esquema de solución definido en este proyecto de tesis se presenta en la Figura 3.1, y se realiza una breve explicación de cada uno de los puntos que la conforman:

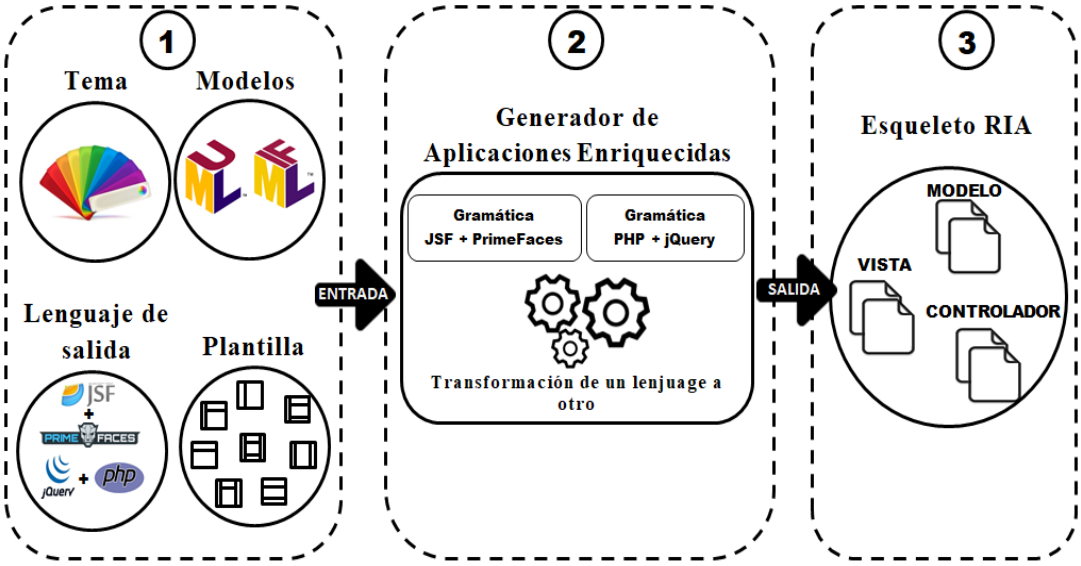

Figura 3.1 Esquema de la solución propuesta

1. Como elementos de entrada, la herramienta espera modelos IFML y UML (sólo diagrama de clases) que representan el *front-end* y *back-end* de la aplicación a generar, la elección de una de las combinaciones de lenguajes disponibles como salida (limitado a JSF con PrimeFaces o PHP con jQuery) y especificaciones de la vista como lo son la selección de un tema y la distribución de los elementos que conformarán la aplicación de salida, entre otros.
2. La solución emplea como instrumentos una herramienta para el reconocimiento de lenguajes y gramáticas predefinidas para realizar la generación del código fuente compilable correspondiente a una aplicación enriquecida con la selección de lenguajes realizada por el usuario y que además cumpla con las especificaciones representadas en los modelos recibidos como entrada y con las directrices indicadas para la vista.
3. Los archivos de la aplicación se agrupan de acuerdo al patrón arquitectónico MVC en vista, modelo y controlador, y su extensión depende del lenguaje señalado por el usuario, cabe mencionar que son esqueletos que cumplen por completo con la navegación indicada en el diagrama navegacional en IFML y con el modelo indicado en UML, pero que no cuentan con código SQL que les proporcione acceso a un repositorio de información.

3.2 Metodología de desarrollo en Espiral

Para solventar la problemática planteada en el presente trabajo de tesis y satisfacer eficazmente los objetivos establecidos se describen los ciclos llevados a cabo para la construcción del generador de Aplicaciones Enriquecidas de Internet, tomando como directriz los elementos del modelo de desarrollo en espiral definidos en [51].

3.2.1 Primera iteración

El primer ciclo corresponde a la primera etapa del generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML, en la cual se contempla su construcción hasta la etapa de análisis semántico de los archivos de entrada.

Siguiendo lo establecido en el primer sector del modelo en espiral, este apartado contiene la especificación de requerimientos del usuario y del sistema junto con las restricciones para el proyecto.

3.2.1.1 Análisis de requerimientos

En un sistema los requerimientos corresponden a la descripción de los servicios que proporciona y sus restricciones operativas, estos se dividen en dos grupos principales, el primero de ellos corresponde a los requerimientos del usuario, que son declaraciones de lo que se espera el sistema suministre y de las restricciones bajos las cuales debe operar, y el segundo concierne con los requerimientos del sistema, que establecen con detalle las funciones, servicios y restricciones operativas del sistema [51].

En la Tabla 3.1, se enlistan cada uno de los requerimientos del usuario, y más adelante se mencionan los requerimientos del sistema correspondientes a cada necesidad del usuario.

Tabla 3.1 Requerimientos del usuario

Número	Requerimiento del usuario
1	El sistema debe permitir al usuario la selección de los archivos que representen el modelo IFML y UML de las Aplicaciones Enriquecidas de Internet a generar.
2	El sistema debe permitir la selección de una combinación de lenguajes de salida para las Aplicaciones Enriquecidas de Internet a generar.
3	El sistema debe permitir la selección de una distribución visual para las Aplicaciones Enriquecidas de Internet a generar.
4	El sistema debe permitir que el usuario agregue un tema para las aplicaciones a generar.
5	La arquitectura de las Aplicaciones Enriquecidas de Internet a generar debe estar basada en el patrón arquitectónico MVC.
6	El sistema debe generar código fuente para Aplicaciones Enriquecidas de Internet de acuerdo a las selecciones realizadas en los puntos anteriores.

En las Tablas 3.2 a 3.7, se describen los requerimientos del sistema respecto a cada especificación del usuario.

Tabla 3.2 Requerimientos del sistema relativos al requerimiento de usuario 1

Requerimiento del usuario	
1	El sistema debe permitir al usuario la selección de los archivos que representen el modelo IFML y UML de las Aplicaciones Enriquecidas de Internet a generar.
Requerimientos del sistema	
1.1	La aplicación debe contar con una interfaz para que se seleccionen los modelos IFML y UML.
1.2	La aplicación debe incluir una validación del contenido de los archivos XML y XMI.

Tabla 3.3 Requerimientos del sistema relativos al requerimiento de usuario 2

Requerimiento del usuario	
2	El sistema debe permitir la selección de una combinación de lenguajes de salda para las Aplicaciones Enriquecidas de Internet a generar.
Requerimientos del sistema	
2.1	La aplicación debe contar con una interfaz para que el usuario seleccione una de las dos posibles combinaciones de lenguajes disponibles, las cuales son JSF enriquecido con PrimeFaces o PHP en conjunto con jQuery.

Tabla 3.4 Requerimientos del sistema relativos al requerimiento de usuario 3

Requerimiento del usuario	
3	El sistema debe permitir la selección de una distribución para las Aplicaciones Enriquecidas de Internet a generar.
Requerimientos del sistema	
3.1	La aplicación debe contar con al menos dos distribuciones disponibles para aplicar y permitir la selección de una de ellas.

Tabla 3.5 Requerimientos del sistema relativos al requerimiento de usuario 4

Requerimiento del usuario	
4	El sistema debe permitir la selección de una combinación de lenguajes de salida para las Aplicaciones Enriquecidas de Internet a generar.
Requerimientos del sistema	
4.1	La aplicación debe contar con una interfaz para que el usuario seleccione el archivo con el tema de acuerdo a las características de la tecnología seleccionada.
4.2	La aplicación debe incluir una validación de que el contenido del archivo indicado como tema corresponde con la tecnología seleccionada.

Tabla 3.6 Requerimientos del sistema relativos al requerimiento de usuario 5

Requerimiento del usuario	
5	La arquitectura de las Aplicaciones Enriquecidas de Internet a generar debe estar basada en el patrón arquitectónico MVC.
Requerimientos del sistema	
5.1	La aplicación, al momento de la generación de código final, debe separar físicamente los archivos generados de acuerdo al patrón arquitectónico MVC

Tabla 3.7 Requerimientos del sistema relativos al requerimiento de usuario 6

Requerimiento del usuario	
6	El sistema debe generar Aplicaciones Enriquecidas de Internet que cumplan con las directrices señaladas para la generación de código final.
Requerimientos del sistema	
6.1	La aplicación debe realizar un proceso de análisis que abarque la lectura, procesamiento y validación de los archivos de entrada.
6.2	La aplicación debe realizar la generación de una representación intermedia en función de los resultados obtenidos por el proceso de análisis.
6.3	La aplicación debe generar del código final esperado con base en la representación intermedia junto con las pautas indicadas por el usuario.
6.4	La aplicación debe notificar al usuario en caso de error en alguna de las etapas del proceso de generación de las Aplicaciones Enriquecidas de Internet.

Como requerimiento general para el funcionamiento del sistema se determina que los requerimientos del 1 al 4 son obligatorios, y en caso de que el usuario intente iniciar el proceso de generación sin especificar alguno de ellos el sistema notifica un error.

3.2.1.2 Restricciones

En esta sección se encuentran las restricciones establecidas para el proyecto que afectan su desarrollo, entre las cuales se encuentran:

A. El tema de color a aplicar debe corresponder a un archivo provisto por el marco de trabajo CSS ThemeRoller para el caso de las aplicaciones PHP con jQuery, y para la combinación de JSF con PrimeFaces es necesaria la obtención un archivo jar por medio del convertidor de temas de PrimeFaces o de su repositorio de temas.
B. El XMI correspondiente a la representación del modelo de negocio, de la aplicación a generar, debe ser un diagrama de clases que tenga concordancia con el modelo de dominio establecido en el archivo XML.

C. La representación XML de los modelos de dominio y navegacional debe ser propia de la herramienta WebRatio Web Platform.
D. La representación XMI del modelo de negocio debe provenir de la herramienta Visual Paradigm.
E. Las Aplicaciones Enriquecidas de Internet a generar no proveen código SQL para el acceso a ningún tipo de repositorio de información, pero deben estar dotadas de datos preestablecidos en el modelo para permitir al usuario la visualización de los elementos generados.
F. Aunque por el momento sólo se cuentan como código final las combinaciones de JSF con PrimeFaces y PHP con jQuery, la aplicación debe estar diseñada de tal forma que permita la posibilidad de que con el tiempo se le agreguen módulos para generar otras combinaciones de lenguajes destino.
G. La herramienta de generación debe cumplir con tener un diseño modular que permita la generación de características particulares como la distribución de elementos dentro de la distribución general.
H. Las versiones a utilizar de cada lenguaje destino son:
    - PHP versión 7.2.5
    - JQuery versión 3.3.1
    - JavaServer Faces versión 2.2
    - PrimeFaces versión 6.1
I. Las versiones requeridas de los modelos tomados como archivos fuente son:
    - Versión del exportador de archivos XMI de Visual Paradigm 7.0.2 y como archivo XMI 2.1.
    - El archivo XML se obtiene de WebRatio Web Platform versión 7.2.16.

3.2.1.2.1 Arquitectura de las aplicaciones a generar

Dentro de las especificaciones del proyecto se establece que la arquitectura de las aplicaciones a generar debe de encontrarse basadas en el patrón arquitectónico MVC, esto sustentado en que:

a) MVC permite al desarrollador identificar fácilmente los componentes de cada capa y su comunicación con los demás componentes existentes en la aplicación.

b) Por la separación en capas del patrón se obtiene un código más claro, flexible y reusable.
c) Su gran nivel de abstracción permite el desarrollo exitoso de aplicaciones Web complejas [52].

Por lo cual, se especifica la arquitectura mostrada en la Figura 3.3 para las aplicaciones a generar con la combinación de lenguajes de PHP y jQuery, en donde: 1) El cliente realiza una petición al controlador, 2) El controlador realiza una solicitud al modelo, 3) El modelo entrega una respuesta con los datos solicitados, 4) El modelo , que contiene archivos PHP y JavaScript, realiza el procesamiento necesario para satisfacer la solicitud del cliente, 5) La vista recibe la información del controlador y, por último 6) La vista expone el resultado al cliente.

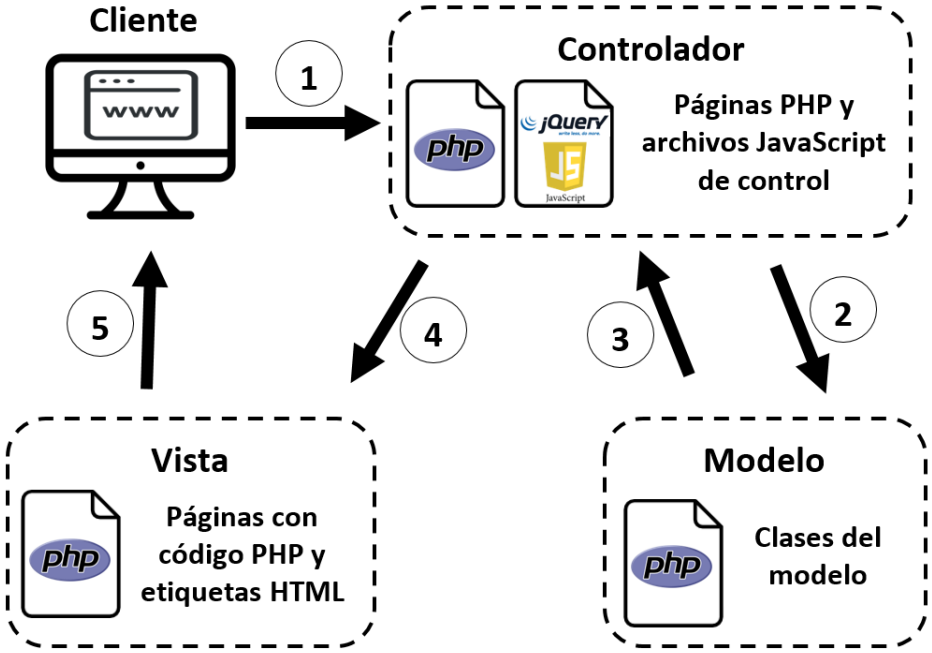

Figura 3.2 Arquitectura basada en MVC de las aplicaciones a generar en PHP con jQuery

En la Figura 3.4 se muestra la arquitectura establecida para las aplicaciones a generar con la combinación de lenguajes de JSF con PrimeFaces, bajo la pauta de Modelo 2, ya introducido en el presente documento, en donde: 1) El cliente realiza una petición al servidor que pasa al

controlador de JSF más los *Beans* administrados, 2) El controlador interpreta la solicitud como un comando para el modelo, 3) El controlador hace que el resultado de la operación se encuentre disponible para la vista, 4) La vista hacer uso de los JavaBeans para el despliegue de información y , por último, 5) envía una respuesta al cliente.

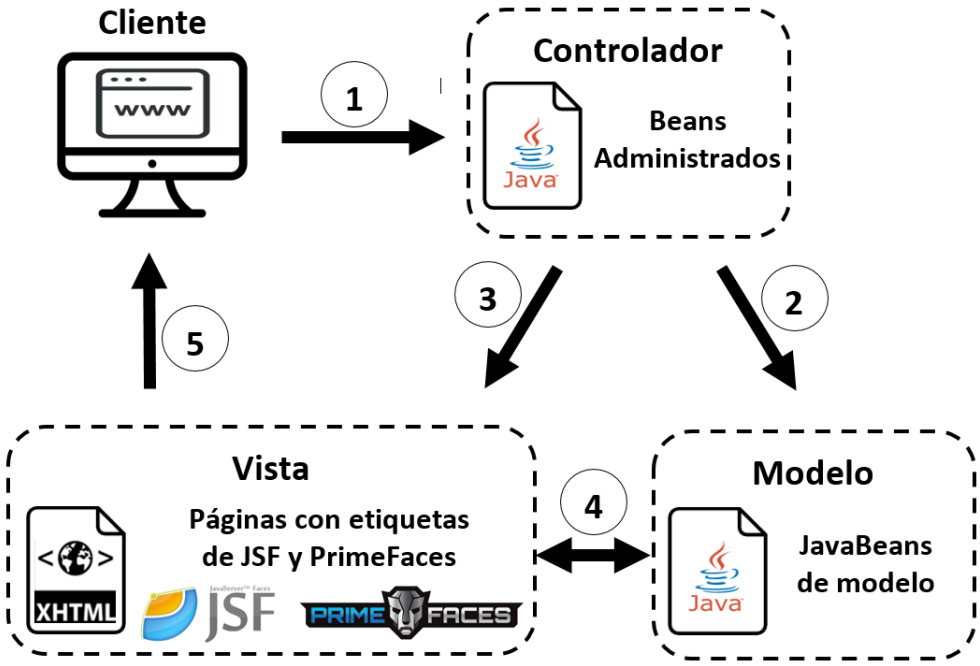

Figura 3.3 Arquitectura basada en MVC de las aplicaciones a generar en JSF con PrimeFaces.

Es pertinente hacer hincapié en que los conceptos de modelo, ya sean JavaBeans o clases PHP, no cuentan con ningún código SQL para el acceso a un repositorio de información, ya que se espera que eventualmente estas clases sean complementados por el desarrollador para proveer el acceso al repositorio de información que requiera, el cual puede ser desde un intermediario para elementos de Hibernate o de EJBs de sesión, una base de datos relacional, modelos de minería de datos, o hasta acceso a tripletas de tipo BigData, por mencionar algunas. Lo que proporcionan los elementos del modelo son datos fijos para permitir que el usuario visualice los elementos generados en las aplicaciones.

3.2.1.2.2 Especificación de las distribuciones de contenido a soportar

Recordando que una de las limitaciones encontradas en las herramientas para la obtención de código final a partir de modelos analizadas en el estado del arte es que las salidas que generan cuentan con una distribución de contenido por omisión, lo cual reduce la utilidad del código generado, ya que, si esa distribución de contenido no es lo requerido por el desarrollador, éste debe editar gran parte del código y lo que puede derivar en errores dentro de la aplicación generada, por tal motivo, como parte de la solución propuesta se establecen cuatro opciones de distribuciones de contenido, representadas en la Figura 3.5.

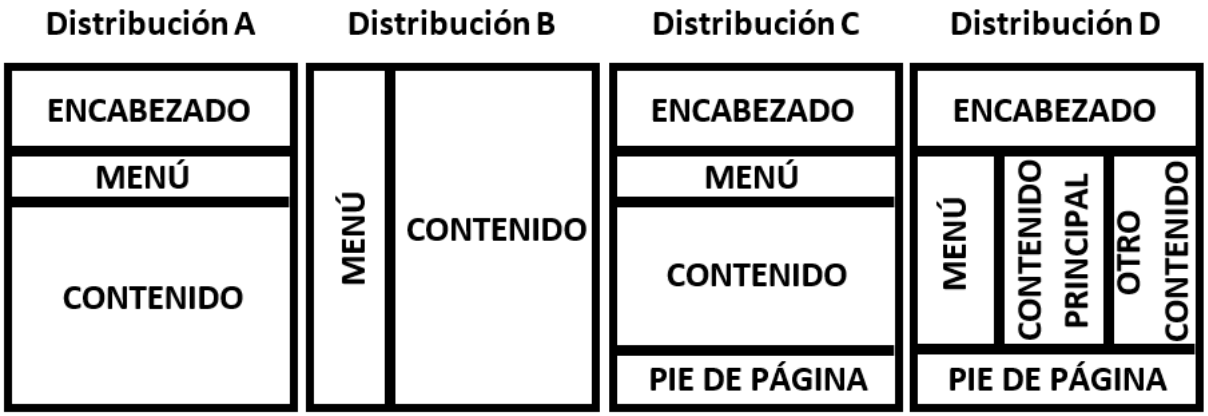

Figura 3.4 Distribuciones de contenido soportadas por el generador de aplicaciones

Dentro de las cuatro distribuciones la sección de “menú” corresponde al apartado en donde se coloca los elementos de navegación, y las partes de “contenido” y ”contenido principal” (para la distribución D), corresponden a la sección principal de las páginas a generar en donde se colocan los elementos que representan los componentes de la vista definidos en el modelo navegacional.

3.2.1.3 Evaluación de riesgos

Un riesgo se define como la posibilidad de que un evento adverso ocurra. Estos riesgos representan una amenaza para la culminación exitosa del proyecto, por tal motivo una parte importante del proceso de desarrollo en espiral corresponde a la identificación de riesgos y a la implantación de acciones para minimizar sus efectos en caso de que aparezcan [51].

Se comenzó el análisis estableciendo las métricas para la probabilidad de aparición de un riesgo, las cuales son:

- **Muy bajo:** Porcentaje de aparición menor del 10%.
- **Bajo:** Porcentaje de aparición desde el 10% hasta no mayor al 25 %.
- **Moderado:** Porcentaje de aparición entre el 25% y 50%.
- **Alto:** Porcentaje de aparición mayor del 50% y no superior al 75%.
- **Muy alto:** Porcentaje de aparición mayor al 75%.

Y para la medición del impacto de cada riesgo se establecen los criterios de: catastrófico, serio, tolerable o insignificante [51]. En la Tabla 3.8 se encuentran los principales riesgos identificados junto con su probabilidad de aparición e impacto.

Tabla 3.8 Análisis de los principales riesgos identificados

Número	Riesgo	Porcentaje de aparición	Impacto
1	La herramienta WebRatio deje de proporcionar la representación XML del modelo de dominio y navegacional.	Bajo	Catastrófico
2	La herramienta WebRatio modifique la representación XML del modelo de dominio y navegacional.	Moderado	Tolerable
3	Complejidad en el uso de ANTLR.	Moderado	Serio
4	Cambios drásticos en los marcos de trabajo a generar.	Bajo	Serio

Para solventar la posible aparición de los riesgos considerados, en la Tabla 3.9, se establecen las estrategias de gestión de riesgos.

Tabla 3.9 Estrategias de gestión de riesgos

Número de riesgo	Estrategia para la gestión del riesgo
1	Identificar una herramienta alternativa que provea una representación ,en algún lenguaje de alto nivel, de modelos navegacionales IFML aparte de la plataforma WebRatio.
2	Realizar a la gramática para el reconocimiento del XML, que representa el modelo de dominio y navegacional, de tal forma que cambios en la representación de los modelos provistos por la herramienta no la dejen inutilizable y que dichos cambios, si son relevantes para la generación de código, puedan ser fácilmente integrados.
3	Manejo de libros y tutoriales del propio autor.
4	No se cuenta con una estrategia para el manejo de éste riesgo ya que no afecta el desarrollo de la aplicación, su impacto radica en el uso final de la herramienta.

3.2.1.4 Desarrollo y validación

Este apartado describe cómo se llevó a cabo el desarrollo de la primera etapa del generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML.

3.2.1.5 Arquitectura del generador de Aplicaciones Enriquecidas de Internet

El desarrollo de este proyecto considera la teoría de compiladores, sin embargo, no se trata de un traductor tradicional que siga el proceso de traducción en su totalidad como lo establecido en [25], el enfoque aplicado corresponde a lo que en [26] se define como una aplicación de lenguaje, ya que se hace uso de solo algunas partes del proceso de traducción sin tratarse de un

compilador de un lenguaje de alto nivel, y para su construcción se empleó la herramienta ANTLR4.

Las Figuras 3.5 y 3.6 representan la arquitectura del generador de Aplicaciones Enriquecidas de Internet, la cual se encuentra dividida en dos partes, la primera de ellas concierne a la lectura, procesamiento y validación de los archivos de entrada, la generación de los ASTs y de las tablas de símbolos, la segunda parte de la arquitectura atañe a la generación de la representación intermedia en función de los resultados obtenidos por el proceso de análisis, y a la generación del código final esperado.

Cada uno de los elementos representados en la arquitectura del generador se encuentran documentados detalladamente más adelante en este documento.

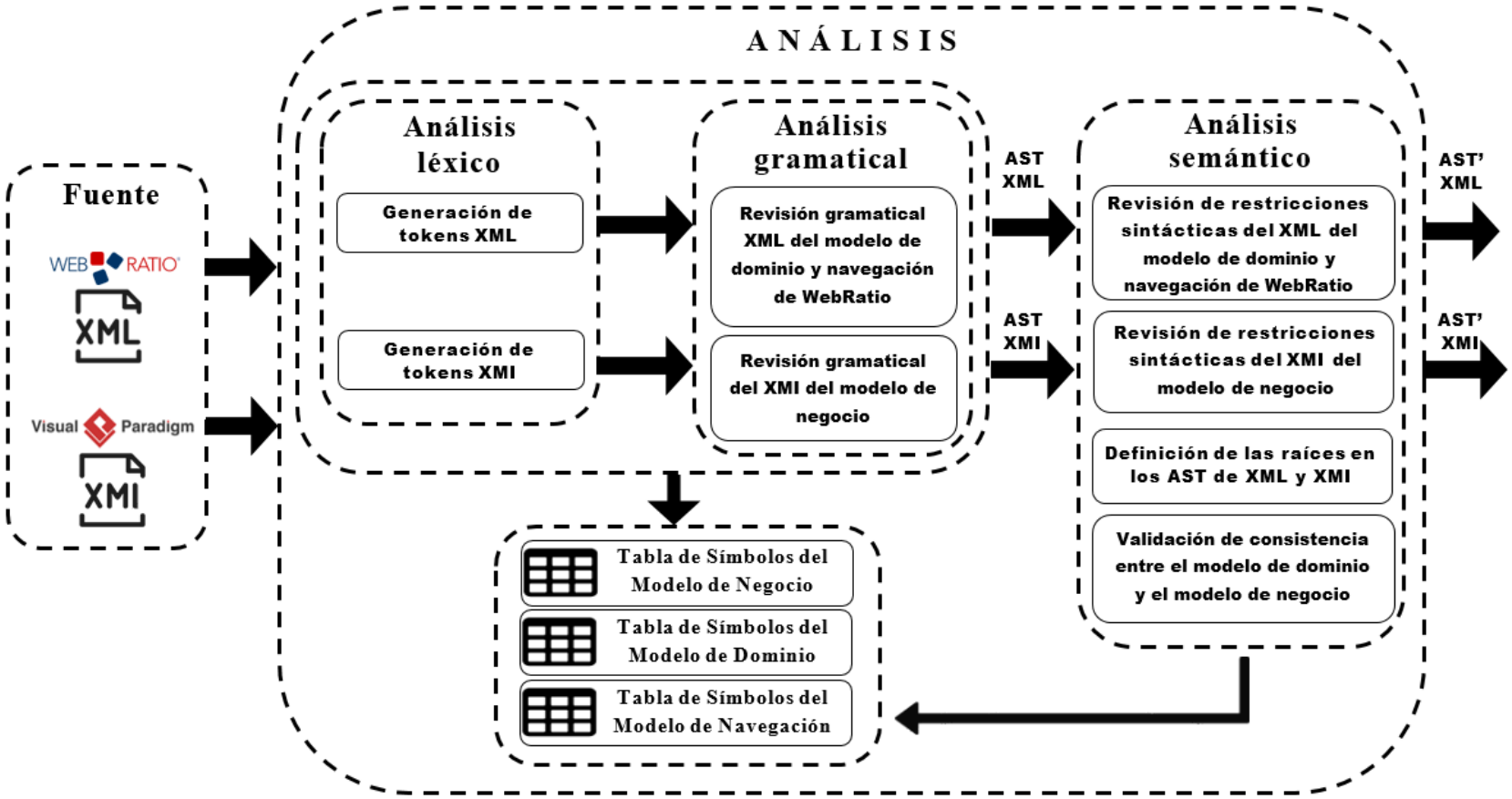

Figura 3.5 Arquitectura del generador de Aplicaciones Enriquecidas de Internet, fase de análisis

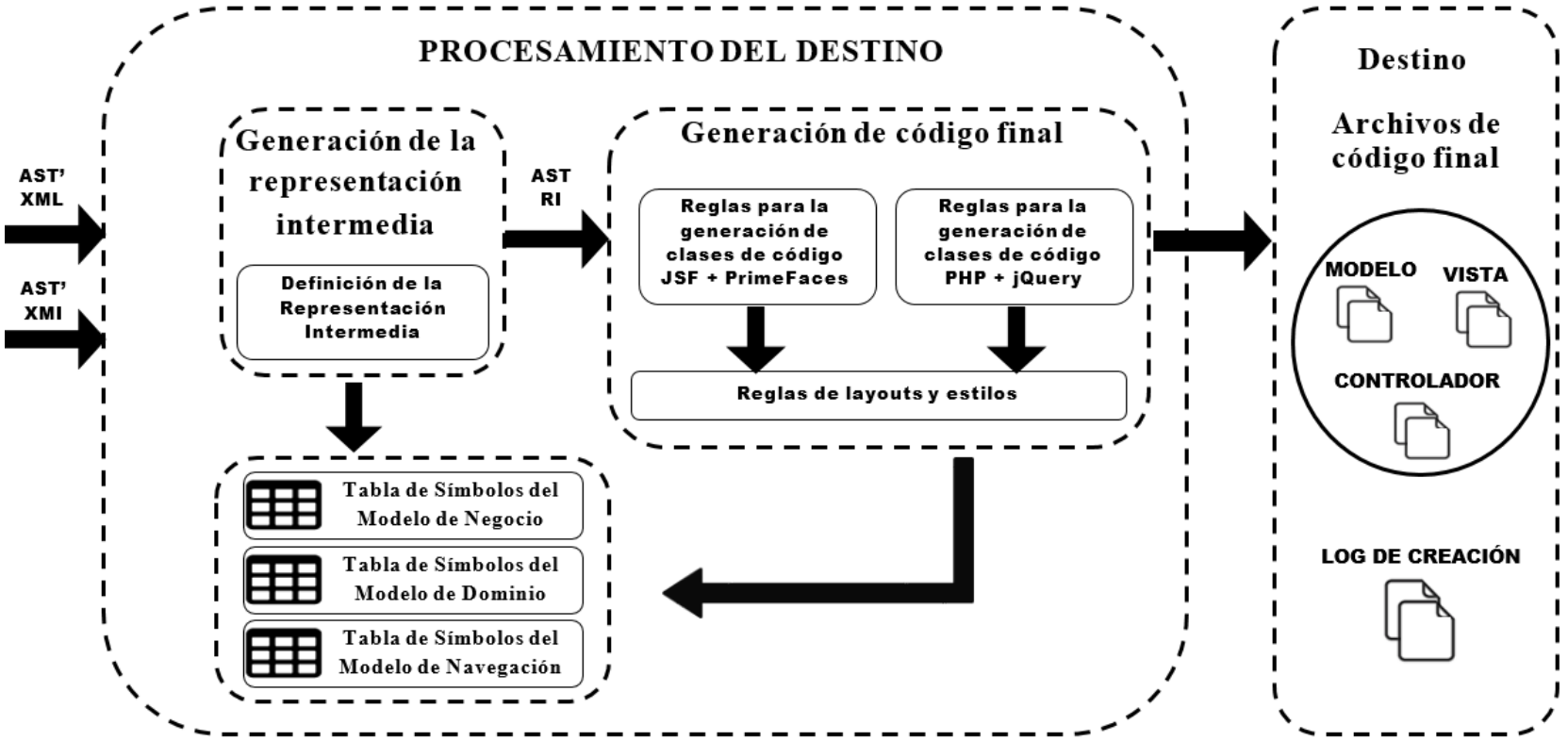

Figura 3.6 Arquitectura del generador de Aplicaciones Enriquecidas de Internet, fase del procesamiento del destino

3.2.1.6 Fuente

El generador opera en función de dos archivos fuente, el primero de ellos corresponde al XML generado por la herramienta WebRatio que contiene la información del modelo de dominio y el modelo navegacional de las aplicaciones a generar, y el segundo archivo corresponde al XMI que contiene información del modelo de negocio necesaria para complementar la generación de las clases del modelo (JavaBeans o clases PHP, dependiendo cual sea el caso).

