Tecnológico Nacional de México

Instituto Tecnológico de Orizaba

División de Estudios de Posgrado e Investigación

Maestría en Sistemas Computacionales

TESIS

Módulo de Modelado IFML para la herramienta ITO's_iBRAIN

PRESENTADO POR:

I.S.C. Eliud Federico De los Santos Vera M14011028

PARA OBTENER EL GRADO DE:

Maestro en Sistemas Computacionales

DIRECTOR DE TESIS:

M.C.E. Beatriz Alejandra Olivares Zepahua

CODIRECTOR DE TESIS:

Dr. Ulises Juárez Martínez

Orizaba, Veracruz, 07/marzo/2023
Dependencia: **División de Estudios de Posgrado e Investigación**
Asunto: **Autorización de Impresión**
OPCION: I

C. ELIUD FEDERICO DE LOS SANTOS VERA
CANDIDATO A GRADO DE MAESTRO EN:
SISTEMAS COMPUTACIONALES
P R E S E N T E.-

De acuerdo con el Reglamento de Titulación vigente de los Centros de Enseñanza Técnica Superior, dependiente de la Dirección General de Institutos Tecnológicos de la Secretaría de Educación Pública y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

" Módulo de Modelado IFML para la herramienta ITO's_IBRAIN."

comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

Cuauhtémoc Sánchez R.

DR. CUAUHTÉMOC SÁNCHEZ RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLOGICO DE ORIZABA
DIV. DE EST. DE POSGRADO E INVEST.

OG-13-F06

Orizaba, Veracruz, 14/febrero/2023
Asunto: **Revisión de trabajo escrito**

C. CUAUHTÉMOC SÁNCHEZ RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
P R E S E N T E.-

Los que suscriben, miembros del jurado, han realizado la revisión de la Tesis del (la) C.

ELIUD FEDERICO DE LOS SANTOS VERA

La cual lleva el título de:

Módulo de Modelado IFML para la herramienta ITO's_IBRAIN.

Y concluyen que se acepta.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

PRESIDENTE:	M.C.E. BEATRIZ ALEJANDRA OLIVARES ZEPAHUA	FIRMA
SECRETARIO:	DR. ULISES JUÁREZ MARTÍNEZ	FIRMA
VOCAL:	DR. JOSÉ LUIS SÁNCHEZ CERVANTES	FIRMA
VOCAL SUP.:	M.C. CELIA ROMERO TORRES	FIRMA

TA-09-18

CARTA DE ORIGINALIDAD

En la ciudad de Orizaba, Veracruz, el día 01 del mes de marzo del año 2023, el que suscribe Eliud Federico De los Santos Vera, alumno del programa de Maestría en Sistemas Computacionales con número de control M14011028 manifiesta que es autor del trabajo de tesis titulado "Módulo de Modelado IFML para la herramienta ITO's iBRAIN" y declaro que el trabajo es original, ya que sus contenidos son producto de mi directa contribución intelectual. Todos los datos y las referencias a materiales ya publicados están debidamente identificados con su respectivo crédito e incluidos en las notas bibliográficas y en las citas que se destacan como tal y, en los casos que así lo requieran, cuento con las debidas autorizaciones de quienes poseen los derechos patrimoniales. Por lo tanto, me hago responsable de cualquier litigio o reclamación relacionada con derechos de propiedad intelectual, exonerando de toda responsabilidad al Tecnológico Nacional de México / Instituto Tecnológico de Orizaba.

Eliud Federico De los Santos Vera

Nombre y firma

CARTA DE CESIÓN DE DERECHOS

En la ciudad de Orizaba, Veracruz, el día 01 del mes de marzo del año 2023, el que suscribe Eliud Federico De los Santos Vera, alumno del programa de Maestría en Sistemas Computacionales con número de control M14011028 manifiesta que es autor del trabajo de tesis bajo la dirección de M.C.E. Beatriz Alejandra Olivares Zepahua y cede los derechos del trabajo de tesis titulado "Módulo de Modelado IFML para la herramienta ITO's iBRAIN" al TecNM/Instituto Tecnológico de Orizaba para su difusión y divulgación, con fines académicos y de investigación.

Queda estrictamente prohibido reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del Tecnológico Nacional de México/Instituto Tecnológico de Orizaba. Este puede obtenerse escribiendo a la siguiente dirección: msc@orizaba.tecnm.mx . Si el permiso se otorga, cualquier usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Eliud Federico De los Santos Vera

Nombre y firma

Agradecimientos

En primer lugar, este proyecto de tesis está dedicado a mis padres y hermano que con sus conocimientos, motivación, consejos, enseñanzas y apoyos he salido adelante y que son lo más importante para mí, siempre han estado en todos mis logros académicos y que a pesar de todo siempre me han apoyado y corregido cuando me he equivocado.