Fue necesario realizar un análisis detallado de estos elementos de entrada para identificar la información relevante para la generación de código, tal información se encuentra plasmada en los siguientes apartados:

- Apéndice A: Mapeo IFML a WebRatio, comprende el mapeo realizado entre los elementos definidos por el estándar IFML y su correspondiente implementación en la herramienta WebRatio, indica las etiquetas XML generadas para cada elemento implementado por WebRatio y señala los atributos relevantes soportados en esta etapa del generador de código.
- Apéndice B: Mapeo XMI, abarca el análisis de la representación XMI de los elementos que conforman un diagrama de clases UML a soportar por el generador y la especificación de los atributos relevantes para la generación de código final esperada.

3.2.1.7 Análisis léxico

La primera fase de un compilador corresponde al análisis léxico. El analizador léxico lee la secuencia de caracteres que compone el programa fuente y agrupa a los caracteres en secuencias significativas llamadas lexemas[25].

En esta etapa se reconocieron los elementos que correspondían a las palabras o símbolos válidos dentro de los archivos XML y XMI, y que además sean relevantes para la generación de código final esperada. Para lograr este objetivo se codificó una serie de reglas léxicas para la lectura de cada uno de los archivos.

El Listado 3.1 muestra un fragmento de las reglas léxicas definidas para el archivo XML, que contiene la información del modelo de dominio y del modelo navegacional, en donde, por mencionar algunos, se observa en las líneas 20 y 21 que es relevante asignar *tokens* a la aparición de las palabras *Entity* y *Attribute* dentro de las etiquetas que conforman el XML.

Listado 3.1 Fragmento de código del analizador léxico XML

```
01. lexer grammar IFMLLexer;
02. COMENTARIO: '<!--' .*? '-->'  -> skip;
03. ETIQ_INI: '<'  -> pushMode(ETIQUETA);
04. XML_INI: '<?xml' WS_INTERNO -> pushMode(ETIQUETA);
05. WEBML_INI: '<?webml' WS_INTERNO -> pushMode(ETIQUETA);
06. WS_EXTERNO: (' '|'\t'|'\r'? '\n');
07. TEXTO: ~[<]+;
08.
09. mode ETIQUETA;
10. ETIQ_FIN: '>' ->popMode;
11. XML_PROC_INST_FIN: '?>' -> popMode;
12. ETIQ_AUTOFIN: '/>' -> popMode;
13. DIAG: '/';
14. IGUAL: '=';
15. CADENA: '"' ~[<"]*? '"'
16.       | '\'' ~[<']*? '\'';
17. ET_WEB_PROJECT: 'WebProject';
18. ET_SERVICE_DATA_PROVIDERS: 'ServiceDataProviders';
19. ET_DATA_MODEL: 'DataModel';
20. ET_ENTITY: 'Entity';
21. ET_ATTRIBUTE: 'Attribute';
22. ET_RELATIONSHIP: 'Relationship';
```

El mismo principio se observa en el Listado 3.2, en donde las líneas 21 y 22 definen que es necesaria la asignación de un *token* a la aparición de las palabras ownedMember y *xmi:Extension* dentro del XMI que define el diagrama del modelo de dominio, ya que son elementos de conocimiento relevante para la generación de código.

Listado 3.2 Fragmento del analizador léxico XMI

```
01. lexer grammar XMILexer;
02. COMENTARIO: '<!--' .*? '-->'  -> skip;
03. ETIQ_INI: '<'  -> pushMode(ETIQUETA);
04. XML_INI: '<?xml' WS_INTERNO -> pushMode(ETIQUETA);
05. XMI_INI: '<xmi:XMI' WS_INTERNO -> pushMode(ETIQUETA);
06. XMI_DOCUMENTATION: '<xmi:Documentation' .*? '</xmi:Documentation>\n\t'->skip;
07. UML_DIAGRAM: '\n\t<uml:Diagram'  .*?  '</uml:Diagram>' -> skip;
08. WS_EXTERNO: (' '|'\t'|'\r'? '\n');
09. TEXTO: ~[<]+;
10.
11. mode ETIQUETA;
12. ETIQ_FIN: '>' ->popMode;
13. XML_PROC_INST_FIN: '?>' -> popMode;
14. XMI_FIN: '/xmi:XMI>' -> popMode;
15. ETIQ_AUTOFIN: '/>' -> popMode;
16. DIAG: '/';
17. IGUAL: '=';
18. CADENA: '"' ~[<"]*? '"'
19.       | '\'' ~[<']*? '\'';
20. ET_UML_MODEL: 'uml:Model';
21. ET_OWNED_MEMBER: 'ownedMember';
22. ET_XMI_EXTENSION: 'xmi:Extension';
```

La Figura 3.7 ejemplifica la aplicación de las reglas léxicas definidas para la obtención de los *tokens* de una etiqueta del archivo XML que describe un atributo del modelo de dominio.

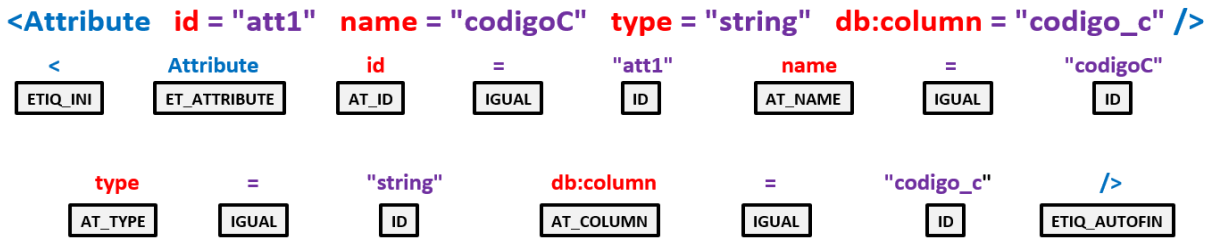

Figura 3.7 Ejemplo de la obtención de *tokens* de una etiqueta XML

Como resultado de esta fase el generador obtiene las listas de todos los *tokens* que componen cada uno de los archivos fuente.

3.2.1.8 Análisis gramatical

La segunda fase del proceso de traducción corresponde al análisis de sintaxis o gramatical, en donde el analizador utiliza los *tokens* producidos por el analizador léxico para crear una representación similar a un árbol que simboliza la estructura gramatical de la secuencia de *tokens* [25].

Es esta fase fue necesario establecer el conjunto de reglas que describen la sintaxis del contendido esperado para cada uno de los archivos fuente. El Listado 3.3, en las líneas 2, 5, 8 y 10, muestran las reglas especificadas sobre el contenido, que resulta relevante para el generador, que puede recibir la etiqueta *DataModel* dentro del archivo XML que representa los modelos de dominio y navegacional, y las reglas definidas en las líneas 13 y 15 representan el resto de elementos que puede albergar la etiqueta pero que no representan información relevante para el proyecto.

Listado 3.3 Fragmento de la gramática IFML que define el contenido de la etiqueta *DataModel* dentro del archivo XML

```
01. dataModel_cont: /* ----- Definición de las etiquetas que puede contener
                                                        'DataModel' ----- */
02.                 datosTexto? ETIQ_INI ET_ENTITY entity_atr+ ETIQ_FIN
                    (datosTexto? entity_cont datosTexto?)+ ETIQ_INI DIAG ET_ENTITY
                    ETIQ_FIN datosTexto?
03.                 #etiq_Entity_DataModel
04.
05.               | datosTexto? ETIQ_INI ET_RELATIONSHIP relationship_atr+ ETIQ_FIN
                     (datosTexto? relationship_cont datosTexto?)+ ETIQ_INI DIAG
                    ET_RELATIONSHIP ETIQ_FIN datosTexto?
06.                 #etiq_Relationship_DataModel
07.
08.               | datosTexto? ETIQ_INI ET_DATA_BASE db_database_atr+ ETIQ_FIN
                    datosTexto? (contenidoIrrelevante datosTexto?)? ETIQ_INI DIAG
                    ET_DATA_BASE ETIQ_FIN datosTexto?
09.                 #etiq_dbDatabase_DataModel
```

```
10.             | datosTexto? ETIQ_INI ET_DATA_BASE db_database_atr+ ETIQ_AUTOFIN
                  datosTexto?
11.               #etiqAF_dbDatabase_DataModel
12.
13.             | ETIQ_INI ID atributo* ETIQ_FIN datosTexto?
                   (contenidoIrrelevante datosTexto?)?  ETIQ_INI DIAG ID ETIQ_FIN
14.               #etiq_NoRelevante_DataModel
15.             | ETIQ_INI ID atributo* ETIQ_AUTOFIN
16.               #etiqAF_NoRelevante_DataModel
17.             ;
```

Por medio de la definición de una gramática, ANTLR crea una estructura de datos llamada árbol de análisis sintáctico o árbol de sintaxis, la cual representa el orden de las sentencias de entrada y cómo éstas se agrupan en frases, en la Figura 3.8 se ilustra este concepto [26].

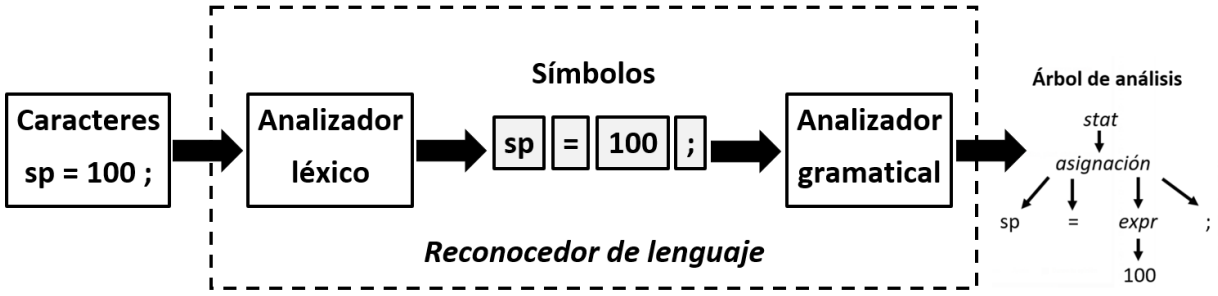

Figura 3.8 Reconocedor de lenguaje

Una vez definidas las gramáticas para el contenido esperado en los archivos XML y XMI, a través del uso de la herramienta ANLTR4, se generaron los archivos necesarios para el reconocimiento de las sentencias del lenguaje descrito por cada una de las gramáticas. Los archivos obtenidos de la herramienta y que fueron utilizados en el desarrollo de la aplicación se muestran en la Figura 3.9. Enseguida se describen cada uno de los archivos con extensión Java utilizados:

- IFMLLexer / XMILexer: Este archivo contiene la definición de la clase del analizador léxico que ANTLR genera al analizar las reglas léxicas y los literales gramaticales definidos en los archivos con extensión G4 IFMLLexer y XMILexer.

- IFMLParser / XMIParser: Contiene la definición de la clase del analizador específica de las gramáticas establecidas en IFMLParser y XMIParser.
- IFMLParserListener / XMIParserListener: Por omisión el analizador de ANTLR construye un árbol a partir de la entrada, y para su recorrido lanza eventos a un objeto escucha que proporciona la misma herramienta. Este archivo contiene la definición de una interfaz que describe los métodos a implementar para el recorrido del AST.
- IFMLParserBaseListener / XMIParserBaseListener: Esta clase contiene un conjunto de implementaciones vacías, por omisión, de cada uno de los métodos definidos en la interfaz ParserListener.

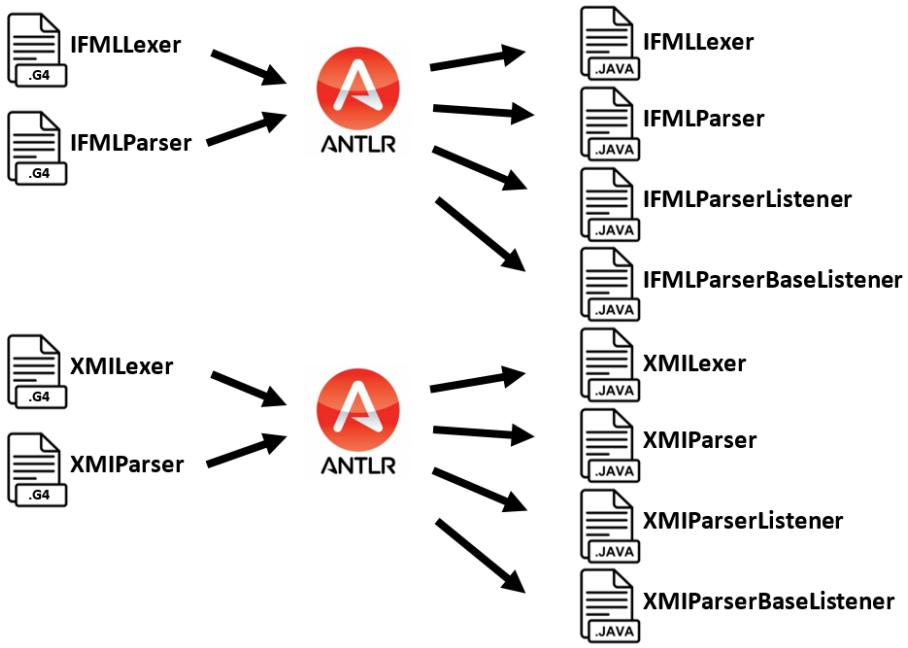

Figura 3.9 Archivos generados por ANTLR4 para el reconocimiento y recorrido del contenido de los archivos XML y XMI

ANTLR provee los patrones de Escuchas y Visitantes para el recorrido de ASTs. Entre sus principales diferencias se encuentran:

- En un escucha los métodos de visita a los nodos hijos se llaman de forma automática por un objeto de tipo ParseTreeWalker mediante el cual se recorre el AST, mientras que en un visitante los métodos deben indicar de forma explícita la visita a sus nodos hijos, ya

que el no invocar el método visit() en los nodos hijos implica que no son visitados al realizar el recorrido del árbol.

- Los escuchas los métodos no tienen un valor de retorno, mientras que los métodos en los visitantes se personalizan para devolver el valor de retorno especificado por el desarrollador.
- Los visitantes utilizan una pila de llamadas para la gestión de los recorridos del árbol, en cambio los escuchas utilizan una pila explícita gestionada por el objeto que realiza el recorrido del árbol [26].

La Figura 3.10 muestra las clases DiagramaClasesRecorridoUno y DiagramaClasesRecorridoDos, que realizan la recuperación de la información alojada en cada etiqueta reconocida en el archivo XMI. La implementación de dichas clases corresponde con el patrón de Visitantes para el recorrido de los ASTs.

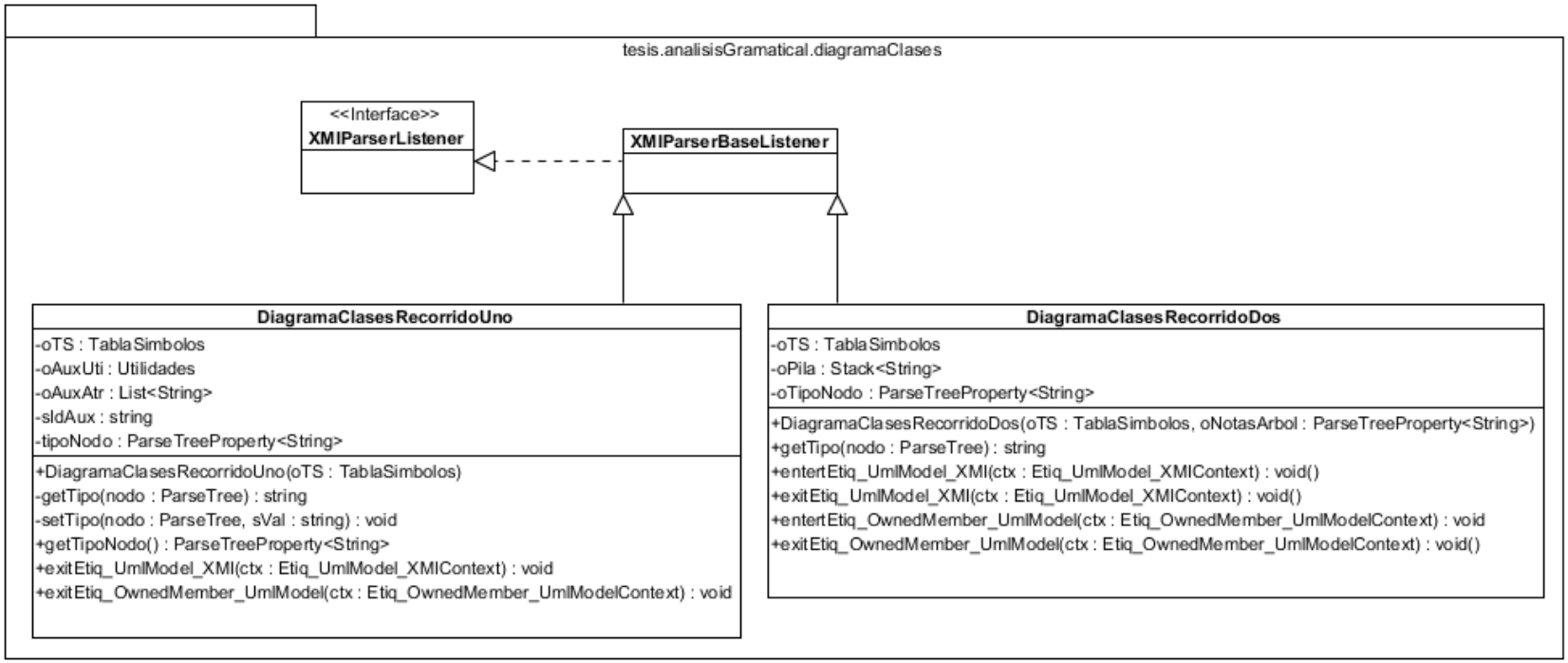

Figura 3.10 Diagrama de clases del paquete tesis.analisisGramatical.diagramaClases

Para la extracción de la información contenida en las etiquetas que integran cada uno de los archivos fuente se empleó la estrategia de realizar dos recorridos sobre sus correspondientes ASTs. En el primer paso por los árboles se visitan los métodos de salida de cada regla, esto porque se trata del punto en que ya se tienen disponibles todos los datos sobre el reconocimiento de una etiqueta por una regla, se realiza una recuperación de datos hallados en el contexto disponible y se realizan anotaciones a los nodos visitados, mientras que en el segundo paso por los árboles se extrae la información sobre el alcance de la etiquetas a partir de las anotaciones que se le añadieron a los nodos en el primer recorrido.

En el Listado 3.4 se muestra el código para la recuperación de los datos de la regla *Etiq_Entity_DataModel* del archivo XMI en un primer recorrido a su AST, en donde se observa que se hace uso de un objeto del contexto llamado ctx (línea 02), que contiene todos los *tokens* de la etiqueta *Entity*, del cual se obtienen los atributos de la etiqueta y se almacenan como una entrada a la tabla de símbolos (líneas 04 a 09) y por último se realiza una anotación al nodo del árbol con el objeto del contexto y el identificador de la etiqueta (línea 10).

Listado 3.4 Código de visita al nodo *Etiq_Entity_DataModel* en el primer recorrido del AST

```
01. @Override
    public void exitEtiq_Entity_DataModel(IFMLParser.Etiq_Entity_DataModelContext
    ctx) {
02.     for(Entity_atrContext atributo : ctx.entity_atr())
03.          arrAtributos.add(atributo.getText());
04.     oTS.guarda(new EntradaTabla(ctx.start.getLine(),
05.                oAuxUti.extraerID(arrAtributos),
06.                "",
07.                oAuxUti.extraerNombre(arrAtributos),
08.                oAuxUti.extraerAtributos(arrAtributos,
                   oAuxAtr.getEntity_atr()).toString(),
09.                "Entity"));
10.     setTipo(ctx, oAuxUti.extraerID(arrAtributos));
11.     arrAtributos.clear();
```

En el listado 3.5 se halla la implementación a los métodos de entrada y salida para la regla *Etiq_Entity_DataModel* en el segundo paso por el AST del XMI, se observa que el método de entrada a la regla (líneas 01 a 04) actualiza el alcance en la entrada de la tabla de símbolos de la etiqueta con el objeto situado al principio de la pila (línea 02) y apila el contexto del nodo anotado en el recorrido previo (línea 3), mientras que en el método de salida de la regla (líneas 06 a 08) se saca de la pila el contexto de la etiqueta (línea 07).

Listado 3.5 Código de visita al nodo *Etiq_Entity_DataModel* en el segundo recorrido del AST

```
01. @Override
    public void enterEtiq_Entity_DataModel(IFMLParser.Etiq_Entity_DataModelContext
    ctx) {
02.     oTS.actualizaAlcance(ctx.start.getLine()+"-"+getTipo(ctx), pila.peek());
03.     pila.push(getTipo(ctx));
04. }
05.
06. @Override
    public void exitEtiq_Entity_DataModel(IFMLParser.Etiq_Entity_DataModelContext
    ctx) {
07.     pila.pop();
08. }
```

Como parte del proceso de traducción, un compilador obtiene datos sobre las diversas entidades manipuladas por el programa que se está traduciendo, y debe almacenar muchos tipos distintos de información, como nombres de variables, constantes definidas, funciones y etiquetas, por mencionar algunas [53].

La tabla de símbolos definida para almacenar los datos relevantes de cada etiqueta de los archivos XML y XMI se encuentra representada en la Figura 3.11. Esta estructura se diseñó considerando necesario que el traductor fuera capaz de encontrar rápidamente el registro de cada etiqueta y sus características asociadas.

Los atributos relevantes de cada etiqueta que son almacenados como una entrada a la tabla de símbolos son el número de línea (dentro del archivo) en que se encuentra la etiqueta, el

identificador y nombre que tiene definidos, los atributos relevantes que posee para la generación de código, su alcance (la etiqueta que anida a esta etiqueta), el tipo (nombre como tal de la etiqueta) y una bandera para señalar si la entrada pasó el proceso de verificación semántica.

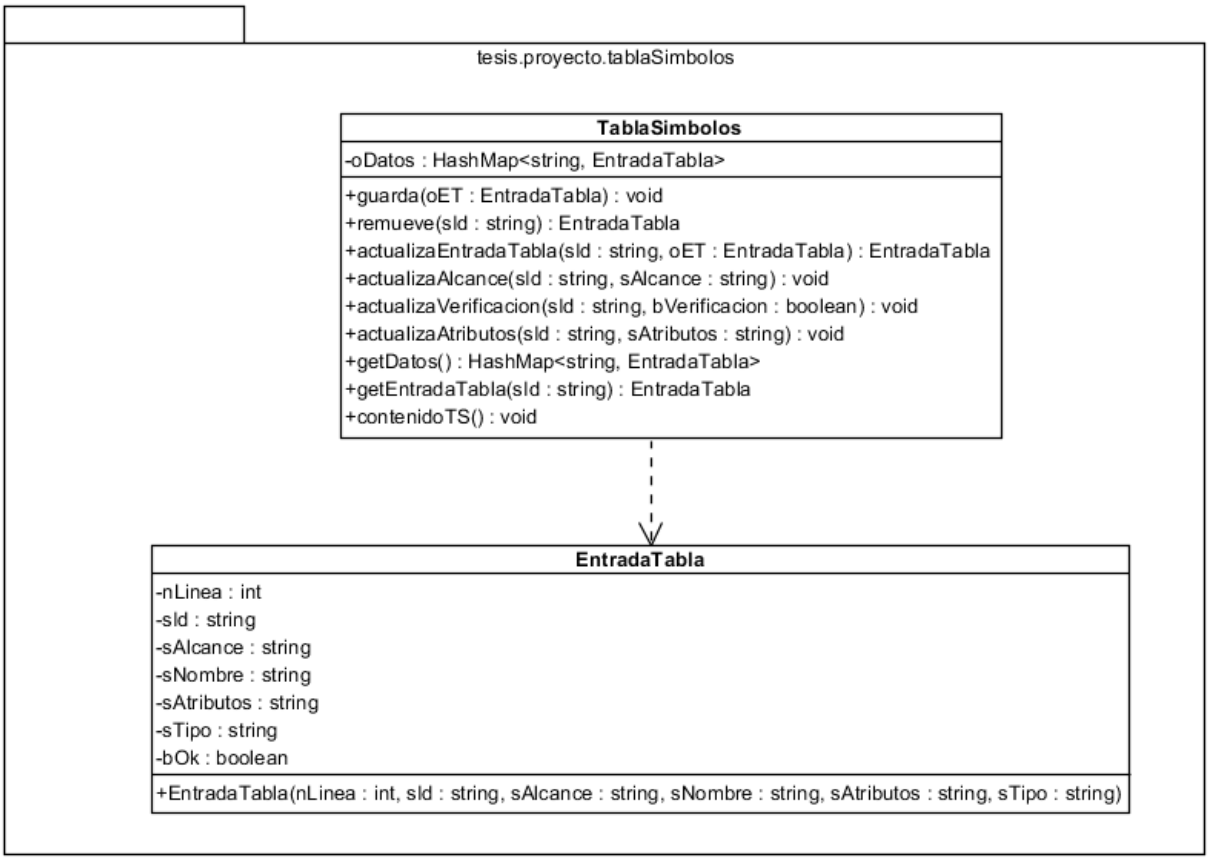

Figura 3.11 Diagrama de clases del paquete tesis.proyecto.tablaSimbolos

3.2.1.9 Análisis semántico

En el marco de un compilador de un lenguaje de alto nivel, el analizador semántico utiliza el árbol de sintaxis y la información en la tabla de símbolos para llevar a cabo la verificación de la coherencia semántica del programa de origen con la definición del lenguaje, además de recopilar información de tipos para su uso posterior durante la generación de código intermedio.

Dentro de las verificaciones más importantes del análisis semántico en los compiladores de alto nivel es la verificación de tipos, donde el compilador comprueba que cada operador tenga

operandos coincidentes, la verificación de coerción en casos en que la especificación del lenguaje lo permita, así como revisión de flujos de control, unicidad y consistencia de nombres, entre otras [25].

En particular para las gramáticas especificadas en el proyecto se realizaron una serie de comprobaciones semánticas, entre las más sobresalientes se encuentran:

a) Comprobación de la consistencia del XML y XMI.
b) Revisión de la definición de identificadores y su uso.
c) Verificación de contexto y alcance.
d) Comprobación del flujo de navegación del modelo navegacional.
e) Evaluación de la consistencia entre el modelo de dominio y el modelo de negocio.

La validación a fue necesaria para verificar que el contenido de los archivos fuente efectivamente correspondiera a un XML bien formado, es decir, que se trata de documentos que siguen las recomendaciones XML especificadas por el W3C, y se implementó con el uso Java SAX, el cual es un mecanismo de acceso a documentos XML que cuenta con una verificación de documentos XML bien formados. Los resultados de esta revisión se encuentran en la Figura 3.12. en donde se observa que al leer un documento que no cumple con la especificación XML de que todo elemento debe tener su correspondiente etiqueta de inicio y de cierre, se marca el error correspondiente a un XML mal formado.

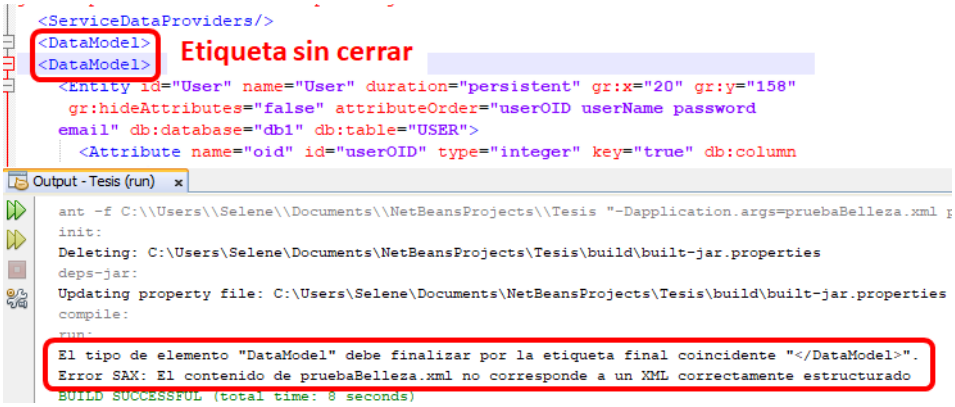

Figura 3.12 Resultado de la revisión semántica de un XML bien formado

La revisión b concerniente a la definición de identificadores y su uso se refiere a la comprobación de que los identificadores, a los que puede hacer referencia una etiqueta, efectivamente se encuentren definidos dentro del documento. En la Figura 3.13 se muestra el resultado de esta verificación al leer un archivo modificado que hace referencia a un identificador no definido dentro de la tabla de símbolos.

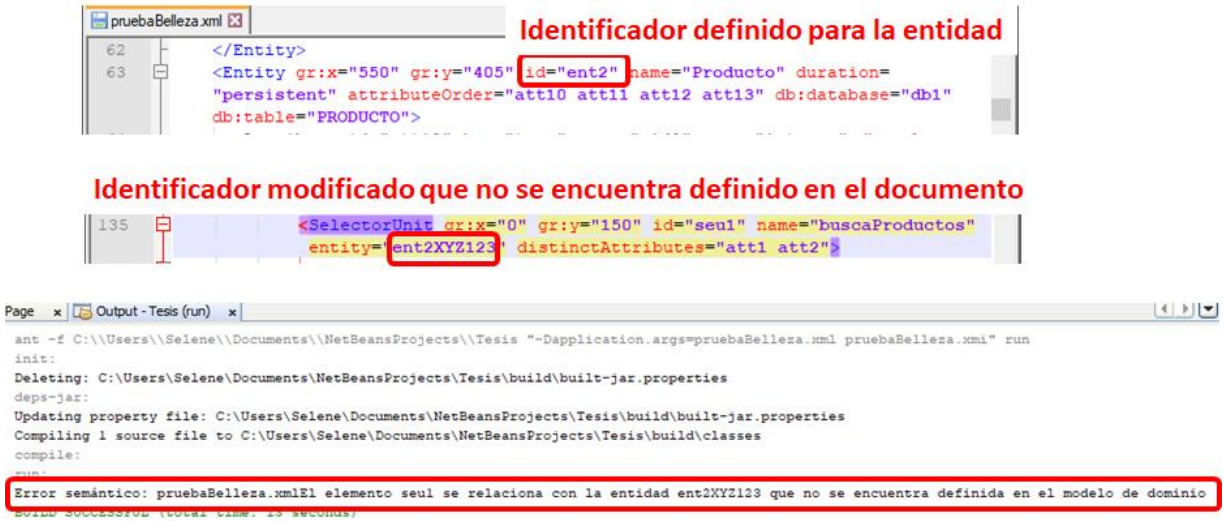

Figura 3.13 Resultado de la revisión semántica de identificadores y su uso

Para la implementación de la revisión c, respecto a la verificación de contexto y alcance. se verifica que dentro de un mismo contexto los identificadores y nombres de elementos definidos sean únicos. La Figura 3.14 muestra el resultado de esta verificación semántica al encontrarse dentro del alcance de la etiqueta *Entity* con identificador “ent1” tres atributos con el mismo identificador definido.

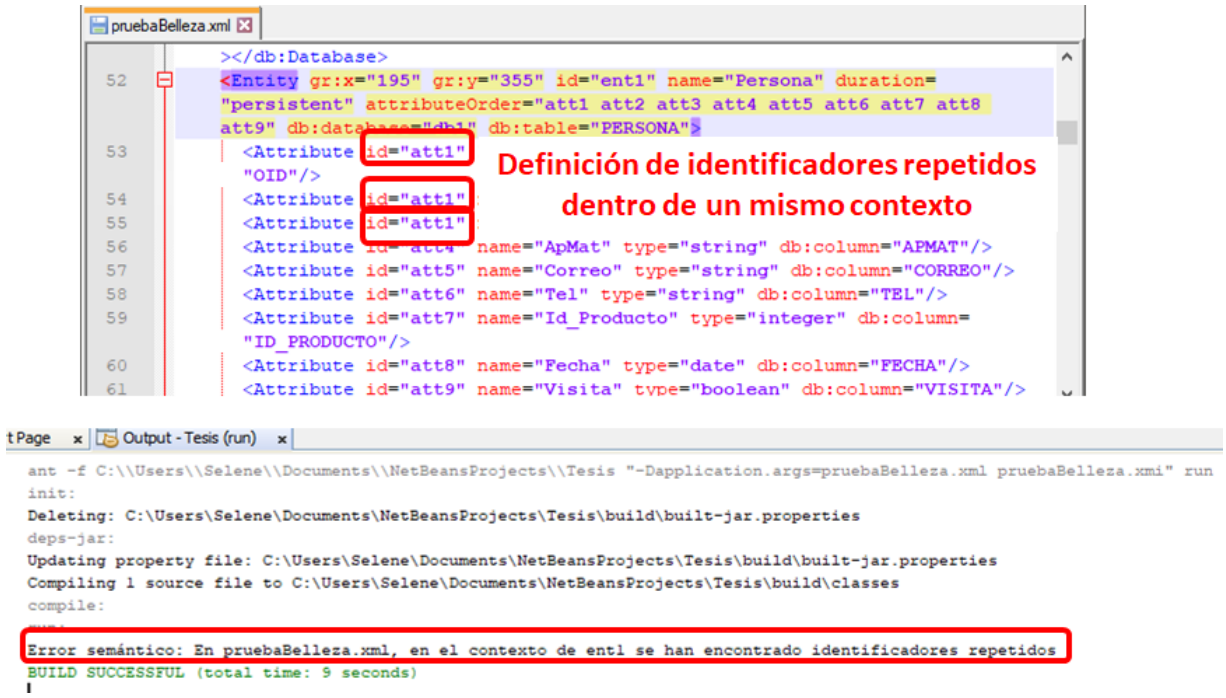

Figura 3.14 Resultado de la revisión semántica de contexto y alcance

En relación con la validación d, respecto a la comprobación de los flujos de navegación definidos en el modelo navegacional, se verifica que los elementos que representan un flujo de navegación hagan referencia a un identificador definido en la tabla de símbolos, no obstante, en caso de no encontrarse dicho identificador se reemplaza por el símbolo de numeral (#), y no es tomado como un error, este caso se maneja como una alerta semántica que se notifica al usuario pero que no detiene la fase de análisis. Los resultados de esta evaluación semántica se encuentran en la Figura 3.15.

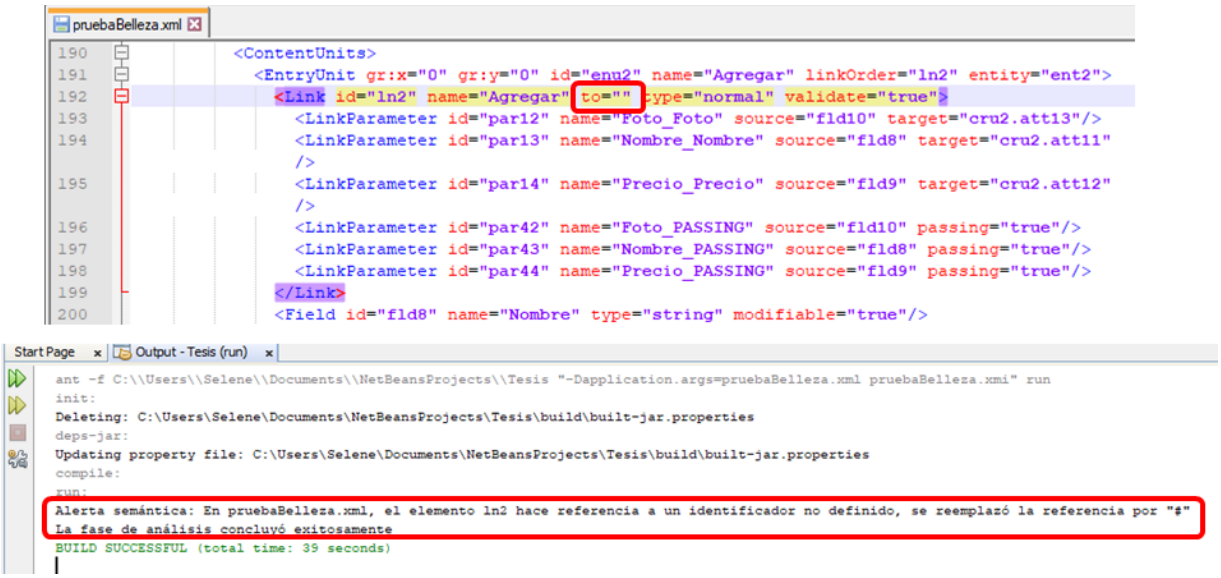

Figura 3.15 Resultado de la revisión semántica del flujo de navegación del modelo navegacional

Con la finalidad de asegurar que exista consistencia entre el modelo de dominio y el modelo de negocio se realizó la implementación de validación e, cuyo resultado se observa en la Figura 3.16, en donde al encontrarse definido el atributo Foto como tipo *blob* en la entidad Producto, en el modelo de dominio, y en la clase producto que corresponde a la entidad , en el modelo de negocio, se define el mismo atributo pero de tipo *integer*, se considera una inconsistencia semántica que corresponde a un mensaje de error.

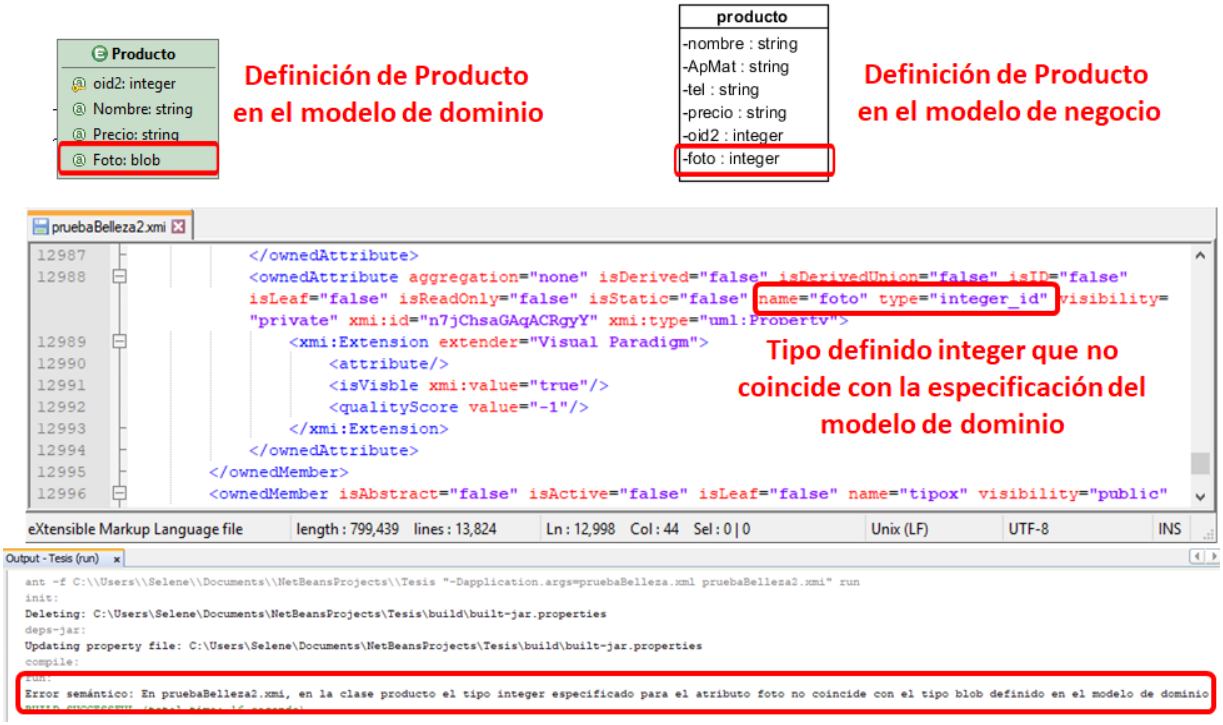

Figura 3.16 Resultado de la revisión semántica de consistencia entre el modelo de dominio y el modelo de negocio

3.2.1.10 Planificación

Acorde a la planificación estipulada en el proyecto de tesis, el ciclo finaliza con la estructura de la aplicación como lo muestra la Figura 3.17, donde la primera parte corresponde a los elementos ya descritos generados por ANTLR más componentes desarrollados para su implementación, y los elementos marcados en verde son los implementados en el siguiente ciclo.

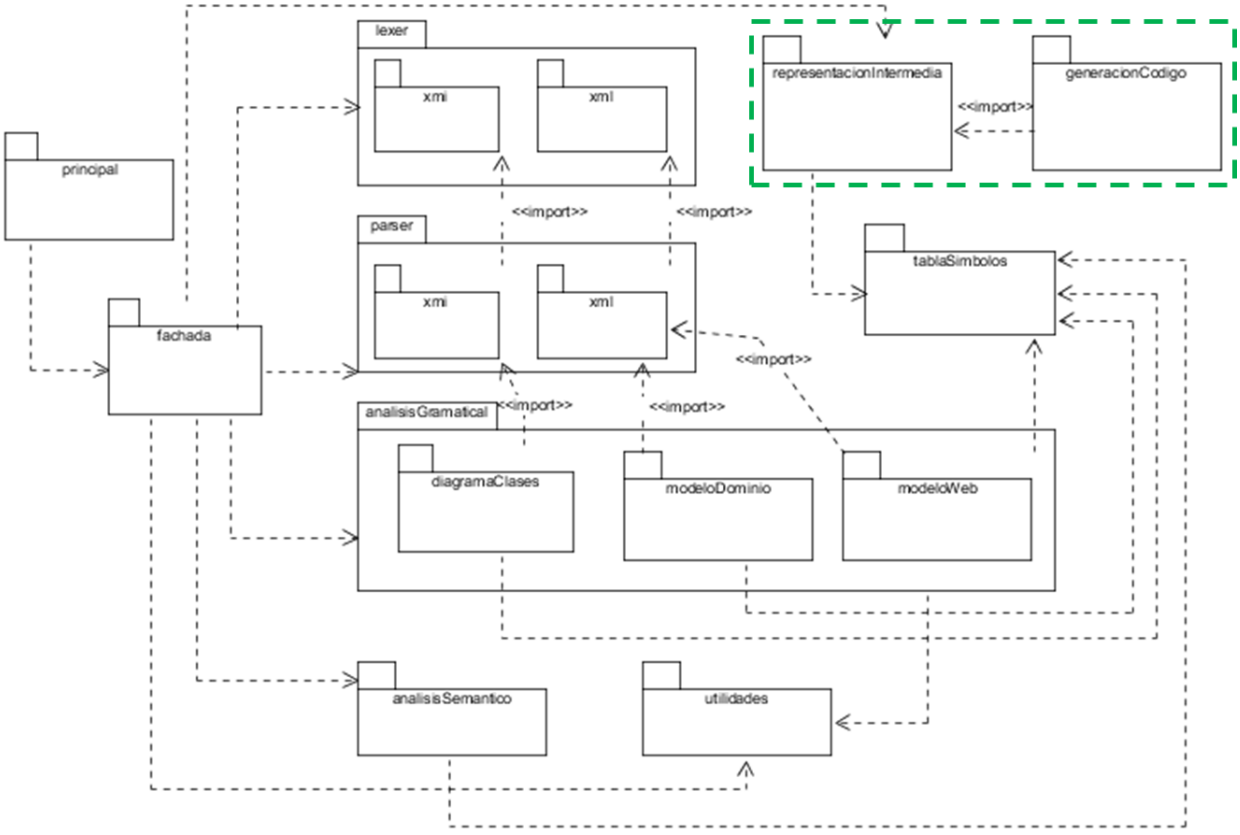

Figura 3.17 Estructura de la aplicación al finalizar el primer ciclo de la espiral

3.2.2 Segunda iteración

Corresponde a la segunda etapa del generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML, en la cual se lleva a cabo la especificación de la representación intermedia y la generación de código final para las tecnologías soportadas.

3.2.2.1 Representación intermedia

En el proceso de traducir un programa fuente a código objetivo, un compilador construye una o más representaciones intermedias que presentan una variedad de formas, un ejemplo de este concepto son los árboles de sintaxis, los cuales son una forma de representación intermedia y que son utilizados durante los análisis sintáctico y semántico [25].

En este proyecto se define una representación neutra de los elementos visuales relevantes de IFML en su representación de WebRatio que comparten características en común y que tienen una representación visual similar. Esto para contar con una abstracción, independiente del lenguaje del código fuente a obtener, dotada de características programables y libre de propiedades meramente de diagramación.

La Figura 3.18 se refiere a la estructura de la representación intermedia, se trata de un árbol de nodos con las especificaciones de los componentes encontrados en el modelo navegacional. Se observa como nodo raíz `IR_Application` de donde parten dos ramas principales, `IR_DomainModel` e `IR_WebModel`, correspondiendo al modelo de dominio y modelo navegacional de la aplicación.

La rama de `IR_DomainModel` despliega las propiedades de clases, atributos, métodos y sus respectivos parámetros; asimismo, su contraparte `IR_WebModel` engloba contenedores y componentes de la vista, los flujos de navegación, los *widgets* y sus reglas de validación.

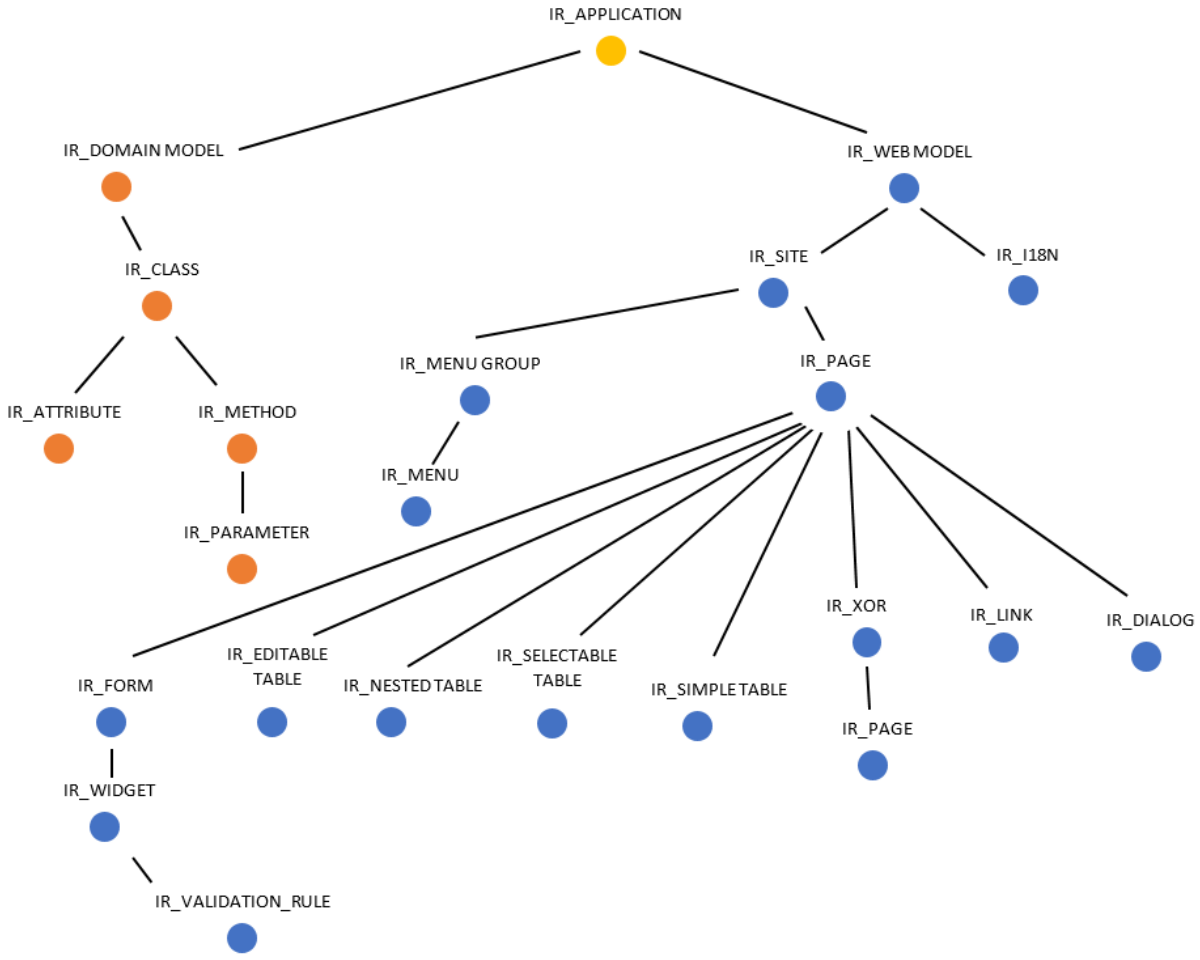

Figura 3.18 Árbol de la representación intermedia

La Figura 3.19 muestra un fragmento del diagrama de clases de la representación intermedia, donde, para cada uno de los tipos de nodo que conforman el árbol existe una clase con una herencia directa proveniente de la clase `IR_Node`, en adición a esto, el nodo puntualiza las propiedades relevantes del elemento del modelo que representa. Un ejemplo de esto es la clase `IR_Message`, que representa una unidad contenedora de un texto, cuyas propiedades particulares son el nombre del componente junto con el mensaje a desplegar. `IR_Node` es una clase abstracta que representa el comportamiento de cualquier nodo de representación intermedia, por ejemplo, el hecho de contar con un conjunto de subelementos o nodos hijos (`arrChildren`).

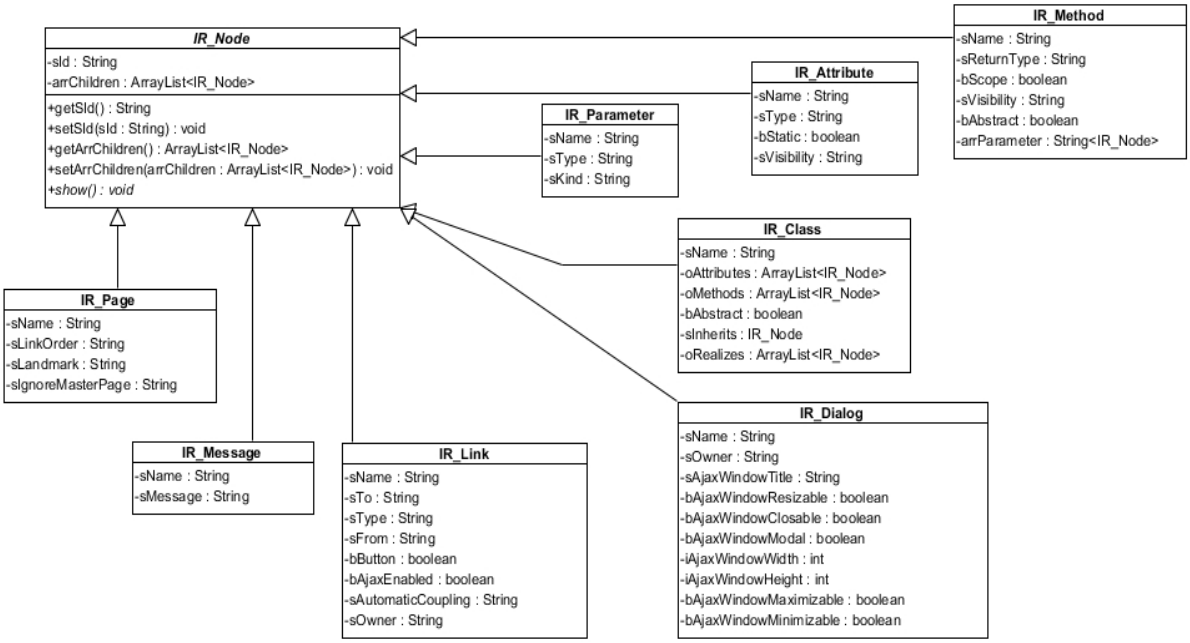

Figura 3.19 Fragmento del diagrama de clases de la representación intermedia

La representación intermedia creada corresponde a la estructura abstracta de una Aplicación Enriquecida de Internet independiente de alguna tecnología particular, de manera que, la información obtenida del modelo navegacional y de domino es transformada en nodos que son elementos con comportamiento común dentro cualquiera de las combinaciones de lenguajes a soportar, tal es el caso de clases, páginas, mensajes, formularios, campos, reglas de validación y flujos de navegación, que respectivamente son nodos `IR_Class`, `IR_Page`, `IR_Message`, `IR_Form`, `IR_Widget`, `IR_ValidationRule` e `IR_Link`. Pero, existen situaciones particulares en las que el componente del modelo no corresponde de forma directa con un nodo de la representación intermedia.

Un caso puntal de esta situación ocurre con el modelado de diálogos, los cuales, dentro del modelo se representan con el mismo componente que las páginas, dejando la definición de las propiedades específicas de un diálogo dentro del flujo de navegación que va hacia él. Esta situación origina el inconveniente de que las páginas y diálogos se encuentren como elementos separados y en un mismo nivel jerárquico, lo cual no es apropiado para este proyecto en

específico, a causa de que en las tecnologías implementadas los diálogos se encuentran dentro de las páginas y no como elementos separados.

También, se tienen componentes que conforme a sus propiedades corresponden a nodos distintos, tal es el caso del componente de WebRatio `List`, el cual, en primera instancia es un nodo *IR_SimpleTable*, pero, si en sus propiedades se indica como seleccionable (propiedad *checkable)* corresponde con un nodo *IR_CheckableList*.

Por último, se tienen nodos creados para contener información específica para las aplicaciones a generar, como `IR_I18N` con los datos de patrones de configuración, `IR_MenuGroup` e `IR_Menu`, que corresponden respectivamente a áreas con páginas anidadas, y a páginas definidas como punto de referencia (*landmark*) y el nodo `IR_Unssuported` para los elementos reconocidos del modelo que no tienen una representación en la tecnología a generar.