Igualmente le agradezco a mi directora de tesis la M.C.E. Beatriz Alejandra Olivares Zepahua, por alentarme y motivarme para iniciar y terminar la maestría, sus palabras de aliento, las pláticas que compartimos, por ofrecerme su hombro para llorar cuando yo estaba realmente mal emocionalmente, en pocas palabras la maestra Betty es una maravillosa maestra y ser humano que ha dejado marca en mi vida y es mi modelo a seguir para ser un buen profesionista, nunca voy a terminar de agradecer todo el apoyo y confianza que me brindo.

De igual manera le doy las gracias al M.C.C Ignacio López Martínez por haber sido mi tutor durante mi estancia en la maestría, por su consejos y apoyo.

Para mí este proyecto significa la continuación de mi vida y de las innumerables experiencias, que me han hecho crecer y por otro lado, es el comienzo de una nueva etapa profesional.

Le doy la gracias al Tecnológico Nacional de México Campus Orizaba por ofrecer sus instalaciones necesarias para realizar este proyecto de tesis.

Y para concluir le doy las gracias al Consejo Nacional de Ciencia y Tecnología (CONACYT), por el apoyo financiero otorgado en el periodo febrero 2021 – febrero 2023.

Índice General

Índice de tablas

Índice de figuras

Índice de listados

Resumen

ITO’s_iBRAIN es una herramienta que permite generar aplicaciones enriquecidas de Internet, aplicaciones de escritorio y aplicaciones para dispositivos móviles modeladas con IFML, la fuente de la funcionalidad de la herramienta ITO’s_iBRAIN es un archivo XML que se obtiene de la herramienta de modelado WebRatio Web Platform en su versión académica.

WebRatio Web Platform es una herramienta para modelar diagramas navegacionales y de dominio, con simbología de IFML y “ejecutar” dichos diagramas; las versiones de paga de WebRatio proporcionan acceso al código resultante pero la versión comunitaria no. A partir de los diagramas mencionados es posible obtener un archivo XML que es el que alimenta a ITO’s_iBRAIN con el fin de obtener código para aplicaciones enriquecidas de Internet, aplicaciones de escritorio o aplicaciones para dispositivos móviles bajo distintas combinaciones de lenguaje.

WebRatio ha tenido actualizaciones frecuentes, lo que ha provocado cambios en los archivos XML, lo que conlleva el riesgo de que ITO’s_iBRAIN quede inoperable en el momento en que los cambios sean tan drásticos que no sea posible obtener la información básica del modelado (contenedores, eventos, entidades, atributos entre otros). El hecho de que ITO’s_iBRAIN sea independiente de otras herramientas permitirá que siga funcionando en el futuro.

Por lo anteriormente mencionado, en este trabajo se propuso que ITO’s_iBRAIN contara con su propio módulo de modelado de los diagramas navegacional y de dominio IFML, con la posibilidad además de exportar este último, mediante XMI, para su posterior desarrollo como diagrama de clases en otras herramientas UML.

El módulo de modelado se desarrolló con las bibliotecas JavaFX y JAXB; la primera ya estaba incorporada en ITO’s_iBRAIN, aunque fue necesario incluir lo relacionado con el manejo de gráficos (elementos que heredan de la clase *Shape*); por su parte, JAXB es una biblioteca para el manejo de archivos XML, la metodología de desarrollo utilizada fue la Metodología Incremental.

Abstract

ITO's_iBRAIN is a tool to generate Rich Internet Applications, desktop applications and mobiles applications modeled with IFML, the source of ITO's_iBRAIN tool functionality is an XML file obtained from the WebRatio Web Platform modeling tool in its academic version.

WebRatio Web Platform is a tool for modeling navigational and domain diagrams with IFML symbology and "execute" those diagrams; the paid versions of WebRatio provide access to the resulting code but the community version does not. From the aforementioned diagrams it is possible to obtain an XML file that feeds the ITO's_iBRAIN in order to obtain code for Rich Internet Applications, desktop applications or mobiles applications under different language combinations.

WebRatio has had frequent updates, which has caused changes in the XML files, leading to the risk that ITO's_iBRAIN becomes inoperable when the changes are so drastic that it is not possible to obtain the basic modeling information (containers, events, entities, attributes, among others). The fact that ITO's_iBRAIN is independent from other tools will allow it to continue working in the future.