Para muestra de la definición intermedia particular de un modelo definido por el usuario, se tiene el proyecto Web `Ejemplo_RI`, el cual se integra por: a) la vista de sitio llamada `Ventas` (Figura 3.20) conformada por las páginas `Bienvenida`, `Ventas` e `Inventario`, y con un flujo de navegación; b) la vista del sitio `Empresa` (Figura 3.21) conformada por un total de nueve páginas: `Bienvenida`, `Ventas`, `Compras`, `Autos`, `Catalogo` (ubicada dentro del área `Catalogo de reportes`), y `Balances`, `Facturas`, `Activos`, `Pasivos`, agrupados dentro del área `Catalogo contable`, y, c) un modelo de dominio que define la entidad `Automovil` (Figura 3.22), junto con las entidades `User`, `Group` y `Model` propias de WebRatio y que no son tomadas en cuenta en el proceso de representación intermedia. La representación intemedia resultante se encuentra en la Figura 3.23.

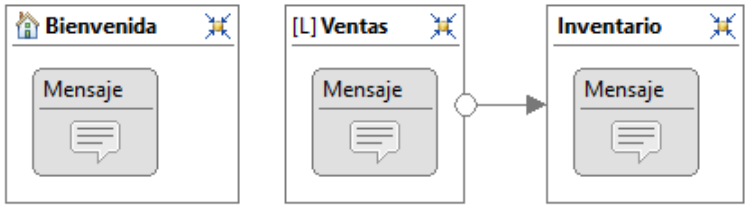

Figura 3.20 Vista del sitio Ventas en Ejemplo_RI

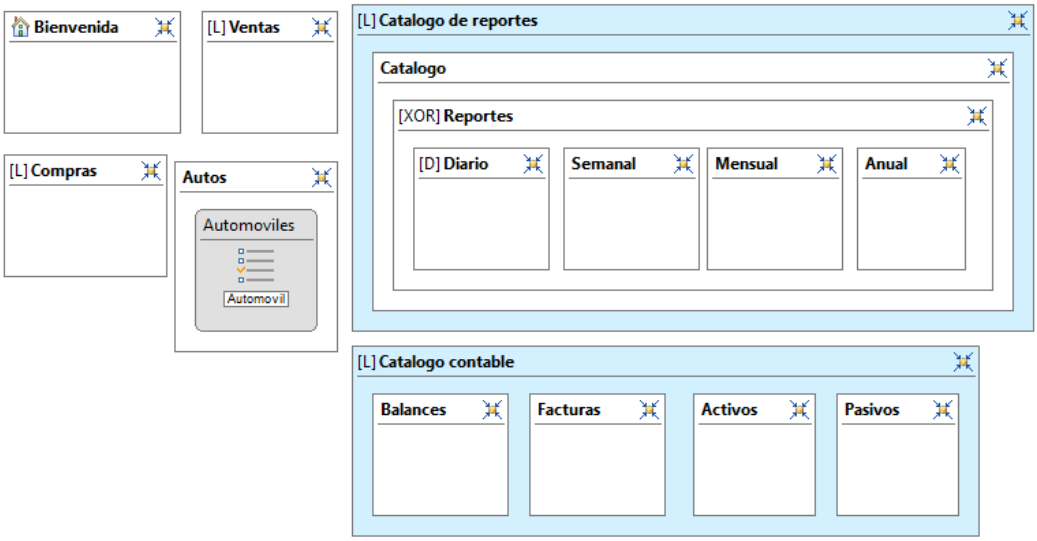

Figura 3.21 Vista del sitio Empresa en Ejemplo_RI

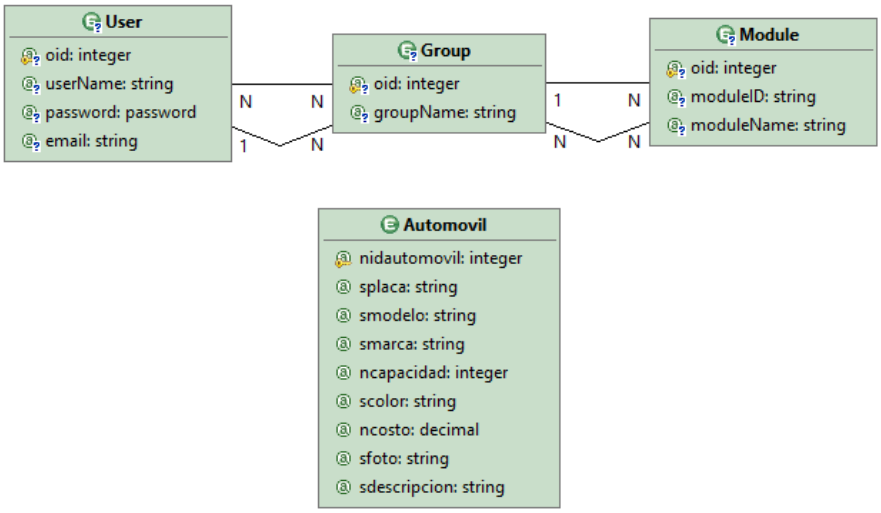

Figura 3.22 Modelo de dominio de Ejemplo_RI

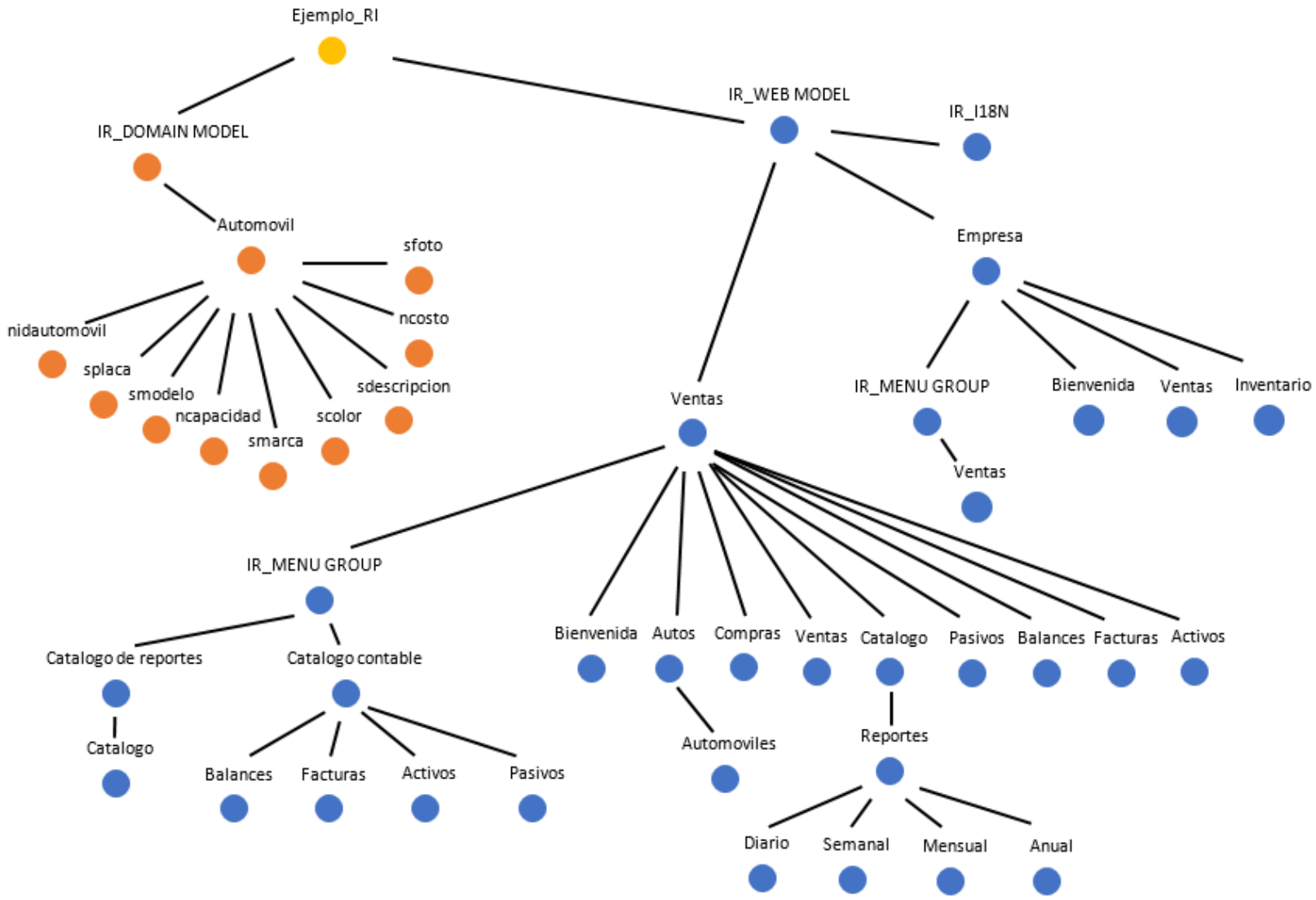

Figura 3.23 Árbol de la representación intermedia de Ejemplo_RI

La Figura 3.23 muestra la representación intermedia formada para Ejemplo_RI, se observa como el proyecto se divide en las ramas principales de `IR_DomainModel` e `IR_WebModel`. La rama del modelo de dominio tiene como hijo a la clase `Automovil`, la cual, tiene como hijos al conjunto de atributos que la describen. En la rama del modelo Web se tiene `IR_I18N` con información de internacionalización y configuración, los nodos `Ventas` y `Empresa` corresponden a las vistas del sitio de la aplicación, y el nodo `IR_MenuGroup` tiene por hijos a `IR_MenuGroup` e `IR_Menu`, que son las áreas y páginas modeladas como punto de referencia.

3.2.2.2 Generación de código

La fase final dentro del proceso de traducción corresponde a la generación del código, en la cual se toma como entrada una representación intermedia del programa fuente y se produce la salida en un programa objeto equivalente [25].

Esta etapa se realizó considerando las pautas del patrón de diseño *Abstract Factory*, ya que es idóneo para la creación de familias de objetos relacionados o dependientes, ya que encapsula la creación de objetos de manera separada [54]. La estructura del patrón se integra por: a) una fábrica abstracta que define la interfaz de las fábricas concretas, b) fábricas concretas que implementan la fábrica abstracta y sobrescriben sus métodos para devolver la instancia de un producto concreto, c) productos abstractos que definen las interfaces para la familia de productos genéricos, y, c) productos concretos que implementan la interfaz de los productos abstractos.

La figura 3.24 muestra las clases `Abstract_Application_CodeGenerator` y `Abstract_Enviroment_CodeGenerator` con las firmas de los métodos a ser implementados por `PHP_Application_CodeGenerator` y `JSF_Application_CodeGenerator` y `PHP_Enviroment_CodeGenerator` junto `JSF_Enviroment_CodeGenerator`, las cuales, son las clases concretas con la capacidad de entregar los archivos correspondientes a clases, páginas, controladores, en una tecnología particular junto con los archivos JSON que simulan el repositorio de información.

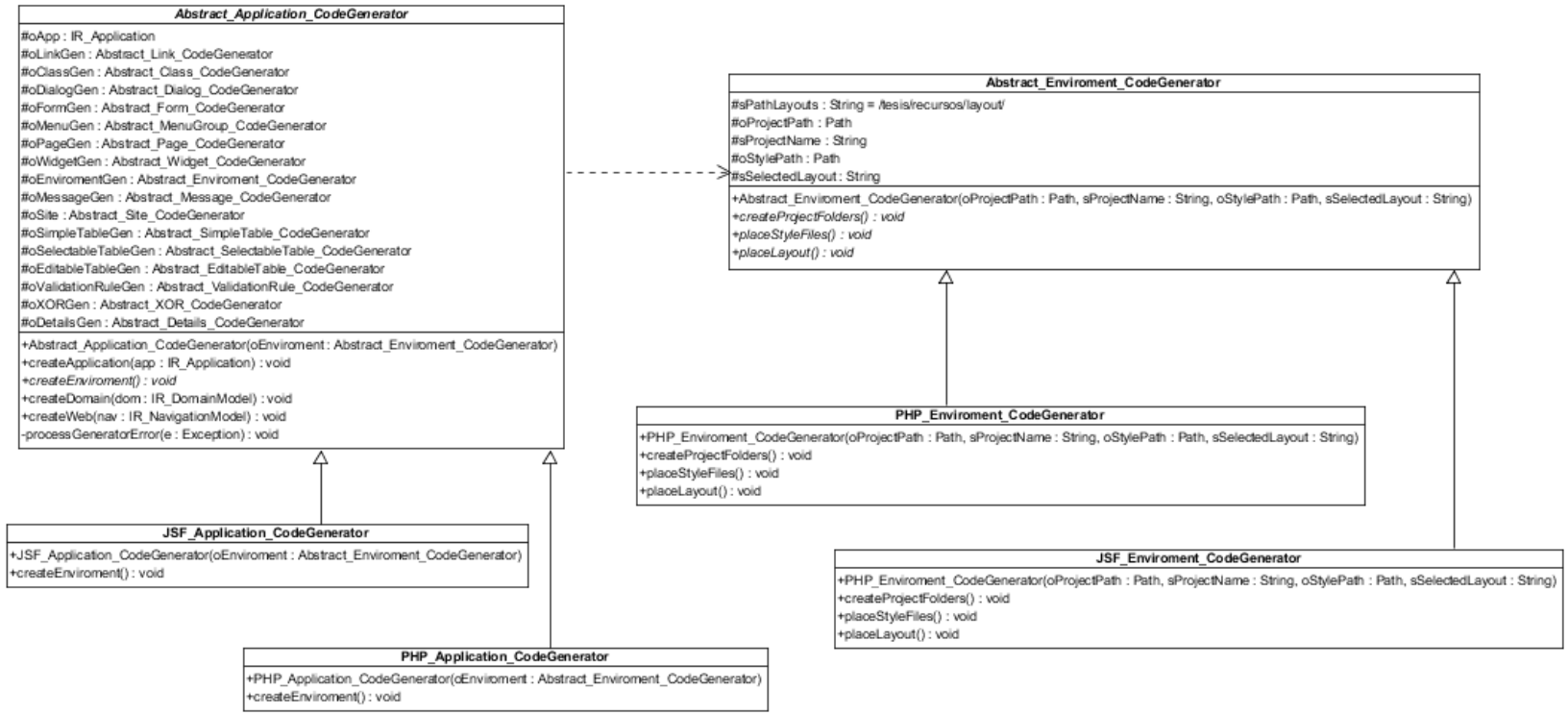

Figura 3.24 Fragmento del conjunto de clases abstractas para la generación de código

La clase encargada de recorrer el árbol de la representación intermedia es `Abstract_Application_CodeGenerator` con el método `createApplication`, mostrado en el Listado 3.6, en donde se observa que recibe como parámetro el nodo raíz de la representación intermedia de la aplicación (línea 01), para después crear el ambiente específico de la tecnología (línea 04), recorrer el contenido del nodo con la representación intermedia de la aplicación (línea 05), y ejecutar los métodos para la creación del modelo de dominio y del modelo navegacional (líneas 07 y 09).

Listado 3.6 Método `createApplication`

```
01. public void createApplication(IR_Application app){
02.     this.oApp = app;
03.     try{
04.         createEnviroment();
05.         for (IR_Node node : this.oApp.getChildren()){
06.              if (node.getClass().equals(IR_DomainModel.class))
07.                   createDomain((IR_DomainModel)node);
08.              else if (node.getClass().equals(IR_NavigationModel.class))
09.                   createWeb((IR_NavigationModel)node);
10.              else
11.                  throw new Exception ("Error, Incorrect node type in the
                     application " + node.getClass());
12.           }
13.     }catch(Exception e){
14.           processGeneratorError(e);
15.     }
16. }
```

3.2.2.3 Planificación

El ciclo finaliza con la estructura de la aplicación como lo muestra la Figura 3.25, en la cual se observa la adición de los paquetes de la representación intermedia junto con el paquete que contiene las clases abstractas de los elementos a generar.

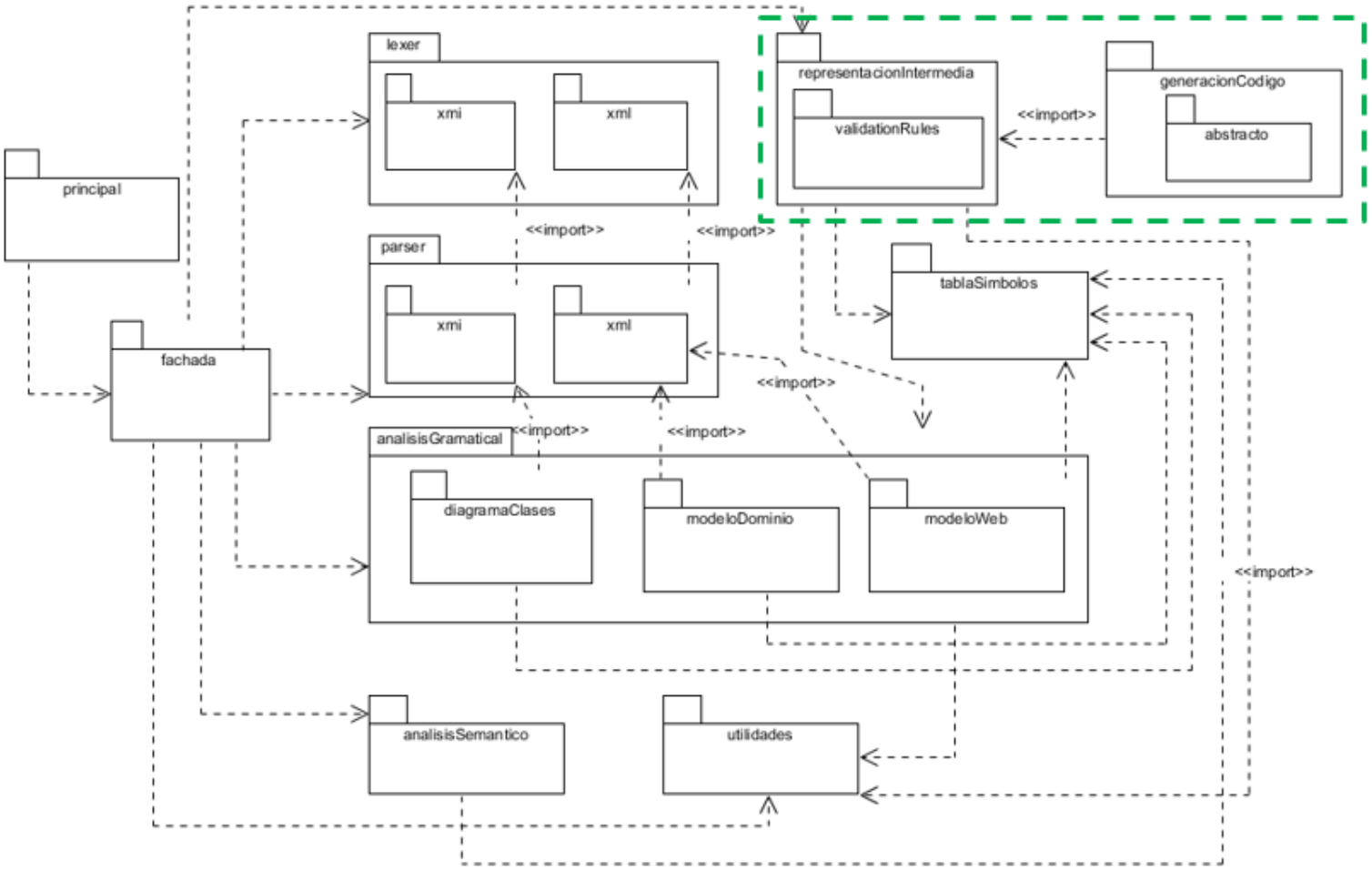

Figura 3.25 Estructura de la aplicación al finalizar el segundo ciclo de la espiral

3.2.3 Tercera iteración

Corresponde a la tercera etapa del generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML, en la cual se lleva a cabo la especificación de la generación de código propia de la combinación de tecnologías PHP con jQuery.

3.2.3.1 Generación de código para la combinación de PHP con jQuery

Se crearon las clases que entregan código objetivo para la combinación de tecnologías PHP con jQuery, las cuales implementan las clases abstractas definidas en la segunda iteración.

Se entrega un directorio con los siguientes elementos que conforman una RIA generada para esta tecnología:

- El archivo `index.php` que redirecciona a la página marcada como inicial.
- Archivos PHP de acuerdo a la distribución seleccionada.
- Un archivo con el menú de navegación de la aplicación.
- Los archivos PHP correspondientes a las páginas definidas en el modelo navegacional.
- El directorio `css` con los elementos de estilo proporcionados y el CSS de la distribución seleccionada.
- El directorio `ctrlPHP` con los archivos PHP que controlan el comportamiento de las páginas.
- El directorio `js` con los archivos JavaScript para la manipulación del contenido de las páginas.
- El directorio `jsLib` con la biblioteca jQuery junto con las bibliotecas para el funcionamiento de diálogos, menús y campos especiales de captura de fecha y hora.
- El directorio `media` para archivos multimedia.
- El directorio `model` con las clases PHP que representan cada entidad del modelo de dominio, junto con un archivo JSON por entidad, con datos de prueba.
- El archivo `Log.txt` con la información de los procesos realizados durante la generación de la aplicación (este documento se crea en todas las tecnologías soportadas por el generador).

Las siguientes imágenes muestran fragmentos del contenido de los archivos generados a partir de la información definida en la representación intermedia.

La Figura 3.26 muestra como la página incluye los elementos de menú, cabecera y pie de página, junto con el archivo JavaScript que controla su contenido visualizado.

Autos.php

```
<?php
$page_title = 'Autos';
include_once("header.php");
include_once("menu.php");
?>
            <script type="text/javascript" src="js/Autos.js"></script>
            <main class="mainContent">

            <div id="divMciu1" name="divMciu1">
                <p class="ui-widget-header" style="width:100%;"><br/>Automoviles<br/><br/></p>
                <form id="frmTableGralMciu1" name="frmTableGralMciu1"  style="display:block" method="POST"
                onsubmit="return false;">
                    <table border="1" id="tblMciu1" class="display">
                        <thead>
                            <tr>
                                <th></th>
                                <th>splaca</th>
                                <th>smodelo</th>
                                <th>smarca</th>
                                <th>ncosto</th>
                            </tr>
                        </thead>
                        <tbody id="tblBodyMciu1"></tbody>
                    </table>
                    <div class="ui-widget-header" style="width:100% " align="center">
                    </div>
                </form>
                <script> drawTableMciu1(); </script>
            </div>
            </main>
            <aside class="otherContent"></aside>
        </div>
    </section>
<?php
include_once("footer.php");
?>
```

Figura 3.26 Código generado para Autos.php

La Figura 3.27 muestra el archivo PHP de control de la entidad Automovil, se observa la inclusión del elemento del modelo con el que trabaja, así como el proceso de recuperar la información del modelo, y crea una cadena con formato JSON con la información a ser utilizada por los archivos JavaScript que despliegan contenido en la vista.

ctrlEntity_Automovil.php

```php
<?php
include_once("../model/Automovil.php");
$sErr="";
$arrData = null;
$oObject = new Automovil();
$sCadJson="";

    $arrData = $oObject->findAll();

    if ($sErr == "" && $arrData != null){
        $sCadJson =
        '{
            "successful":true,
            "sSit": "ok",
            "arrData":[';
        foreach( $arrData as $oObj){
            $sCadJson = $sCadJson.'{
                    "nidautomovil":"'.$oObj->getNidautomovil().'",
                    "splaca":"'.$oObj->getSplaca().'",
                    "smodelo":"'.$oObj->getSmodelo().'",
                    "smarca":"'.$oObj->getSmarca().'",
                    "ncapacidad":"'.$oObj->getNcapacidad().'",
                    "scolor":"'.$oObj->getScolor().'",
                    "ncosto":"'.$oObj->getNcosto().'",
                    "sfoto":"'.$oObj->getSfoto().'",
                    "sdescripcion":"'.$oObj->getSdescripcion().'",
                    "automovil":"'.$oObj->getAutomovil().'"
            },';
        }
        $sCadJson = substr($sCadJson,0, strlen($sCadJson)-1);
        $sCadJson = $sCadJson.'
            ]
        }';
    }
    else{
        $sCadJson =
        '{
            "successful":false,
            "sSit": "'.$sErr.'",
            "arrData": []
        }';
    }
    header('Content-type: application/json');
    echo $sCadJson;
?>
```

Figura 3.27 Código generado ctrlEntity_Automovil.php

Un ejemplo del código que realiza el despliegue de información se muestra en la Figura 3.28, en donde se tiene la función drawTableMciu1, responsable de, partiendo de la cadena JSON entregada por el controlador, dibujar el contenido del archivo JSON en forma de una tabla.

Autos.js

```
$(document).ready(function(){
});
function drawTableMciu1(){
var sURL = 'ctrlPhp/ctrlEntity_Automovil.php';
var oBodyTbl = $('#tblBodyMciu1');
var oLine;
var oCell0,oCell1,oCell2,oCell3,oCell4;
var i=0;
    var llamada=$.post(
        sURL,
        {
        },
        function(oData) {
            if(oData.successful==true){
                oBodyTbl.empty();
                for (i=0; i< oData.arrData.length; i++){
                    oLine = $('<tr/>');
                    oCell0 = $('<td/>');
                    oCell1 = $('<td/>');
                    oCell2 = $('<td/>');
                    oCell3 = $('<td/>');
                    oCell4 = $('<td/>');
                    oCtrlCheckbox = $('<input/>').attr({
                        type: 'checkbox',
                        id: 'checkboxMciu1' + i ,
                        name:'indexMciu1',
                        value: i,
                    });
                    oCell0.append(oCtrlCheckbox);
                    oCell1.text(oData.arrData[i].splaca);
                    oCell2.text(oData.arrData[i].smodelo);
                    oCell3.text(oData.arrData[i].smarca);
                    oCell4.text(oData.arrData[i].ncosto);
                    oLine.append(oCell0,oCell1,oCell2,oCell3,oCell4);
                    oBodyTbl.append(oLine);
                };
                var table = $('#tblMciu1').DataTable({ paging: false, ordering:  true,
                searching: false, info:false});
                table.order(  [ 2 , 'desc' ] );
                table.draw();
            }else{
                sError = sSit;
                alert(sError);
                }
            });
}
```

Figura 3.28 Código generado para Autos.js

Las clases generadas en PHP se integran por atributos con sus respectivos métodos set y get, y un constructor (Figura 3.29), en conjunto con métodos especializados para la interacción con el repositorio de información simulado con los archivos JSON (Figura 3.30).

Es importante enfatizar que el desarrollador deberá realizar las modificaciones pertinentes para la comunicación con el repositorio de información de su elección.

Automovil.php

```php
<?php
class Automovil {

    private $nidautomovil;
    private $splaca;
    private $smodelo;
    private $smarca;
    private $ncapacidad;
    private $scolor;
    private $ncosto;
    private $sfoto;
    private $sdescripcion;
    private $automovil;

    public function __construct($nidautomovil=null,$splaca=null,$smodelo=null,$smarca=
    null,$ncapacidad=null,$scolor=null,$ncosto=null,$sfoto=null,$sdescripcion=null,
    $automovil=null){
        $this->nidautomovil = $nidautomovil;
        $this->splaca = $splaca;
        $this->smodelo = $smodelo;
        $this->smarca = $smarca;
        $this->ncapacidad = $ncapacidad;
        $this->scolor = $scolor;
        $this->ncosto = $ncosto;
        $this->sfoto = $sfoto;
        $this->sdescripcion = $sdescripcion;
        $this->automovil = $automovil;
    }

    public  function getNidautomovil (){
         return $this->nidautomovil;
    }

    public  function setNidautomovil($value){
        $this->nidautomovil = $value;
    }
```

Figura 3.29 Código generado para Automovil.php

Automovil.php

```php
    public  function findAll() {
        //THIS METHOD MUST BE MODIFIED BY THE DEVELOPER TO
        //PERFORM THE ACTION ON THEIR INFORMATION REPOSITORY
        $str_data = file_get_contents("../model/Automovil.json");
        $data = json_decode($str_data,true);
        $arrData = array();
        foreach ($data as $obj) {
            array_push($arrData, new Automovil($obj["nidautomovil"],$obj["splaca"],
            $obj["smodelo"],$obj["smarca"],$obj["ncapacidad"],$obj["scolor"],$obj[
            "ncosto"],$obj["sfoto"],$obj["sdescripcion"],$obj["automovil"]));
        }
        return $arrData;
    }
```

Figura 3.30 Código generado para la interacción con el repositorio de información

En la figura 3.31 se muestra un ejemplo del archivo JSON que simula el repositorio de información. Dependiendo del tipo del atributo de la clase al cual está ligada cada una de las claves, se asigna un valor de prueba correspondiente, ya sea numérico, alfanumérico o booleano.

Figura 3.31 Archivo JSON generado para la clase Automovil.php

3.2.3.2 Planificación

Al término de la iteración el generador es capaz de entregar Aplicaciones Enriquecidas de Internet para la combinación de tecnologías PHP con jQuery. En este punto del proceso solo resta la generación de código JSF con PrimeFaces.

La estructura de paquetes final del proyecto al concluirse la iteración final es la mostrada en la Figura 3.32.

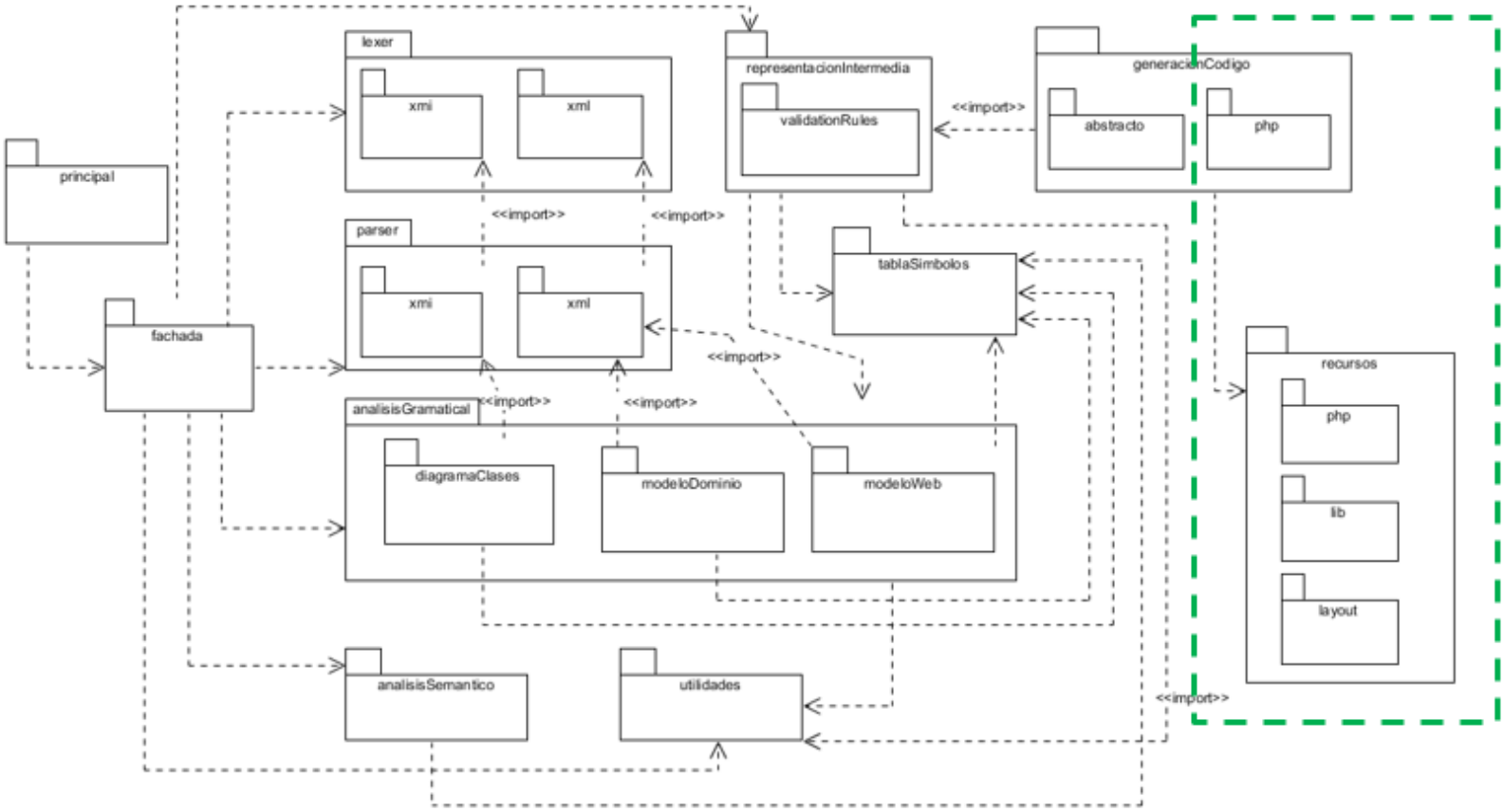

Figura 3.32 Estructura de la aplicación al finalizar el tercer ciclo de la espiral

3.2.4 Cuarta iteración

Concierne a la última etapa del proceso de desarrollo del generador de Aplicaciones Enriquecidas de Internet, en la que se implementa la generación de código destino en JSF con PrimeFaces.

3.2.4.1 Generación de código JSF y PrimeFaces

Se crearon las clases que entregan código objetivo para la combinación de tecnologías JSF con PrimeFaces, las cuales implementan las clases abstractas definidas en el apartado 3.2.2.2 Generación de código.

Se entrega un proyecto Web de NetBeans, cuyo directorio se integra con los siguientes elementos:

- Directorio `lib` con los archivos JAR indispensables para el funcionamiento de la aplicación.
- El directorio `templates` contiene la plantilla de la aplicación, y su contenido depende de la distribución seleccionada.
- El directorio `facelets`, con los directorios `lib` y `templates`, junto a los archivos XML del proyecto Web para el trabajo con el servidor Glassfish.
- El directorio `css` destinado a tener la distribución de contenido indicada por el usuario.
- El directorio `media` para los archivos multimedia de la aplicación.
- El directorio `resources` con los directorios `css` y `media`.
- El directorio `web` con los archivos XHTML que corresponden a las páginas definidas en el modelo junto con los directorios `resources` y `WEB-INF`.
- El directorio `jb` con los archivos JavaBean para el despliegue de información en las |páginas XHTML, además de la validación de campos y procesamiento de formularios.
- El directorio `model` con los archivos Java que representan a cada una de las entidades del modelo de dominio junto a su correspondiente JSON.

La Figura 3.33 exhibe el código generado para la página `Autos.xhtml`, la cual contiene una tabla enlazada al JavaBean Autos_JB. Se observa el uso de etiquetas propias de PrimeFaces para mostrar datos del elemento tabla modelado.

Output | Autos.xhtml

Source | History

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <ui:composition
5          xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
6          xmlns:h="http://xmlns.jcp.org/jsf/html"
7          xmlns:f="http://xmlns.jcp.org/jsf/core"
8          xmlns:p="http://primefaces.org/ui"
9          template="./WEB-INF/facelets/templates/template.xhtml">
10         <ui:define name="title">Autos</ui:define>
11         <ui:define name="mainContent">
12         <h:form>
13             <p:panel id="Mciul" header="Automoviles" style="margin-bottom:20px">
14                     <h:panelGrid columns="1" cellpadding="10">
15                             <p:dataTable id="dataTableMciul" var="item"
16                                 value="#{Autos_JB.dataMciul}"
17                                 selection="#{Autos_JB.selectMciul}"
18                                 rowKey="#{item.index}">
19                                     <p:column selectionMode="multiple" style="width:16px;text-align:center"/>
20                                     <p:column headerText="splaca">
21                                             <h:outputText value="#{item.splaca}" />
22                                     </p:column>
23                                     <p:column headerText="smodelo">
24                                             <h:outputText value="#{item.smodelo}" />
25                                     </p:column>
26                                     <p:column headerText="smarca">
27                                             <h:outputText value="#{item.smarca}" />
28                                     </p:column>
29                                     <p:column headerText="ncosto">
30                                             <h:outputText value="#{item.ncosto}" />
31                                     </p:column>
32             </p:dataTable>
33             </h:panelGrid>
34             </p:panel>
35         </h:form>
36         </ui:define>
37 </ui:composition>
```

Figura 3.33 Código generado para Autos.xhtml

La Figura 3.34 contiene un fragmento del código perteneciente al JavaBean `Autos_JB`. El cual cumple con las directrices establecidas por la especificación de JavaBeans de Sun Microsystems.

Output × Autos_JB.java ×

Source History

```
1  package jb;
2
3  import java.io.Serializable;
4  import javax.annotation.PostConstruct;
5  import javax.faces.view.ViewScoped;
6  import javax.inject.Named;
7  import java.util.ArrayList;
   import javax.faces.application.FacesMessage;
   import javax.faces.context.FacesContext;
   import javax.faces.event.ActionEvent;
11 import model.Automovil;
12
13 @Named("Autos_JB")
14 @ViewScoped
15 public class Autos_JB implements Serializable {
16
17         private ArrayList<Automovil> dataMciul;
18         ArrayList<Automovil> selectMciul;
19
20         @PostConstruct
21         public void initialize() {
22
23                 try{
24                         dataMciul = new Automovil().findAll();
25                 }
26                 catch(Exception e){
27
                           e.printStackTrace();
29
30                 }
31         }
32
33         public ArrayList<Automovil> getDataMciul() {
34                 return dataMciul;
35         }
36
37         public void setDataMciul(ArrayList<Automovil> dataMciul) {
38                 this.dataMciul = dataMciul;
39         }
40
41         public ArrayList<Automovil> getSelectMciul() {
42                 return selectMciul;
43         }
44
45         public void setSelectMciul(ArrayList<Automovil> selectMciul) {
46                 this.selectMciul = selectMciul;
47         }
48
49         public String validateAndNavigate() {
50                 // ADD VALIDATIONS FOR YOUR APPLICATION
51                 return "Autos";
52         }
```

Figura 3.34 Código creado para Autos_JB.java

El archivo `Automovil.java,` representa una entidad del modelo de dominio implementada con una clase (Figura3.35), y al igual que en el caso de la tecnología PHP, cuenta con métodos para recuperar el contenido del repositorio de información simulado.

Output × Automovil.java ×

Source History

```
1  package model;
2
3  import java.io.FileNotFoundException;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.io.Serializable;
8  import java.util.ArrayList;
9  import javax.faces.bean.ApplicationScoped;
10 import org.json.simple.JSONArray;
11 import org.json.simple.JSONObject;
12 import org.json.simple.parser.JSONParser;
13 import org.json.simple.parser.ParseException;
14
15 @ApplicationScoped
16 public class Automovil implements Serializable{
17
18         private int index;
19         private int nidautomovil;
20         private String splaca;
21         private String smodelo;
22         private String smarca;
23         private int ncapacidad;
24         private String scolor;
25         private float ncosto;
26         private String sfoto;
27         private String sdescripcion;
28         private String automovil;
29
30         public Automovil(){}
31
32         public Automovil(int nidautomovil,String splaca,String smodelo,String smarca,
33                 int ncapacidad,String scolor,float ncosto, String sfoto,String sdescripcion,String automovil){
34                 this.nidautomovil = nidautomovil;
35                 this.splaca = splaca;
36                 this.smodelo = smodelo;
37                 this.smarca = smarca;
38                 this.ncapacidad = ncapacidad;
39                 this.scolor = scolor;
40                 this.ncosto = ncosto;
41                 this.sfoto = sfoto;
42                 this.sdescripcion = sdescripcion;
43                 this.automovil = automovil;
44         }
```

Figura 3.35 Código generado para Automovil.java

La Figura 3.36 resalta los últimos elementos añadidos al generador aplicaciones al concluir su proceso de desarrollo.

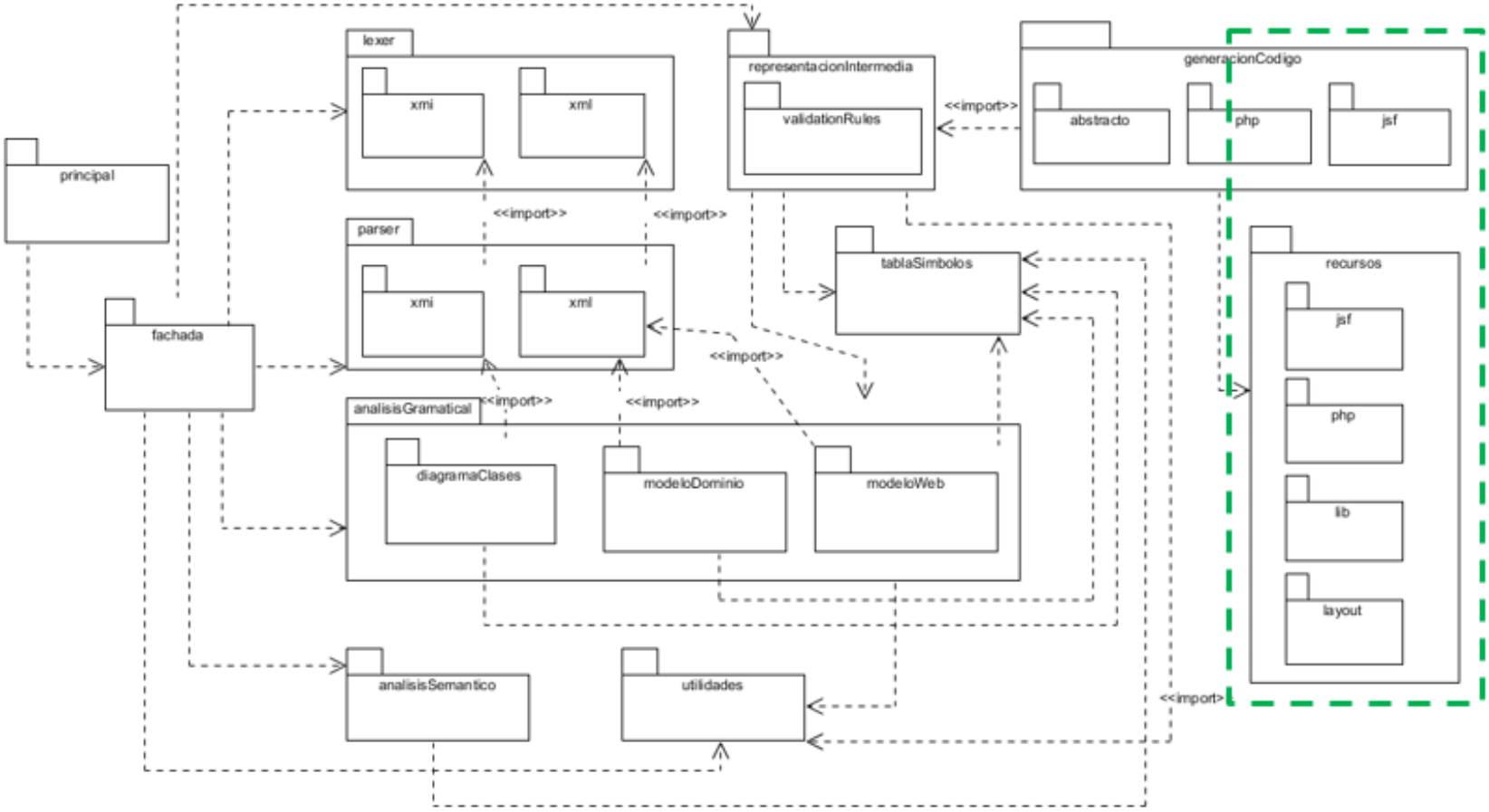

Figura 3.36 Estructura de la aplicación al finalizar el cuarto ciclo de la espiral

Capítulo 4. Resultados

Los resultados mostrados en este capítulo se dividen en tres ejes principales: a) comprobación de la funcionalidad del generador por medio de la formulación y realización de un caso de estudio; b) verificación de la robustez de la arquitectura a través de la generación de código final en AngularJS (marco de trabajo de JavaScript) y c) exposición de una posible aportación en el área educativa, específicamente en asignaturas enfocadas al modelado de aplicaciones.

El apartado del caso de estudio muestra la herramienta en operación y el funcionamiento de las aplicaciones entregadas en cada una de las tecnologías disponibles, con lo cual es posible notar cómo un desarrollador a partir de sus modelos obtiene un código útil funcional. De igual forma, se presenta la capacidad del generador para incorporar nuevas tecnologías y se muestra su aportación en el área educativa.

Es pertinente indicar que el objetivo del presente proyecto de tesis corresponde al desarrollo de un generador de Aplicaciones Enriquecidas de Internet que entrega código final: a) de acuerdo al patrón arquitectónico MVC; b) considerando la especificación del estilo visual por parte del usuario; c) con diversas alternativas de distribución de contenido; d) que sigue las especificaciones de los modelos de negocio, dominio y navegacional que provee el usuario, y e) en una de dos combinaciones de tecnologías, PHP con jQuery y JSF con PrimeFaces.

La herramienta desarrollada entrega aplicaciones que no se encuentran condicionadas para el trabajo con un repositorio de información específico, ya que no proporciona código SQL para la conexión con una base de datos, en su lugar entrega archivos JSON, con información de prueba, correspondientes a cada una de las entidades del modelo de dominio y suministra métodos para la recuperación de la información contenida en dichos archivos. El usuario obtiene una RIA que cumple con los elementos navegacionales y de contenido de la interfaz señalados en el modelo navegacional.

4.1 Planteamiento del caso de estudio

Una agencia de renta de autos para viajeros nacionales necesita una Aplicación Enriquecida de Internet para poner a disposición de sus clientes su catálogo de automóviles y permitirles agendar sus reservaciones, además de esto se requiere que mediante la aplicación los clientes sean quienes registran sus datos, y que se lleve a cabo la gestión de la información de rentas, automóviles y de los daños que puede presentar un automóvil al momento de ser devuelto.

La Figura 4.1 muestra el modelo de negocio correspondiente al caso de estudio mediante un diagrama de clases, en donde se representan las clases con los atributos y métodos necesarios para captar la lógica del negocio. Un ejemplo claro de esto se encuentra en la clase Renta, la cual tiene atributos que corresponden a la reservación del automóvil, a su recolección y devolución. Y también se aprecia la función `calcular renta`, la cual recibe como parámetro un número entero que representa un descuento y que devuelve el valor calculado para la renta del automóvil.

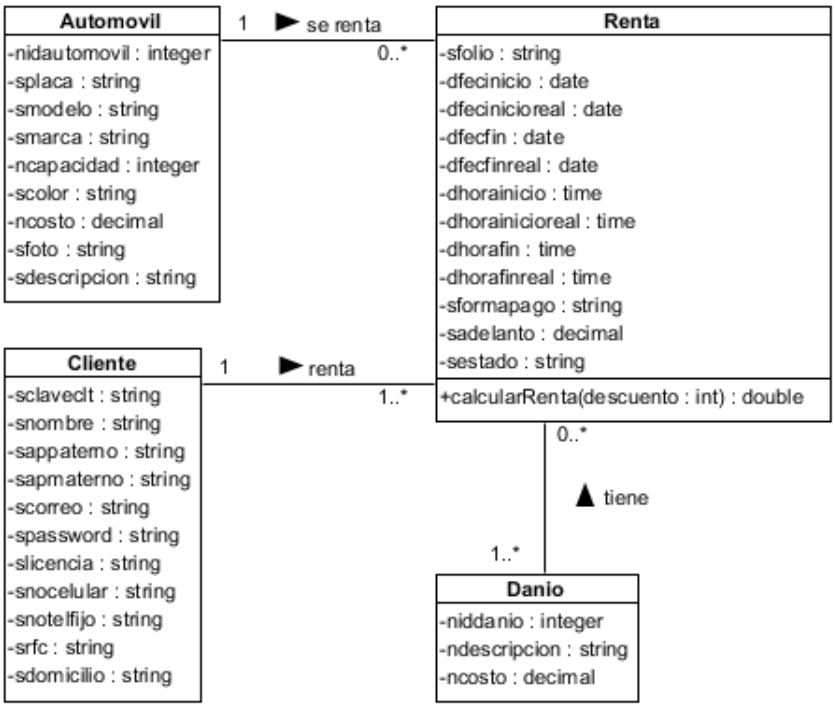

Figura 4.1 Modelo de negocio del caso de estudio

En el listado 4.1 se coloca un fragmento del código XMI generado por la herramienta de modelado que describe a un atributo, líneas 3 a la 9, y a la operación para el cálculo de la renta, líneas 11 a la 19.

Listado 4.1 Fragmento del código XMI correspondiente al modelo de negocio, generado por Visual Paradigm, del caso de estudio

```
01. <ownedMember isAbstract="false" isActive="false" isLeaf="false" name="Renta"
     visibility="public" xmi:id="pbIzd2aGAqACQAro" xmi:type="uml:Class">
02.    <!– More content -->
03.    <ownedAttribute aggregation="none" isDerived="false"
        isDerivedUnion="false" isID="false" isLeaf="false" isReadOnly="false"
        isStatic="false" name="sfolio" type="string_id" visibility="private"
        xmi:id="vvUrd2aGAqACQAuB" xmi:type="uml:Property">
04.       <xmi:Extension extender="Visual Paradigm">
05.          <attribute/>
06.          <isVisble xmi:value="true"/>
07.          <qualityScore value="-1"/>
08.       </xmi:Extension>
09.    </ownedAttribute>
10.    <!– More content -->
11.    <ownedOperation isAbstract="false" isLeaf="false" isOrdered="false"
        isQuery="false" isStatic="false" isUnique="true" name="calcularRenta"
        visibility="public" xmi:id="LWR3_2aGAqACQQeR" xmi:type="uml:Operation">
12.       <ownedParameter kind="return" type="double_id"
           xmi:id="LWR3_2aGAqACQQeR_return" xmi:type="uml:Parameter"/>
13.       <ownedParameter kind="inout" name="descuento" type="int_id"
           xmi:id="2VrGgOaGAqACQQeC" xmi:type="uml:Parameter">
14.          <xmi:Extension extender="Visual Paradigm">
15.             <qualityScore value="-1"/>
16.          </xmi:Extension>
17.       </ownedParameter>
18.    <!– More content -->
19.    </ownedOperation>
20. </ownedMember>
```

En la Figura 4.2 se observa el modelo de dominio, que requiere IFML y que es equivalente al diagrama de clases, en lo que a clases y atributos se refiere.

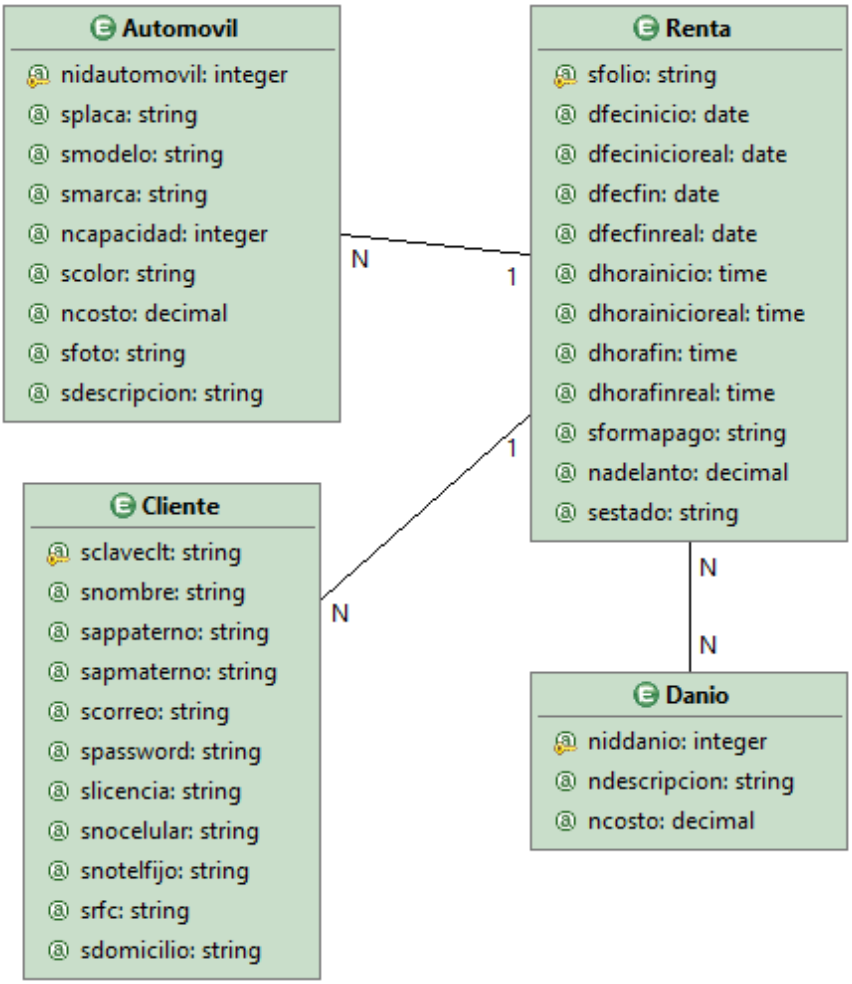

Figura 4.2 Modelo de dominio del caso de estudio

En el Listado 4.2 se tiene un fragmento del código XML correspondiente a la descripción del modelo de negocio expuesto en la Figura 4.2. En donde las líneas 1 y 14 se refieren a las etiquetas de apertura y cierre para la especificación de la entidad Renta, y las líneas 2 a la 13 declaran la especificación de algunos de los atributos que conforman a la entidad.

Listado 4.2 Fragmento del código XMI, generado por WebRatio Web Platform, correspondiente al modelo de dominio del caso de estudio

```
01. <Entity id="ent5" name="Renta" db:database="db1" duration="persistent"
     db:table="renta" gr:x="1175" gr:y="40">
02.    <Attribute id="att26" name="sfolio" db:column="sfolio" type="string"
        key="true"/>
03.    <Attribute id="att27" name="dfecinicio" db:column="dfecinicio" type="date"
        key="false"/>
04.    <Attribute id="att28" name="dfecinicioreal" db:column="dfecinicioreal"
        type="date" key="false"/>
05.    <Attribute id="att29" name="dfecfin" db:column="dfecfin" type="date"
        key="false"/>
06.    <Attribute id="att30" name="dfecfinreal" db:column="dfecfinreal"
        type="date" key="false"/>
07.    <Attribute id="att31" name="dhorainicio" db:column="dhorainicio"
        type="time" key="false"/>
08.    <Attribute id="att32" name="dhorainicioreal" db:column="dhorainicioreal"
        type="time" key="false"/>
09.    <Attribute id="att33" name="dhorafin" db:column="dhorafin" type="time"
        key="false"/>
10.    <Attribute id="att34" name="dhorafinreal" db:column="dhorafinreal"
        type="time" key="false"/>
11.    <Attribute id="att35" name="sformapago" db:column="sformapago"
        type="string" key="false"/>
12.     <Attribute id="att36" name="nadelanto" db:column="nadelanto"
         type="decimal" key="false"/>
13.    <Attribute id="att37" name="sestado" db:column="sestado" type="string"
        key="false"/>
14. </Entity>
```

En la Figura 4.3 se aprecia el diagrama navegacional que satisface los requerimientos planteados en el caso de estudio, se observan las páginas: `Principal`, `Registro clientes`, `Catalogo`, `Reservacion` e `Informacion`, junto con una ventana de diálogo para la captura de comentarios de los usuarios. Las características de cada una de las páginas que conforman la aplicación son:

- `Principal`: Es la página de inicio de la aplicación, muestra un mensaje y un botón para abrir el cuadro de diálogo que contiene el formulario para la captura de los comentarios de los clientes.
- `Registro cliente`: Incluye un mensaje y un formulario para el auto-registro de los clientes en la aplicación.
- `Catalogo`: Contiene un componente de la vista de lista con un flujo de datos hacia un componente de detalles, ambos enlazados a la entidad Automóvil, que se encuentra en el modelo de dominio, esto representa que la página va a contener una lista de los todos los automóviles en el catálogo y permite que el usuario seleccione alguno de ellos y visualice sus detalles.
- `Reservacion`: Consta de un formulario para el registro de reservaciones.
- `Informacion`: Cuenta con un componente de comportamiento mutuamente excluyente, que engloba las opciones: `Autos, Rentas y Danio`, en cuyo interior se encuentran componentes de formulario múltiple para la gestión de las entidades: `Automóvil, Renta y Danio`.

El aspecto navegacional de la aplicación, de acuerdo al modelo navegacional, es que el usuario solo es capaz de llegar a las páginas de `Registro cliente` y `Catalogo` por medio de la página `Principal`. Así mismo, la página para realizar las reservaciones sólo es alcanzable por medio de la página que muestra el catálogo de automóviles, y, la página desde la cual se lleva la gestión de automóviles, rentas y daños es alcanzable por cualquiera de las otras páginas dentro de la aplicación, esto por encontrarse modelada dentro de un contenedor de la vista de área. Respecto al diálogo `Contacto`, es necesario que tenga un comportamiento modal y que el formulario que contiene cuente con las validaciones de campos requeridos y de la verificación de una cadena que tiene el formato de un correo electrónico.

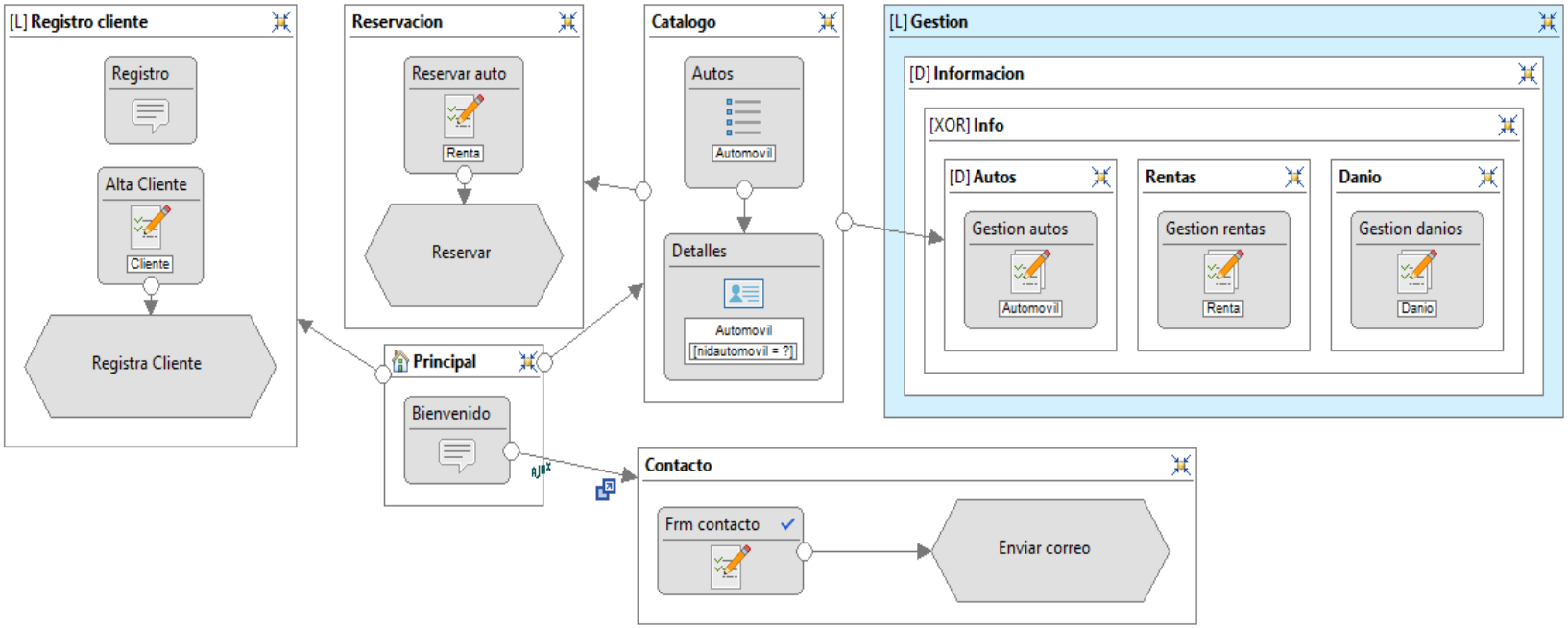

Figura 4.3 Modelo navegacional del caso de estudio

Derivado del modelo navegacional, véase Figura 4.3, se tiene un fragmento del código XML (Listado 4.3) correspondiente a los diversos elementos modelados. En la línea 3 de observa la definición del formulario de contacto, mientras que las líneas 9, 4 y 15 especifican los campos que integran el formulario, en tanto que las líneas 6, 11, 12 y 17 determinan reglas de validación, propias de WebRatio, para los campos del formulario.

Listado 4.3 Fragmento del código XML, generado por WebRatio Web Platform, correspondiente al modelo navegacional del caso de estudio

```
01. <Page gr:x="580" gr:y="365" id="page7" name="Contacto"
     ignoreMasterPage="true">
02.   <ContentUnits>
03.      <EntryUnit gr:x="0" gr:y="5" id="enu1" name="Frm contacto"
          linkOrder="ln11">
04.         <Field id="fld1" name="Nombre" type="string" modifiable="true">
05.            <VRules>
06.               <MandatoryValidationRule id="mnd3" name="Mandatory"/>
07.            </VRules>
08.         </Field>
09.         <Field id="fld2" name="Correo" type="string" modifiable="true">
10.            <VRules>
11.               <MandatoryValidationRule id="mnd2" name="Mandatory"/>
12.               <EMailValidationRule id="email1" name="EMail"/>
13.            </VRules>
14.         </Field>
15.         <Field id="fld3" name="Comentario" type="text" modifiable="true">
16.            <VRules>
17.               <MandatoryValidationRule id="mnd1" name="Mandatory"/>
18.            </VRules>
19.         </Field>
20.         <Link id="ln11" name="Enviar" to="miu3" automaticCoupling="true"
             type="normal" validate="true"/>
21.         </EntryUnit>
22.         <ModuleInstanceUnit gr:x="205" gr:y="0" id="miu3" action="true"
             name="Enviar correo"/>
23.   </ContentUnits>
24.   <!-- Layout -->
25. </Page>
```

4.2 Generador de Aplicaciones Enriquecidas de Internet en funcionamiento

El usuario, al utilizar la aplicación, tiene disponible la interfaz de la Figura 4.4, a través de la cual se recolecta la información necesaria para la generación de una Aplicación Enriquecida de Internet. En el instante que el usuario hace uso del generador de aplicaciones, es necesario que ya cuente con: los archivos XMI y XML que describen los modelos de negocio, dominio y navegacional, junto con los elementos de estilo para la tecnología de su elección, un directorio con archivos CSS y JS para el caso de jQuery o un archivo JAR para el caso de JSF.

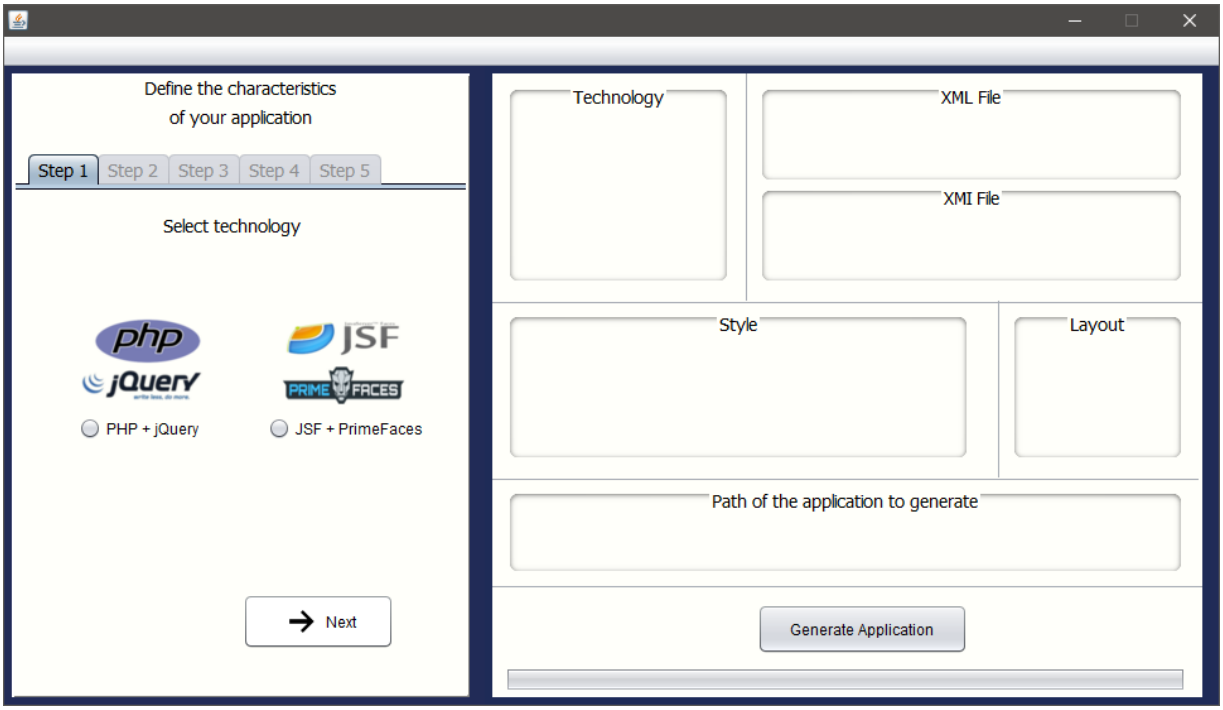

Figura 4.4 Interfaz de la aplicación

Una vez que el usuario selecciona una de las combinaciones de lenguajes disponibles, proporciona los archivos XML y XMI que contienen una descripción de los modelos de la aplicación, suministra los elementos correspondientes de estilo, selecciona una de las cuatro alternativas de distribución de contenido disponibles e indica la ubicación en la cual se va a colocar el directorio de la aplicación generada, el aspecto de la interfaz corresponde al mostrado en la Figura 4.5.

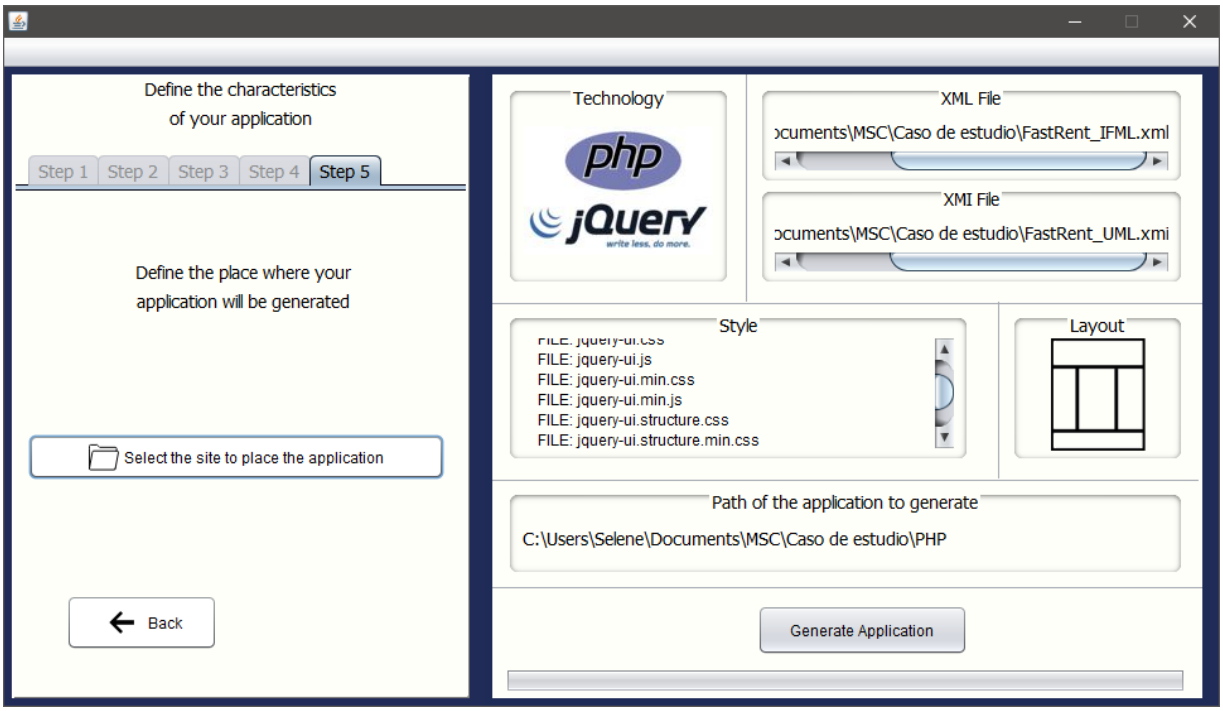

Figura 4.5 Interfaz de la aplicación después de recibir parámetros para la generación de una RIA.

4.3 Aplicación generada en la combinación de PHP y jQuery

La aplicación resultante para la combinación de tecnologías de PHP con jQuery cumple con el comportamiento señalado en el punto 3.2.1.3.1 Arquitectura de las aplicaciones a generar, en donde se especifica que se siguen las pautas dictadas por el patrón arquitectónico MVC. Una vez finalizado el proceso de generación, en la ruta especificada por el usuario se tienen los archivos y directorios correspondientes a la aplicación solicitada, véase Figura 4.6.

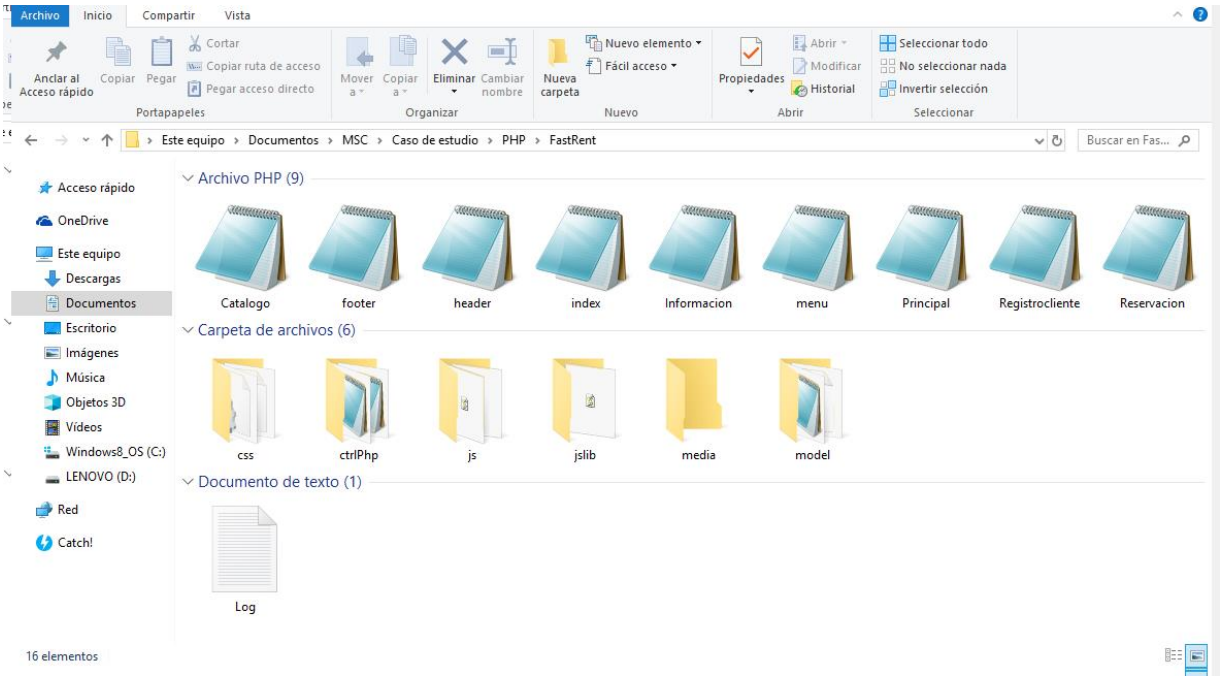

Figura 4.6 Archivos generados, para el caso de estudio, con la combinación de tecnologías PHP y jQuery

Aquí una breve descripción de cada uno de los directorios y archivos generados:

- Directorio raíz, que recibe el mismo nombre que la vista del sitio (siteView) marcada como principal en el diagrama.

- El archivo `index.php` que redirecciona a la página marcada como inicial.
- Los archivos PHP `header` y `footer` de la distribución seleccionada.
- El archivo `menu.php` con el menú de navegación de la aplicación.
- Los archivos PHP correspondientes a las páginas definidas en el modelo navegacional (`Catalogo`, `Informacion`, `Principal`, `Registrocliente`, y `Reservacion`)
- El directorio `css` con los elementos CSS del estilo y la distribución seleccionada.
- El directorio `ctrlPHP` con los archivos PHP que controlan el comportamiento de las páginas, en este caso corresponden a la gestión de las entidades junto con el procesamiento de formularios y validaciones del lado del servidor.

- El directorio `js` con los archivos JavaScript para la manipulación del contenido de las páginas, como lo son: pestañas, tablas, diálogos, campos especiales para la captura de fecha y hora, junto con validaciones, del lado del cliente, por mencionar algunos.
- El directorio `jsLib` contiene la biblioteca jQuery junto con las bibliotecas para el funcionamiento de diálogos, menús y campos especiales de captura de fecha y hora.
- El directorio `media` para archivos multimedia. Si el modelo incluye captcha, el generador coloca en este directorio las imágenes de fondo y para refrescar, en otro caso el directorio permanece vacío. El desarrollador puede colocar más adelante los archivos multimedia necesarios (por ejemplo, imágenes o videos).
- El directorio `model` contiene las clases PHP que representan cada entidad del modelo de dominio, y un archivo JSON, por cada entidad, con datos de prueba para que el usuario visualice los elementos de la aplicación generada.
- El archivo `Log.txt` con la información de los procesos realizados durante la generación de la aplicación (este documento se crea en todas las tecnologías soportadas por el generador).

En conjunto, todos los archivos y directorios descritos conforman la Aplicación Enriquecida de Internet obtenida por el usuario, con el uso del generador, para el caso de estudio señalado.

En la Figura 4.7 se observa como página inicial de la aplicación a `Principal`, asimismo, es notorio que se siguen las pautas indicadas para la distribución de contenido y el uso de las fuentes y colores proporcionadas como estilo. En la zona superior e inferior se encuentran el encabezado y pie de página, mientras que el área lateral izquierda alberga al menú y el área lateral derecha contiene un espacio a ser utilizado posteriormente como el desarrollador lo requiera (dado que uso no se describe en el modelo), la zona central pertenece al contenido modelado para la página, en dónde se tienen presentes los enlaces para navegar a las páginas `Catalogo` y `Registro cliente`, un mensaje de bienvenida a la aplicación, y un botón para mostrar el diálogo con el formulario validado de contacto.

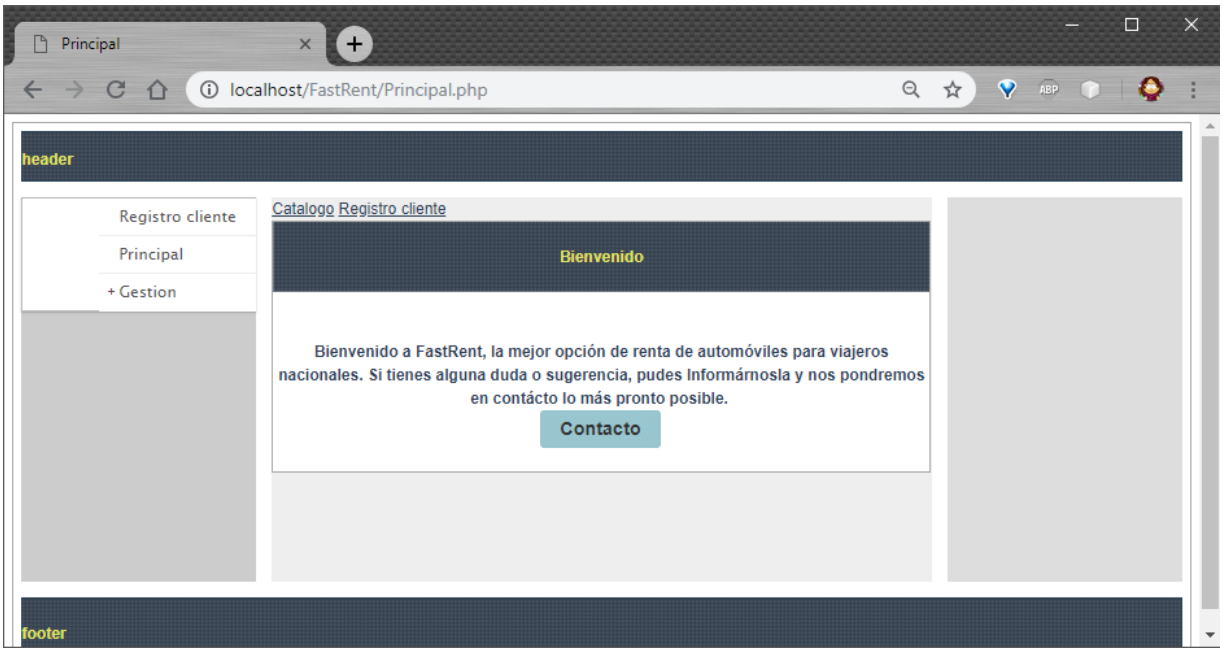

Figura 4.7 Aplicación generada con la combinación de PHP y jQuery, en la página Principal

La Figura 4.8 muestra que el diálogo modelado como modal, efectivamente cumple con este comportamiento. En su interior se tiene el formulario de contacto, por medio del cual el usuario puede comunicar sus quejas y sugerencias a la compañía de rentas de autos. Se observa que el formulario entregado se conforma por los elementos:

- Campo `Nombre`: Se trata de un campo de captura normal, con la validación de captura obligatoria.
- Campo `Correo`: Corresponde a un campo de captura normal, con la validación de captura obligatoria y de que la información introducida corresponda a una dirección de correo electrónico, es decir, es obligatorio que sea una cadena conformada por un nombre de usuario, el símbolo de arroba (@) y un dominio.
- Área de texto `Comentario`: Concierne a un espacio de captura de datos validado para una captura obligatoria.
- Botón `Enviar correo`: Es el botón para procesar el formulario.

En la Figura 4.8 también es posible apreciar que cuando no se satisfacen las condiciones determinadas por las reglas de validación, en la interfaz se le notifica al usuario el elemento está incumpliendo con la verificación.

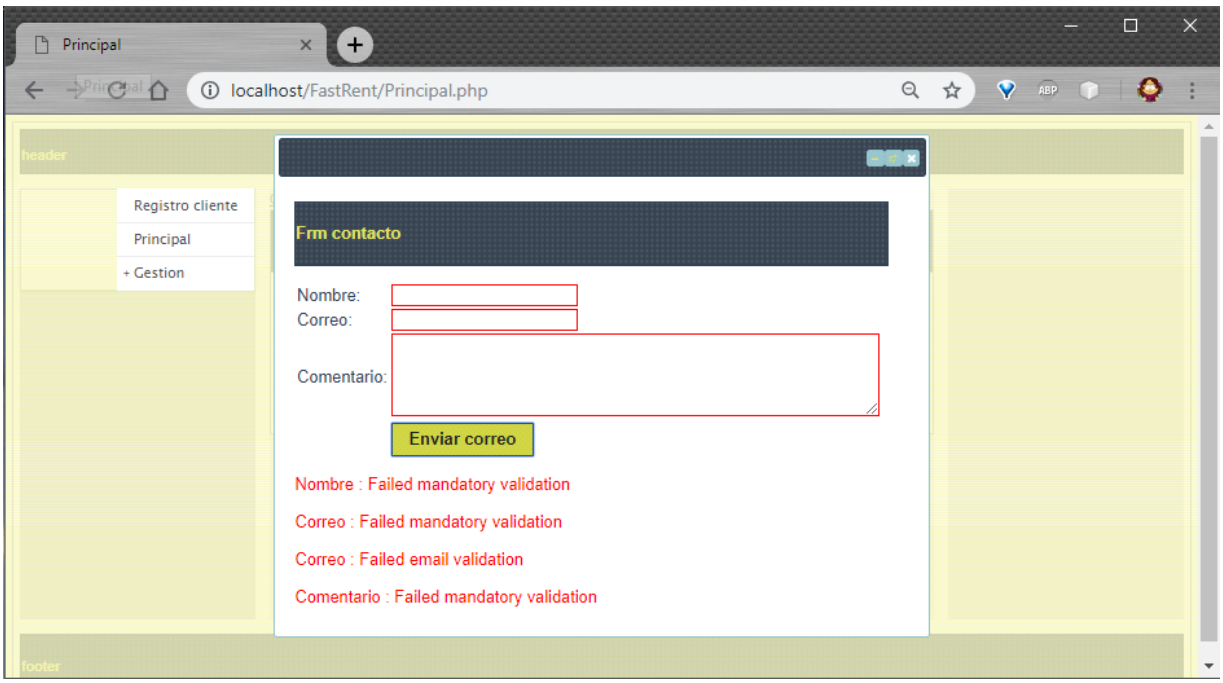

Figura 4.8 Aplicación generada con la combinación de PHP y jQuery, página Principal, diálogo con un formulario validado

La Figura 4.9 muestra que, al satisfacerse todas las validaciones indicadas para los componentes de un formulario se realiza el envío de los datos para su procesamiento. La Figura 4.10 presenta la implementación de la página de auto-registro del cliente, con un mensaje y un formulario de captura. En tanto, que la Figura 4.11 ejemplifica cómo la implementación sigue las pautas del modelo, dado que el campo `fecha de inicio`, el cual indica el inicio de una reservación, se implementa como un campo especial para la selección de fechas. Y la Figura 4.12 exhibe el contenido de `Catalogo.php`, el cual se encuentra conformado por una tabla de selección simple con la información del catálogo de automóviles, en la cual se permite que el usuario visualice, una instancia a la vez, los detalles de los elementos que conforman el catálogo.

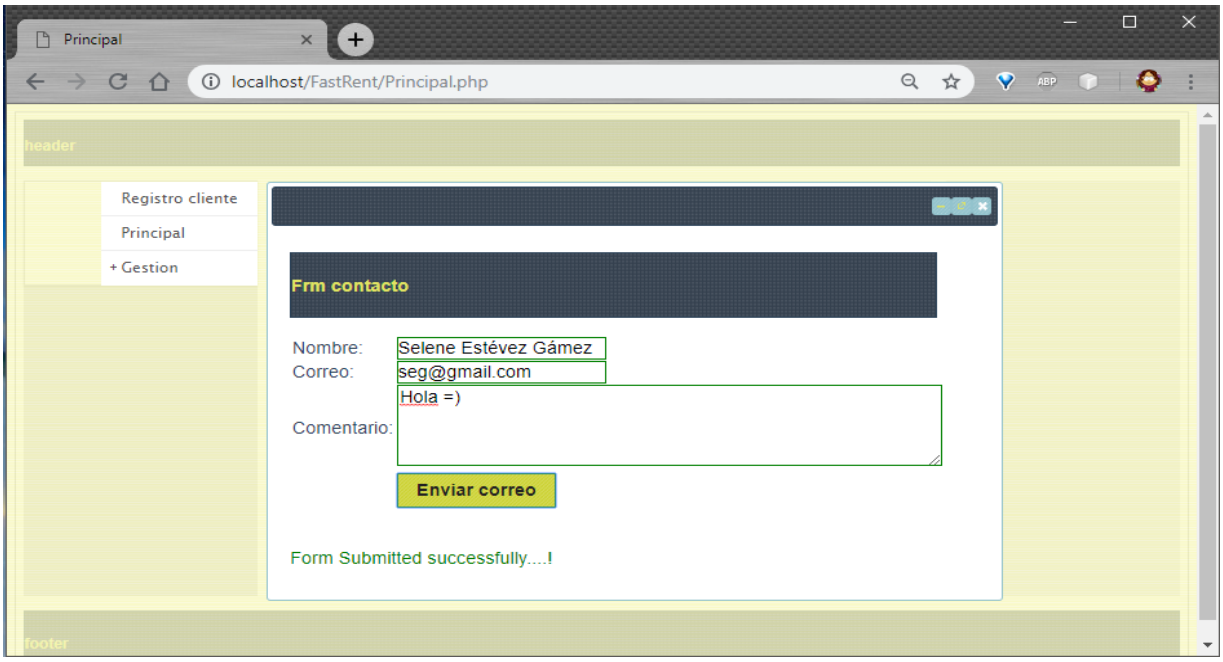

Figura 4.9 Aplicación generada con la combinación de PHP y jQuery, página Principal, diálogo después de procesar un formulario validado

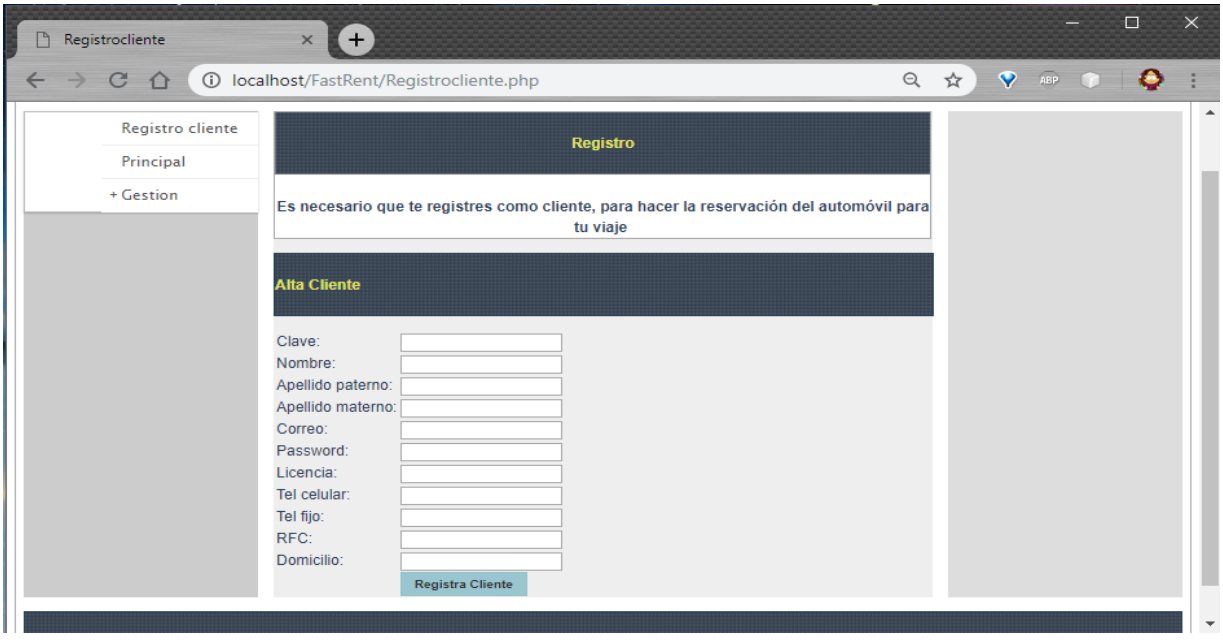

Figura 4.10 Aplicación generada con la combinación de PHP y jQuery, en la página Registrocliente

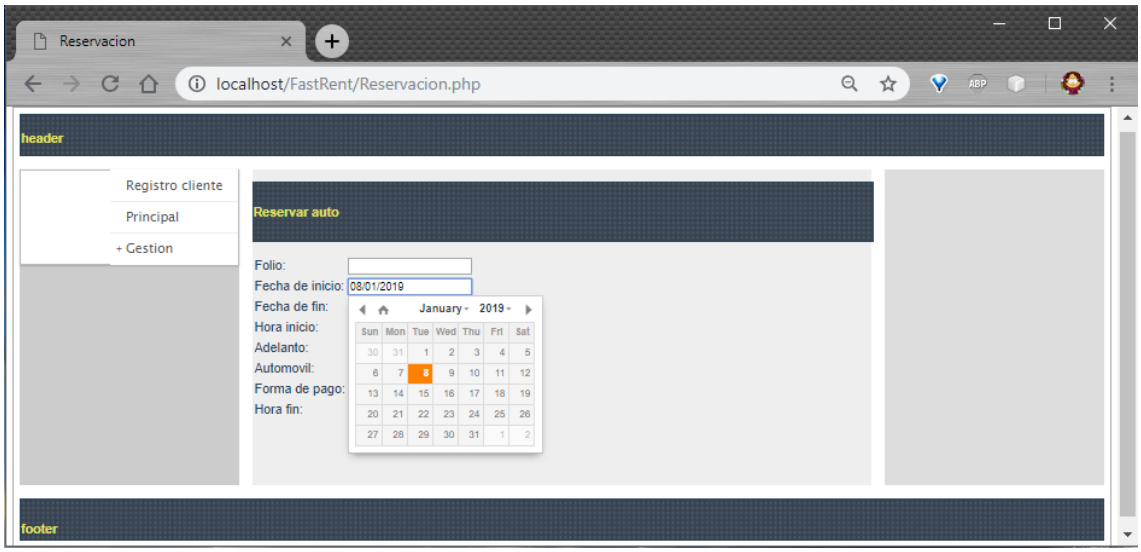

Figura 4.11 Aplicación generada con la combinación de PHP y jQuery, en la página Reservacion

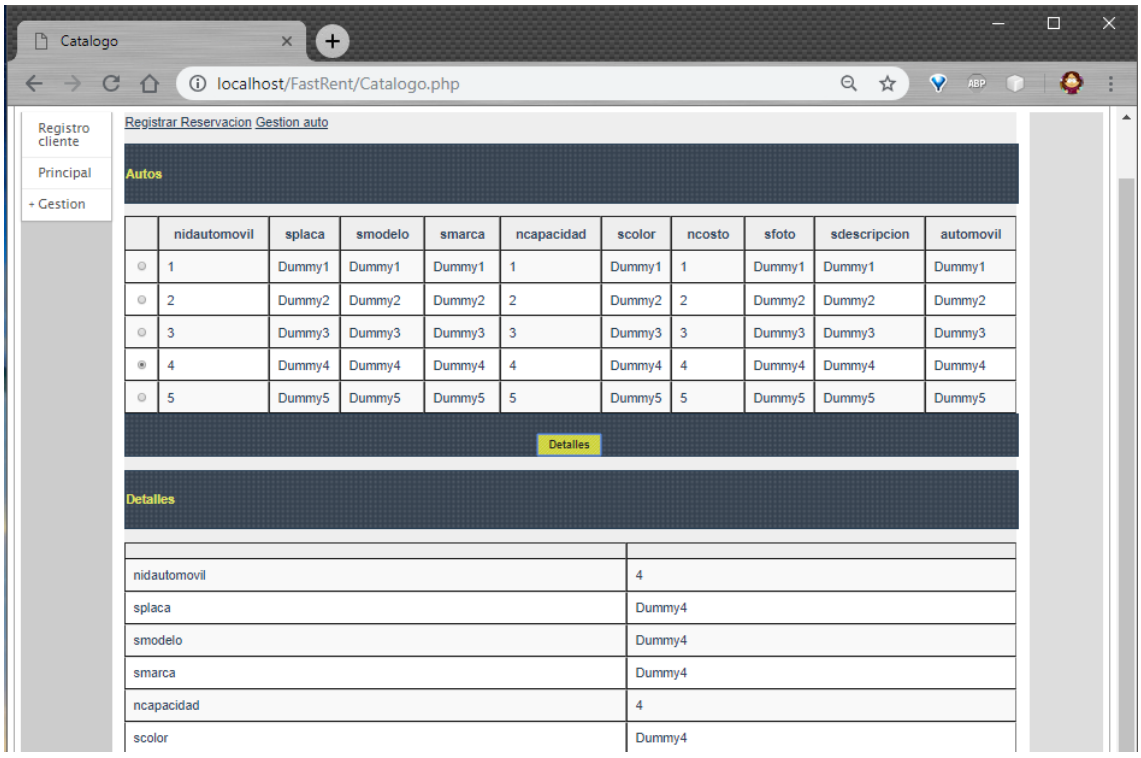

Figura 4.12 Aplicación generada con la combinación de PHP y jQuery, en la página Catalogo

La Figura 4.13 presenta la página `Informacion.php`, cuyo contenido tiene un comportamiento mutuamente excluyente implementado como pestañas: `Autos`, `Rentas` y `Danio`, y dentro de cada una de estas pestañas se encuentran tablas editables para la gestión de los datos de las entidades Automovil, Renta y Danio.

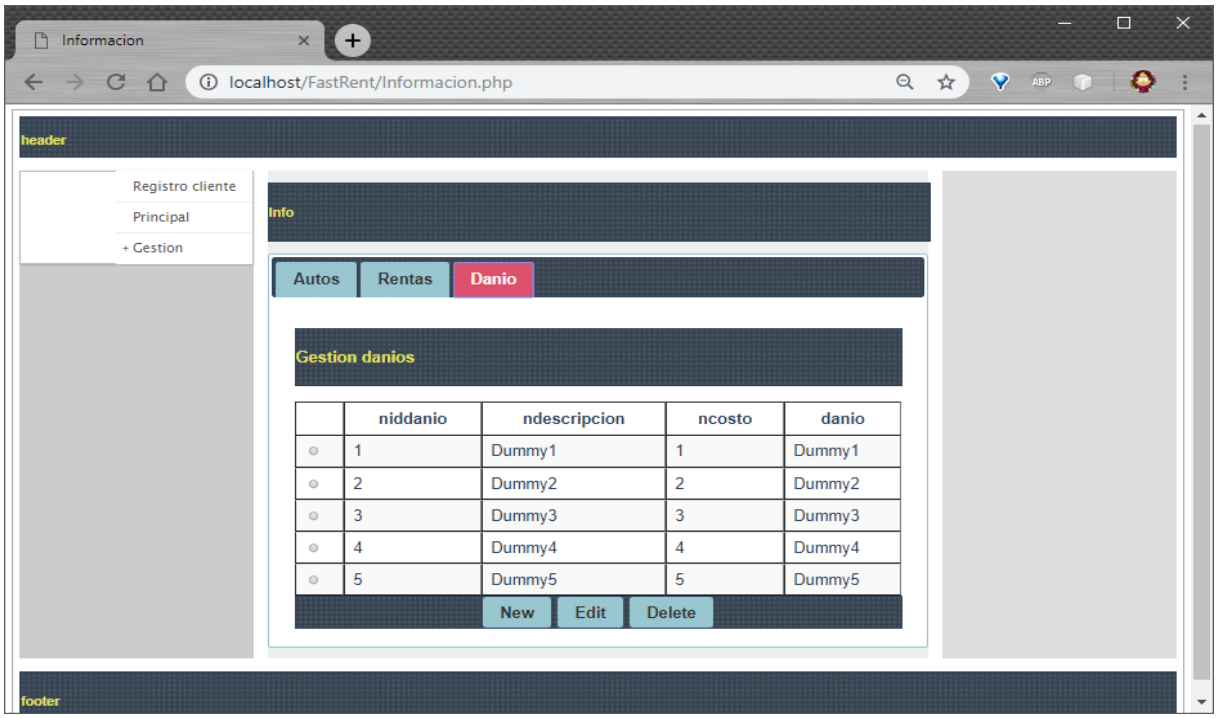

Figura 4.13 Aplicación generada con la combinación de PHP y jQuery, página Informacion, en la pestaña Danio

4.4 Aplicación generada con la combinación de JSF y PrimeFaces

La aplicación obtenida para la combinación de JSF con PrimeFaces cumple con las directrices indicadas en el punto 3.2.1.3.1 Arquitectura de las aplicaciones a generar, en donde se especifica que el funcionamiento de las aplicaciones producidas con esta combinación de tecnologías sigue la pauta de MVC.

La Figura 4.14 expone que la aplicación resultante del proceso de generación, para esta combinación de tecnologías en particular, corresponde a un proyecto Web del IDE NetBeans, lo cual facilita el uso o modificación de la aplicación a los desarrolladores que se valen de dicho entorno de desarrollo. En casos en que el desarrollador necesite manipular o modificar la

aplicación generada con otro IDE, solo es necesario copiar los archivos fuente y colocarlos en los directorios empleados por el entorno de desarrollo utilizado.

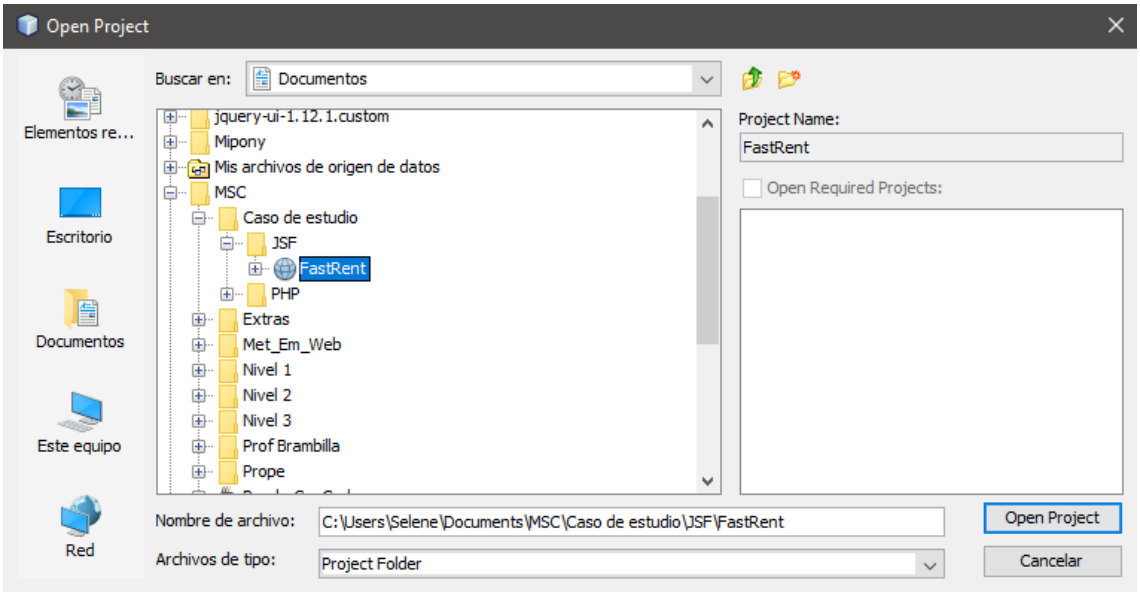

Figura 4.14. Proyecto de NetBeans generado, para el caso de estudio, con la combinación de tecnologías JSF y PrimeFaces

La Figura 4.15 muestra los directorios y archivos que conforman la aplicación generada para esta tecnología. Aquí una breve descripción de ellos:

- Directorio raíz, que recibe el mismo nombre que la vista del sitio (siteView) marcada como principal en el diagrama

- El directorio `lib` aloja archivos JAR indispensables para el funcionamiento de la aplicación, los cuales corresponden con: a) la biblioteca de Java en su versión empresarial Web para el soporte de las etiquetas de JavaServer Faces, b) la biblioteca para el manejo de los archivos JSON que simulan el repositorio de información, c) la biblioteca de PrimeFaces, y finalmente d) el archivo con las directrices de estilo visual para la aplicación.
- El directorio `templates` contiene la plantilla de la aplicación, y su contenido depende de la distribución seleccionada.

- El directorio `facelets`, además de alojar a los directorios `lib` y `templates`, alberga los archivos XML del proyecto para el trabajo con el servidor Glassfish.
- El directorio `css` contiene el archivo CSS correspondiente a la distribución de contenido indicada por el usuario.
- El directorio `media` aloja los archivos multimedia de la aplicación, se coloca vacío para que el desarrollador decida su contenido, debido a que no es parte del modelado.
- El directorio `resources` tiene en su interior los directorios `css` y `media`.
- El directorio `web` aloja los archivos XHTML que corresponden a las páginas definidas en el modelo navegacional (`Catalogo`, `Informacion`, `Principal`, `Registrocliente`, y `Reservacion`) junto con los directorios `resources` y `WEB-INF`.
- El directorio `jb` contiene los archivos JavaBean para el despliegue de información en las |páginas XHTML, además de la validación de campos y procesamiento de formularios.
- El directorio `model` comprende los archivos Java que representan a cada una de las entidades del modelo de dominio junto con archivos JSON que tienen información de prueba útil para visualizar el funcionamiento de la aplicación generada.

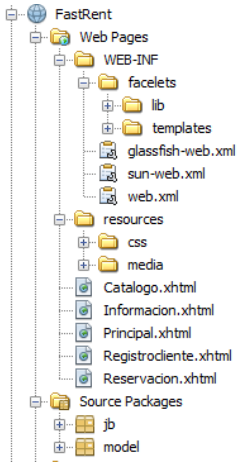

Figura 4.15 Aplicación generada con la combinación de JSF y PrimeFaces como proyecto Web de NetBeans

De la Figura 4.16 la Figura 4.23 se muestra la aplicación generada para esta tecnología, cuyo funcionamiento corresponde con el modelo navegacional proporcionado. La explicación del comportamiento detallado de la aplicación se encuentra descrito en el punto 4.3 Aplicación generada con la combinación de PHP y jQuery, ya que ambas aplicaciones en el *front-end* son idénticas, excepto por las peculiaridades gráficas propias de cada una de las tecnologías, lo que demuestra que el mismo modelo se implementa de forma equivalente en diferentes tecnologías.

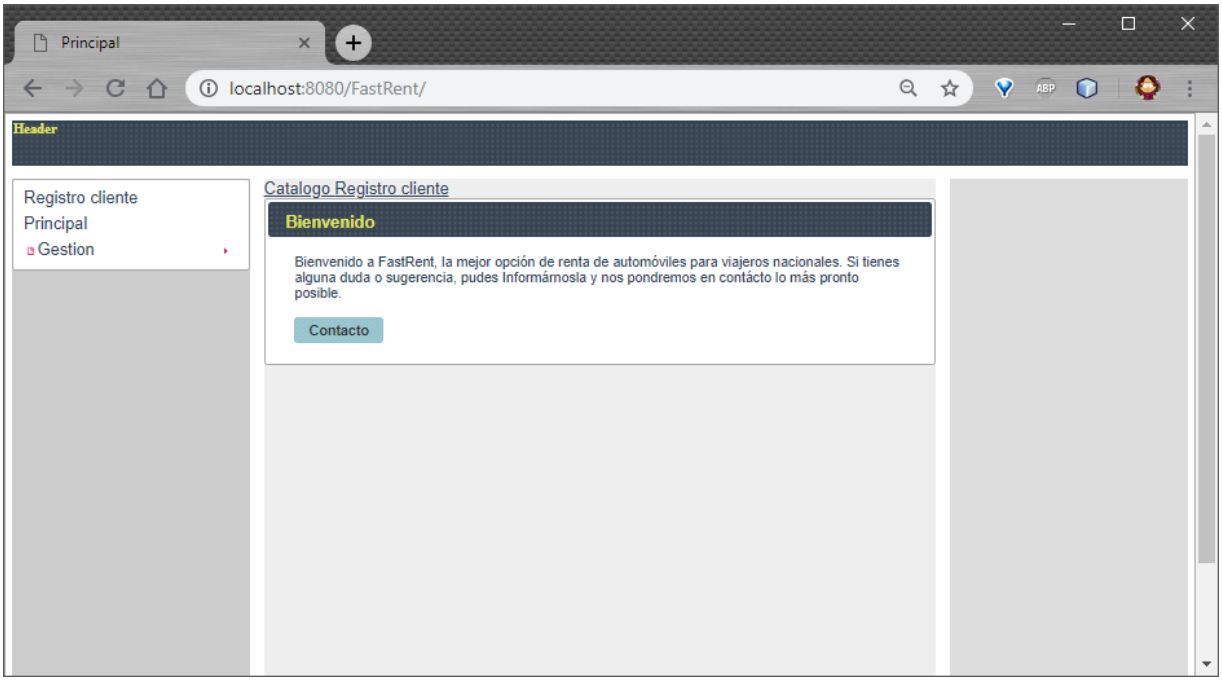

Figura 4.16 Aplicación generada con la combinación de JSF y PrimeFaces, en la página Principal

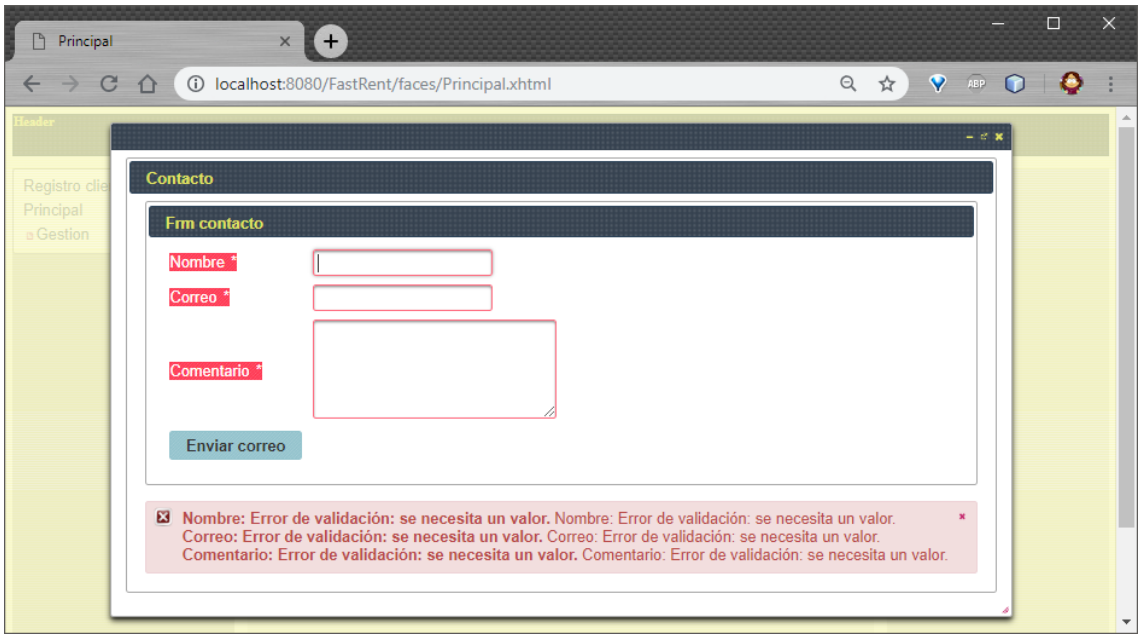

Figura 4.17 Aplicación generada con la combinación de JSF y PrimeFaces, página Principal, diálogo con un formulario validado

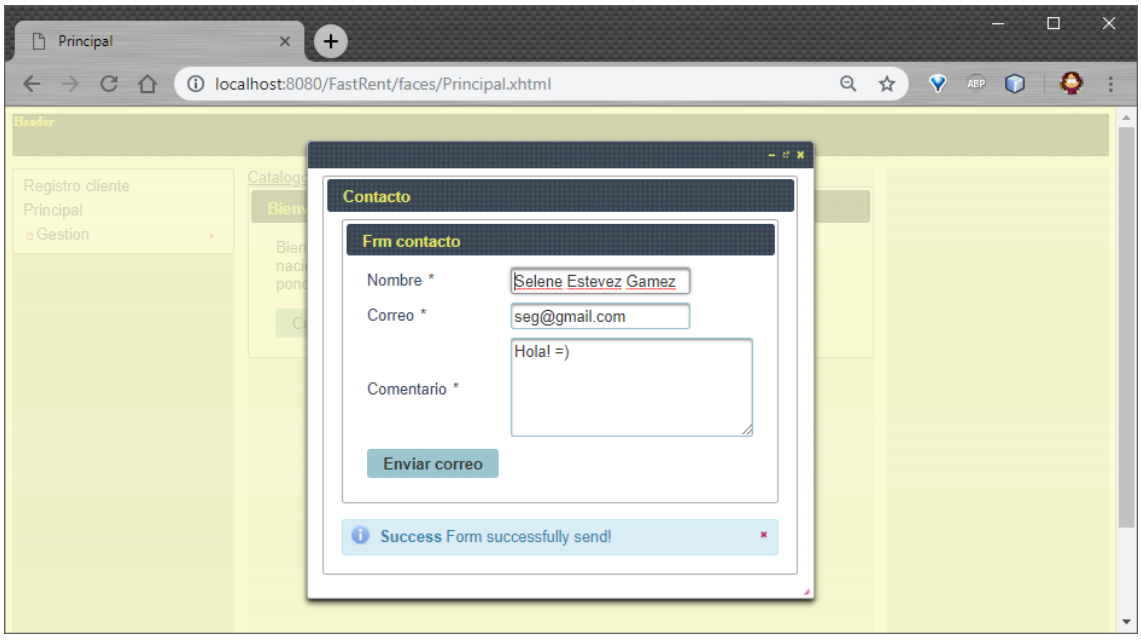

Figura 4.18 Aplicación generada con la combinación de JSF y PrimeFaces, página Principal, diálogo después de procesar un formulario validado

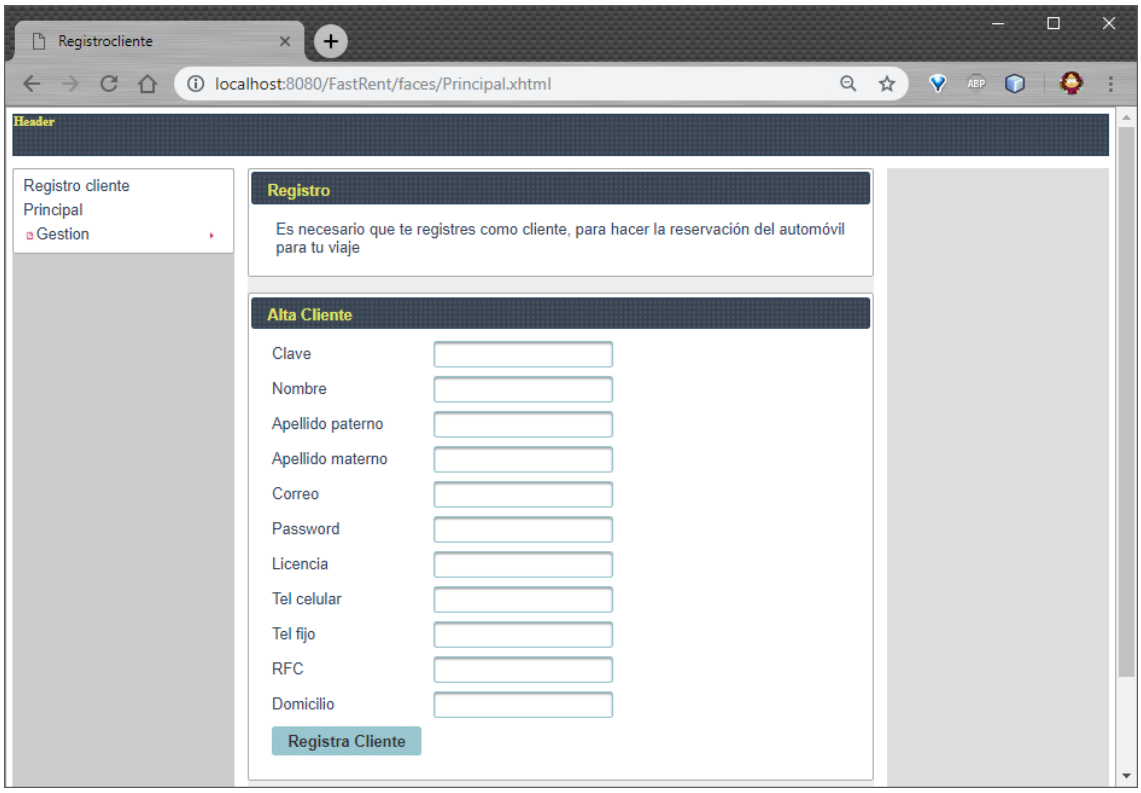

Figura 4.19 Aplicación generada con la combinación de JSF y PrimeFaces, en la página Registrocliente

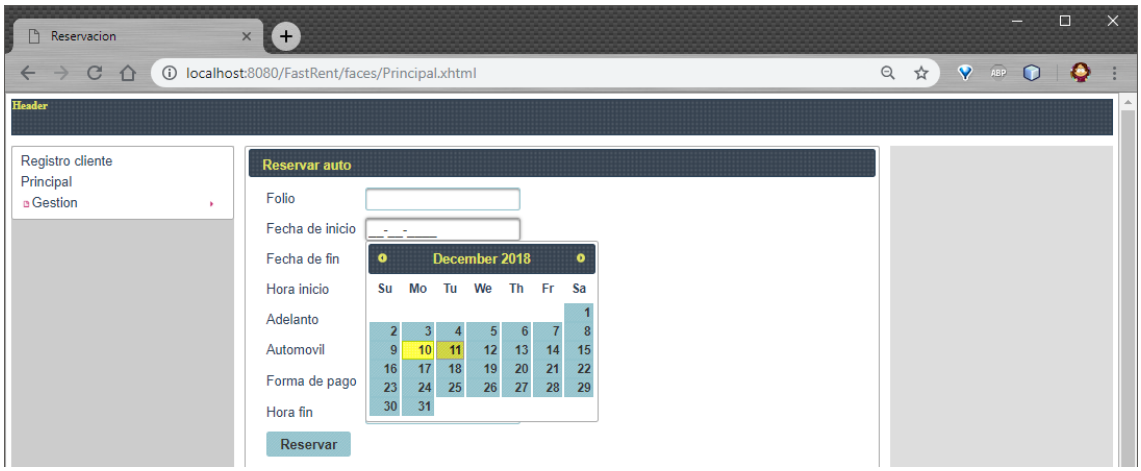

Figura 4.20 Aplicación generada con la combinación de JSF con PrimeFaces, en la página Reservacion

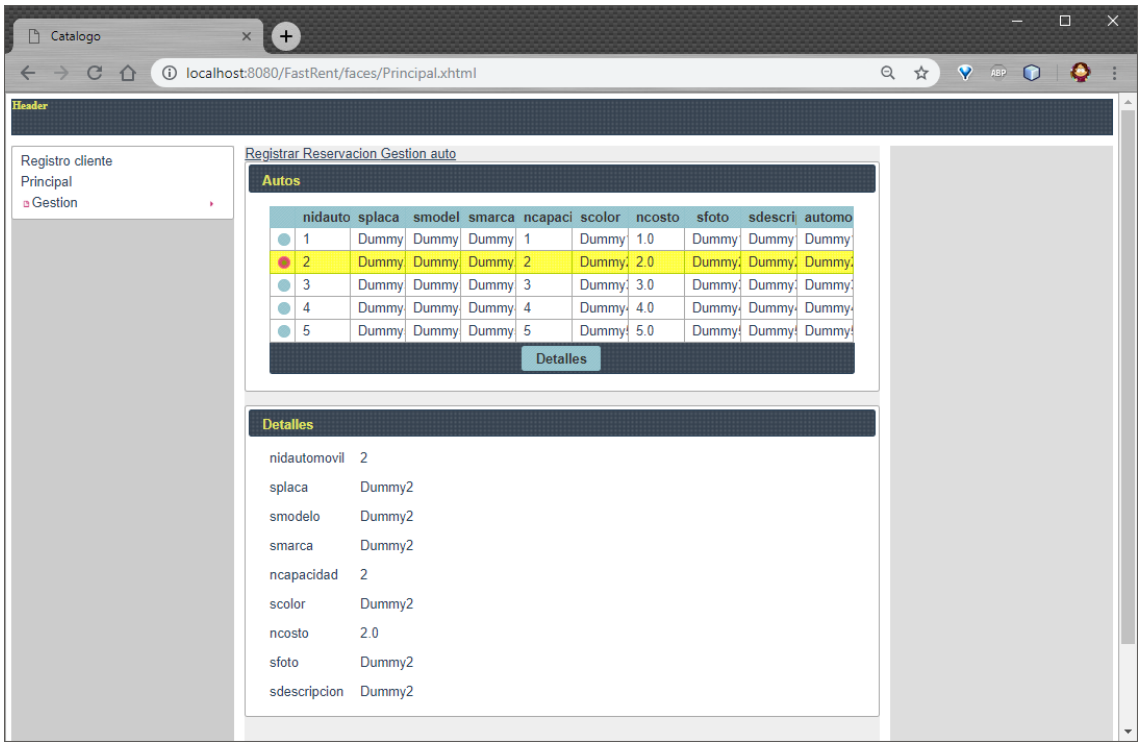

Figura 4.21 Aplicación generada con la combinación de JSF y PrimeFaces, en la página Catalogo

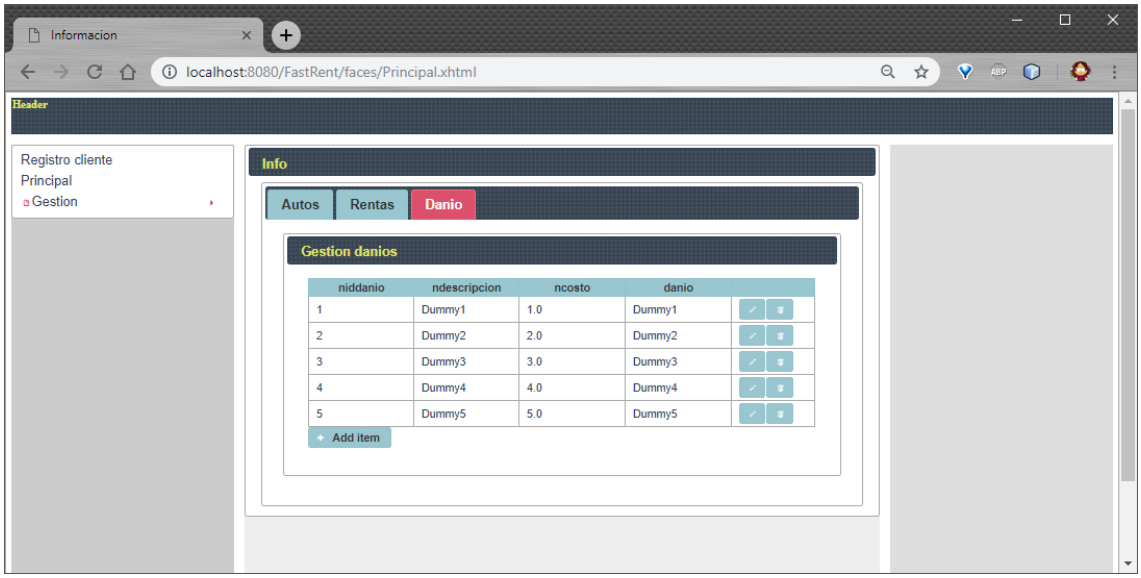

Figura 4.22 Aplicación generada con la combinación de PHP y jQuery, página Informacion, en la pestaña Danio

4.5 Comprobación de la robustez de la arquitectura del generador

La robustez de la arquitectura se comprobó mediante la adición del módulo para la generación de código final en la combinación de PHP con AngularJS. Es importante señalar que la arquitectura del generador recibe el incremento de tecnología de forma natural, sin requerir cambios en su estructura, sólo ajustes en la interfaz para la selección de AngularJS como tecnología de generación y validar la entrada del archivo CSS con las directivas correspondientes al estilo visual.

AngularJS es un marco de trabajo de JavaScript orientado al desarrollo de Aplicaciones Enriquecidas de Internet que: a)Extiende la sintaxis de HTML por medio de atributos propios, del marco de trabajo, para la especificación del comportamiento dinámico, b) Sigue las directrices del patrón arquitectónico MVC, y c) Considera el uso de programación declarativa para construcción de la vista y la programación imperativa para la lógica de negocio [55].

La Figura 4.23 muestra la arquitectura de las aplicaciones generadas en Angular JS, en donde: 1) El cliente realiza una petición a la vista, 2) La vista manipula al controlador, 3) El controlador da instrucciones al modelo, 4) El modelo responde a la solicitud del controlador, y devuelve la información solicitada 5) El controlador realiza la acción requerida y pasa los datos a la vista, y por último 6) La vista expone el resultado al cliente.

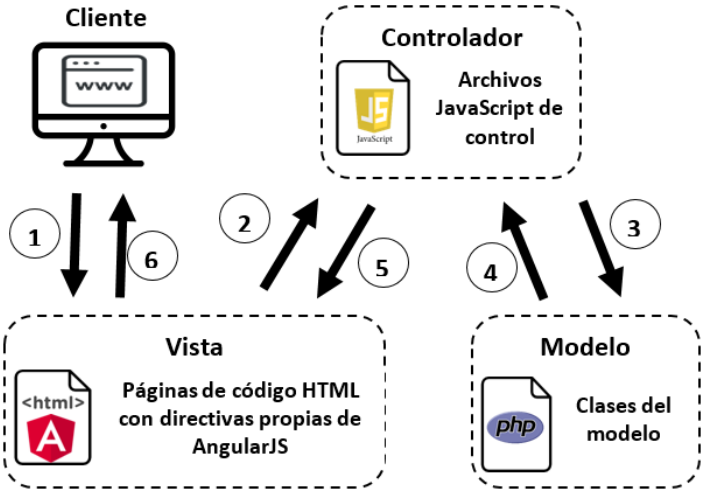

Figura 4.23 Arquitectura basada en MVC de las aplicaciones a generar en AngularJS

La aplicación resultante para la combinación de tecnologías de PHP con AngularJS cumple con las pautas dictadas por el patrón arquitectónico. Una vez finalizado el proceso de generación, en la ruta especificada por el usuario se tienen los archivos y directorios correspondientes a la aplicación solicitada, véase Figura 4.24.

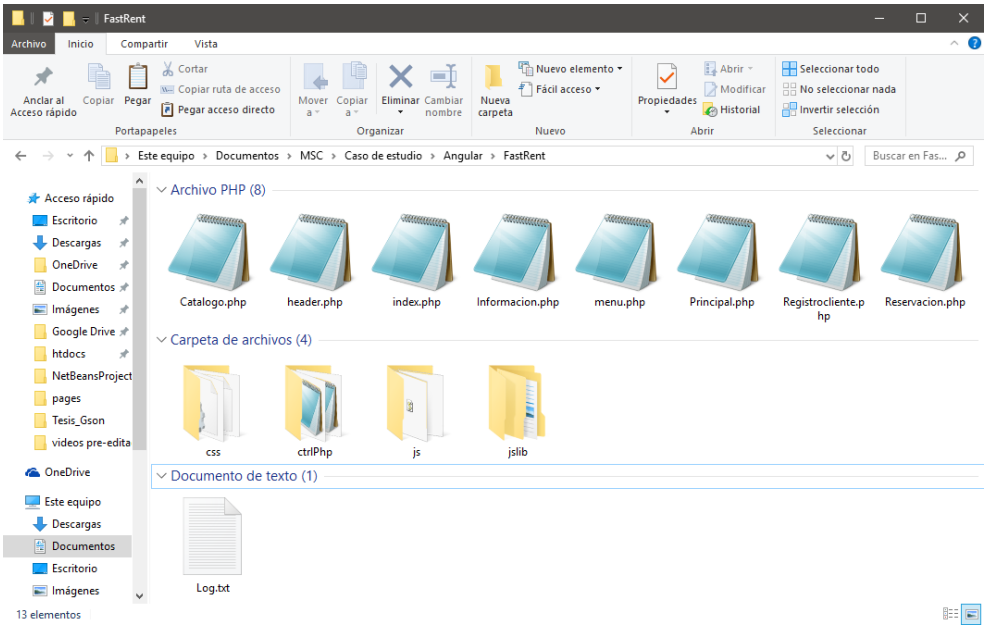

Figura 4.24 Archivos generados, para el caso de estudio, con la combinación de tecnologías PHP y AngularJS

El conjunto de archivos y directorios entregados en AngularJS es el mismo a los obtenidos como salida el punto 4.3 Aplicación generada en la combinación de PHP y jQuery, porque: a) usan el mismo lenguaje del lado del servidor, y, b) tanto jQuery como AngularJS trabajan con JavaScript.

Las diferencias puntuales de los elementos entregados con AngularJS respecto a los obtenidos con PHP y jQuery son:

- El contenido HTML de las páginas `Catalogo`, `Informacion`, `Principal`, `Registrocliente`, y `Reservacion` cuenta con atributos propios de AngularJS para el comportamiento dinámico.

- El directorio `jsLib` contiene la biblioteca AngularJS junto con las bibliotecas para el funcionamiento de menús.

En conjunto, todos los archivos y directorios descritos conforman la Aplicación Enriquecida de Internet obtenida en Angular JS para el caso de estudio señalado.

La Figura 4.25 muestra la aplicación generada para esta tecnología, cuyo funcionamiento corresponde con el modelo navegacional proporcionado. Cabe destacar, que en situaciones en las cuales se encuentra un componente que no es soportado por la tecnología, se incluye un comentario que señala que el elemento fue reconocido, sin embargo, no cuenta con una implementación en la tecnología, como lo muestra la Figura 4.26.

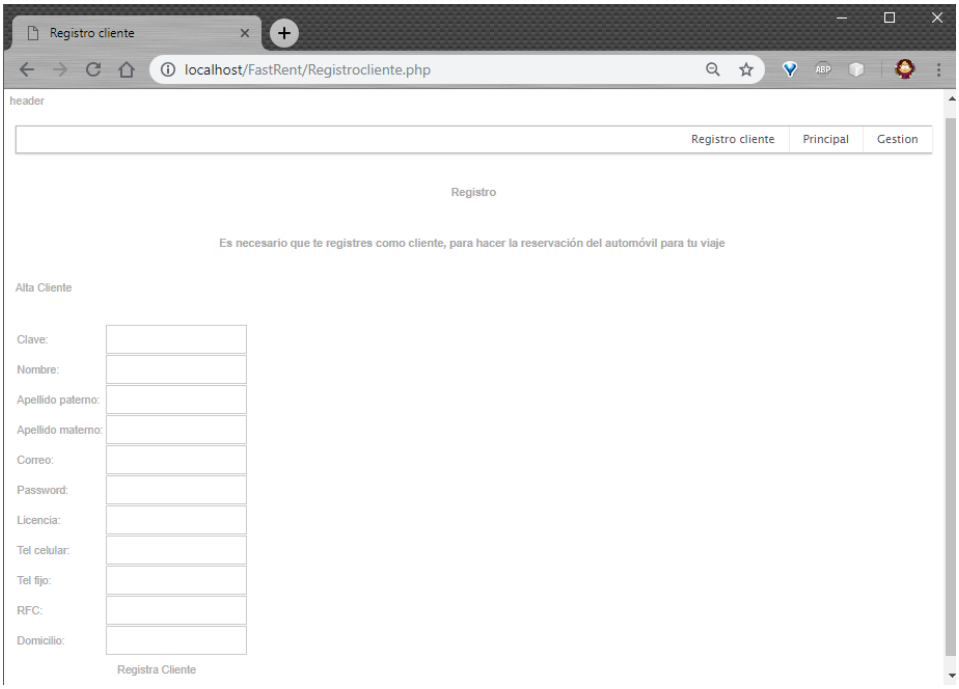

Figura 4.25 Aplicación generada para AngularJS

Para el desarrollo de este módulo se contó con el apoyo de un residente de licenciatura, el cual, llevó a cabo la implementación de la representación intermedia a código final con características propias de AngularJS.

Catalogo.php

```
1  <?php
2  $page_title = 'Catalogo';
3  include_once("header.php");
4  include_once("menu.php");
5  include_once("ctrlPhp/ctrlReservacion.php");
6  include_once("ctrlPhp/ctrlInformacion.php");
7  ?>
8              <main class="mainContent">
9
10                 <a id="ln7"  href="<?=validateAndNavigateReservacion()?>">Registrar
                   Reservacion</a>
11                 <a id="ln9"  href="<?=validateAndNavigateInformacion()?>">Gestion auto
                   </a>
12             <!-- Unsupported element named Autos -->
13             <!-- Unsupported element named Detalles -->
14                 </main>
15             </section>
16         </div>
17     </body>
18 </html>
```

Figura 4.26 Generación de un elemento no soportado en AngularJS

4.6 Contribución del generador de aplicaciones en la educación

El generador de aplicaciones se presentó en dos grupos de la materia “Metodologías emergentes para Web”, que forma parte de la carrera de Ingeniería en Sistemas Computacionales impartida en el Instituto Tecnológico de Orizaba, y que en una de sus unidades se enfoca al modelado de aplicaciones Web.

A los alumnos que comienzan a realizar modelado de aplicaciones Web y que, en ese punto de su formación académica, no cuentan con suficientes conocimientos sobre desarrollo Web, les es difícil comprobar si los modelos que realizar efectivamente satisfacen los requerimientos de las aplicaciones que les son solicitadas en clase, por tal motivo, se especula que el generador desarrollado es una herramienta que les permitirá comprobar de forma certera si sus modelos satisfacen los requerimientos de un usuario.

Por motivo de cumplir con la fecha estipulada para la entrega del presente documento de tesis junto con el producto desarrollado, no fue posible la obtención de información estadística sobre el beneficio del generador en los grupos; sin embargo, los comentarios recibidos durante las presentaciones, tanto de parte de las profesoras como de los alumnos, apuntan en esa misma dirección.

Capítulo 5. Conclusiones y recomendaciones

Este capítulo resume las ideas principales mencionadas en la tesis para dar una conclusión acerca del trabajo realizado, se señalan los objetivos alcanzados y se apunta al trabajo por hacer en cuanto a la generación de aplicaciones a partir de modelos.

5.1 Conclusiones

La principal contribución de este trabajo de tesis consiste en el desarrollo de un generador de Aplicaciones Enriquecidas de Internet que a partir de diagramas navegacionales IFML y diagramas de clases UML entrega esqueletos de RIAs en PHP con jQuery o JSF con PrimeFaces, de acuerdo al patrón arquitectónico MVC, que siguen las directrices de los modelos junto con las pautas de distribución y de estilo indicadas por el usuario.

Para el desarrollo de este trabajo se realizó el análisis comparativo entre las principales herramientas de modelado IFML actualmente disponibles en el mercado, el cual, era un conocimiento inexistente y del que se halló a WebRatio Web Platform como el instrumento idóneo para la obtención de la información del modelo de dominio y navegacional, debido a que presenta apego al estándar IFML y proporciona una descripción de los modelos en un lenguaje procesable.

Otra contribución, es que las aplicaciones generadas no proporcionan código SQL para el trabajo con un repositorio de información particular, ya que cada una de las RIAs generadas simula su propio repositorio por medio de archivos JSON para cada una de las entidades dentro del modelo de dominio proporcionado, resultando en que el usuario puede visualizar apropiadamente las aplicación generadas y el desarrollador tiene la capacidad de complementar el código entregado para el trabajo con el repositorio de su elección.

En cuanto a la arquitectura definida para el generador desarrollado, sus características de modularidad y facilidad de crecimiento se comprobaron con los casos de estudio, de los cuales se: a) cubre la funcionalidad de la aplicación, b) muestran las ventajas que provee para los analistas de requerimientos, y, c) comprueba que la aplicación está preparada para la incorporación de nuevas tecnologías, mientras que se siga la arquitectura de la aplicación final.

Con todo lo anterior se afirma que se cumplieron cada uno de los objetivos específicos marcados para el desarrollo de este proyecto de tesis.

5.1.1 Recomendaciones

Como ya se ha mencionado, un aspecto relevante del generador es su capacidad de permitir que con el tiempo le sean añadidos módulos para generar otras combinaciones de lenguajes destino, de tal forma, que se considera la adición de nuevos módulos para la generación de nuevas combinaciones de tecnologías.

Dentro de las limitaciones de la herramienta, inherentes a IFML, se encuentra la distribución horizontal de los controles dentro de los formularios, por lo cual se propone, como trabajo a futuro, la generación de características particulares como la distribución de elementos dentro de la distribución general.

Productos académicos

Selene Estévez Gámez, Beatriz Alejandra Olivares Zepahua, Ignacio López Martínez, Celia Romero Torres, Luis Ángel Reyes Hernández.

Análisis Comparativo de Herramientas de Modelado IFML.

Avances en Tecnologías de la Información

Estado: *Presentado*

Selene Estévez Gámez, Beatriz Alejandra Olivares Zepahua, Ignacio López Martínez, Celia Romero Torres, Luis Ángel Reyes Hernández.

Arquitectura para un generador de Aplicaciones Enriquecidas de Internet a partir de modelos IFML.

Revista Tlamati Sabiduría (ISNN 2007-2066)

Estado: *Presentado*

Referencias

[1] J. Conallen, *Building Web Applications with Uml*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] M. J. Hadley, "Web Application Description Language (WADL)", Sun Microsystems, Inc., Mountain View, CA, USA, 2006.

[3] Duhl , Joshua, "Rich Internet Applications", *IDC white papers*, nov-2003. [En línea]. Disponible en: https://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf. [Consultado: 17-sep-2017].

[4] J. C. Preciado, M. Linaje, F. Sanchez, y S. Comai, "Necessity of methodologies to model rich Internet applications", en *Seventh IEEE International Symposium on Web Site Evolution*, 2005, pp. 7–13.

[5] R. N. Taylor, N. Medvidovic, y E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[6] E. T. López, A. O. Ramon, E. M. Sarroca, y C. G. Seone, *Diseño de sistemas software en UML*. Universitat Politecnica de Catalunya. Iniciativa Digital Politecnica, 2004.

[7] V. J. E. Muñoz, *El nuevo PHP. Conceptos avanzados.:* Bubok Publishing, 2013.

[8] C. M. Z. Restrepo, *Hacia Una Comunidad Educativa Interactiva*. Fondo Editorial Universidad EAFIT, 2007.

[9] S. A. C. T. D. A. V. Z. Sonia Jaramillo Valbuena, *Programación Avanzada en Java*. Elizcom S.a.s.

[10] Draheim, Dirk y Weber, Gerald, *Form-Oriented Analysis. A New Methodology to Model Form-Based Applications*, 1a ed. Springer-Verlag Berlin Heidelberg, 2005.

[11] OMG, "XML™", *XML*. [En línea]. Disponible en: http://www.omg.org/technology/readingroom/XML.htm. [Consultado: 21-oct-2017].

[12] OMG, "ABOUT THE XML METADATA INTERCHANGE SPECIFICATION VERSION 2.5.1". jun-2015.

[13] OMG, "INTRODUCTION TO OMG'S UNIFIED MODELING LANGUAGE™ (UML®)", *WHAT IS UML*. [En línea]. Disponible en: http://www.uml.org/what-is-uml.htm. [Consultado: 17-sep-2017].

[14] OMG, "OMG Unified Modeling Language TM (OMG UML) Version 2.5". mar-2015.

[15] Visual Paradigm, "Drawing class diagrams", *Visual Paradigm Support*. [En línea]. Disponible en: https://www.visual-paradigm.com/support/documents/vpuserguide/94/2576/7190_drawingclass.html. [Consultado: 20-oct-2017].

[16] uml-diagrams.org, "Class diagrams overview", *uml-diagrams.org*. [En línea]. Disponible en: http://www.uml-diagrams.org/class-diagrams-overview.html. [Consultado: 20-oct-2017].

[17] M. Brambilla y P. Fraternali, *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.

[18] J. Juneau, *JavaServer Faces: Introduction by Example*. Apress, 2014.

[19] A. Bailey y S. Jonna, *PrimeFaces Theme Development*. Packt Publishing, 2015.

[20] Çivici, Çağatay, "PrimeFaces User Guide 6.1", *Documentation*. [En línea]. Disponible en: https://www.primefaces.org/docs/guide/primefaces_user_guide_6_1.pdf. [Consultado: 20-sep-2017].
[21] R. Lerdorf y K. Tatroe, *Programming PHP*. O'Reilly Media, Incorporated, 2002.
[22] W3 Techs, "Usage of server-side programming languages for websites", *Server-side Languajes*. [En línea]. Disponible en: https://w3techs.com/technologies/overview/programming_language/all. [Consultado: 21-oct-2017].
[23] ECMA International, "The JSON Data Interchange Format", *Standandard ECMA-404*, oct-2013. [En línea]. Disponible en: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Consultado: 21-oct-2017].
[24] L. Van Lancker, *jQuery: el framework JavaScript de la Web 2.0*. Ediciones ENI, 2014.
[25] A. V. Aho, M. S. Lam, R. Sethi, y J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.
[26] T. Parr, *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2012.
[27] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*, 1st ed. Pragmatic Bookshelf, 2009.
[28] J. Lambert, *Microsoft Office for iPad Step by Step*. Pearson Education, 2015.
[29] S. Smith, "Layout in ASP.NET Core", *Layout in ASP.NET Core*, 14-oct-2016. [En línea]. Disponible en: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-2.1#what-is-a-layout. [Consultado: 04-mar-2018].
[30] B. A. GUÉRIN, *ASP.NET en C# con Visual Studio 2015: Diseño y desarrollo de aplicaciones Web*. ENI, 2016.
[31] Ó. P. de San Antonio, *Manual imprescindible de CSS*. Anaya Multimedia, 2010.
[32] R. Acerbis, A. Bongio, S. Butti, y M. Brambilla, "Model-driven Development of Cross-platform Mobile Applications with WebRatio and IFML", en *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, Piscataway, NJ, USA, 2015, pp. 170–171.
[33] N. Laaz y S. Mbarki, "A model-driven approach for generating RIA interfaces using IFML and ontologies", presentado en 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt), 2016, pp. 83–88.
[34] A. Salini, I. Malavolta, y F. Rossi, "Leveraging Web Analytics for Automatically Generating Mobile Navigation Models", presentado en 2016 IEEE International Conference on Mobile Services (MS), 2016, pp. 103–110.
[35] N. Laaz y S. Mbarki, "Integrating IFML models and owl ontologies to derive UIs web-Apps", presentado en 2016 International Conference on Information Technology for Organizations Development (IT4OD), 2016, pp. 1–6.
[36] M. Brambilla, A. Mauri, y E. Umuhoza, "Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End", en *Mobile Web Information Systems: 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27-29, 2014. Proceedings*, I. Awan, M. Younas, X. Franch, y C. Quer, Eds. Cham: Springer International Publishing, 2014, pp. 176–191.

[37] E. Umuhoza, H. Ed-douibi, M. Brambilla, J. Cabot, y A. Bongio, “Automatic Code Generation for Cross-platform, Multi-device Mobile Apps: Some Reflections from an Industrial Experience”, en *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*, New York, NY, USA, 2015, pp. 37–44.
[38] K. Frajták, M. Bureš, y I. Jelínek, “Transformation of IFML Schemas to Automated Tests”, en *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems*, New York, NY, USA, 2015, pp. 509–511.
[39] C. Bernaschina, S. Comai, y P. Fraternali, “Online Model Editing, Simulation and Code Generation for Web and Mobile Applications”, en *Proceedings of the 9th International Workshop on Modelling in Software Engineering*, Piscataway, NJ, USA, 2017, pp. 33–39.
[40] R. Rodriguez-Echeverria, J. M. Conejero, J. C. Preciado, y F. Sanchez-Figueroa, “AutoCRUD - Automating IFML Specification of CRUD Operations”, en *Proceedings of the 12th International Conference on Web Information Systems and Technologies - Volume 1: APMDWE, (WEBIST 2016)*, 2016, pp. 307–314.
[41] S. Roubi, M. Erramdani, y S. Mbarki, “A model driven approach to generate graphical user interfaces for Rich Internet Applications using Interaction Flow Modeling Language”, en *2015 15th International Conference on Intelligent Systems Design and Applications (ISDA)*, 2015, pp. 272–276.
[42] Y. Rhazali, Y. Hadi, y A. Mouloudi, “A model transformation in MDA from CIM to PIM represented by web models through SoaML and IFML”, en *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*, 2016, pp. 116–121.
[43] F. P. Basso, R. M. Pillat, T. C. Oliveira, F. Roos-Frantz, y R. Z. Frantz, “Automated design of multi-layered web information systems”, *J. Syst. Softw.*, vol. 117, núm. Supplement C, pp. 612–637, 2016.
[44] S. Roubi, M. Erramdani, y S. Mbarki, “Extending graphical part of the Interaction Flow Modeling Language to Generate Rich Internet Graphical User Interfaces”, en *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016, pp. 161–167.
[45] C. Bernaschina, S. Comai, y P. Fraternali, “IFMLEdit.Org: Model Driven Rapid Prototyping of Mobile Apps”, en *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, Piscataway, NJ, USA, 2017, pp. 207–208.
[46] M. Krunic, I. Letvencuk, I. Povazan, y V. Krunic, “An approach to model driven development and automatic source code generation of GUI controls”, en *2013 IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY)*, 2013, pp. 63–68.
[47] A. Arusoaie y D. I. Vicol, “Automating Abstract Syntax Tree Construction for Context Free Grammars”, en *2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2012, pp. 152–159.
[48] T. Parr y J. Vinju, “Towards a Universal Code Formatter Through Machine Learning”, en *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, New York, NY, USA, 2016, pp. 137–151.
[49] N. Bouraqadi y D. Mason, “Mocks, Proxies, and Transpilation As Development Strategies for Web Development”, en *Proceedings of the 11th Edition of the International Workshop on Smalltalk Technologies*, New York, NY, USA, 2016, pp. 10:1–10:6.

[50] Y. Maheshwari y Y. R. Reddy, “A Study on Migrating Flash Files to HTML5/JavaScript”, en *Proceedings of the 10th Innovations in Software Engineering Conference*, New York, NY, USA, 2017, pp. 112–116.
[51] I. Sommerville y M. I. A. Galipienso, *Ingeniería del software*. Pearson Educación, 2005.
[52] A. Leff y J. T. Rayfield, “Web-Application Development Using the Model/View/Controller Design Pattern”, en *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*, Washington, DC, USA, 2001, pp. 118–.
[53] L. Torczon y K. Cooper, *Engineering A Compiler*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.
[54] A. Shalloway, J. R. Trott, y J. Trott, *Design Patterns Explained: A New Perspective on Object-oriented Design*. Addison-Wesley, 2002.
[55] K. Williamson, *Learning AngularJS: A Guide to AngularJS Development*. O’Reilly Media, 2015.