For the above mentioned, this work proposes that the ITO's_iBRAIN tool has its own modeling module for navigational and IFML domain diagrams, with the possibility of exporting the latter, through XMI, for its later development as a class diagram in other UML tools.

The modeling module was developed with JavaFX and JAXB libraries; the first one was already incorporated in ITO's_iBRAIN although was necessary to include graphic elements (inheritance from Shape class); JAXB is a library for handling XML files; the development methodology used was Incremental.

Introducción

IFML (*Interaction Flow Modeling Language*, Lenguaje de Modelado de Flujo de Interacción) sirve para modelar la parte del *front-end* de un sistema y los mecanismos de interacción de éste con el usuario, se centra en la estructura y comportamiento de la aplicación. Pese a su facilidad de uso, sólo se tiene información de tres herramientas que permiten crear modelos con dicho lenguaje.

Al realizar un modelo de cualquier tipo, se hace una representación simplificada de un concepto, objeto o sistema, esta representación incluye las propiedades más relevantes y significativas. En Computación, modelar es importante para comprender mejor la estructura del sistema que se pretenda desarrollar; si bien el modelado no obliga al uso de una herramienta, es mejor contar con una para controlar de una manera más óptima el proceso de desarrollo y, si dicha herramienta es capaz de transformar el modelo en código, se logran beneficios tanto para el equipo de trabajo como para la calidad del sistema resultante.

La herramienta ITO's_iBRAIN (**I**nstituto **T**ecnológico de **O**rizaba **'s** ***I****FML* ***B****ased* ***R****ich Internet* ***A****ppl****i****cation Ge****n****erator*) cumple con el objetivo de transformar modelos realizados a nivel de análisis en código funcional que permite al desarrollador verificar que los requisitos de interacción del sistema se han interpretado bien y corregir desviaciones en etapas tempranas del proyecto; como su nombre lo indica, se basa en modelos IFML, dichos modelos se realizan con la herramienta comercial WebRatio (tiene una versión académica sin costo); pero, al depender de WebRatio, corre el riesgo de quedar obsoleta o de requerir mucho esfuerzo para mantener la consistencia.

El presente proyecto tiene como objetivo hacer independiente a ITO's_iBRAIN para asegurar su continuidad de uso, para lograr dicho objetivo se propuso desarrollar un módulo de modelado IFML incorporándolo a ITO's_iBRAIN.

El trabajo está dividido cinco capítulos: en el capítulo uno se describen los conceptos más relevantes, el objetivo general y los específicos, el planteamiento del problema y la justificación; en el capítulo dos se abarca el estado de la práctica, el análisis comparativo correspondiente y la propuesta de solución; el capítulo tres se muestra la aplicación de la metodología seleccionada para el desarrollo del módulo; en el capítulo cuatro se muestran

los resultados aplicando un caso de estudio y, por último, en el capítulo cinco están las conclusiones y recomendaciones sobre el trabajo realizado así como futuras mejoras al módulo obtenido.

Capítulo 1 Antecedentes

En este capítulo se presenta el marco teórico del proyecto, donde se describen los conceptos básicos relacionados con el mismo, así como los objetivos, la descripción del problema y la justificación del proyecto.

1.1 Marco teórico

A continuación, se explican los conceptos más importantes relacionados con el trabajo presentado.

1.1.1 Modelado

Un modelo representa una simplificación de un concepto, de un objeto o de un sistema y explica propiedades específicas del mismo, por ejemplo: el comportamiento, la estructura, la relación de una entidad con otras entidades. Los desarrolladores de sistemas modelan para comprender mejor el sistema, para visualizar lo que va hacer el sistema, especificar estructura, comportamiento, y guiar su construcción. Además, modelar apoya al desarrollador a comunicarse con el cliente o usuario, a explorar posibles soluciones a un problema durante el análisis y el diseño y a documentar las decisiones tomadas durante el desarrollo[1].

1.1.2 Herramientas de modelado

En la Ingeniería de Software se han propuesto varios procesos, metodologías y herramientas para facilitar y estandarizar el proceso de desarrollo. Entre estas las herramientas se encuentran las que son enfocadas a Ingeniería de Software Asistida por Computadora (CASE) que soportan varios pasos del ciclo de vida de las metodologías para el desarrollo de software con el objetivo mejorar la productividad y eficiencia, reduciendo tiempos y costos en el desarrollo de un proyecto de software, ayudando a mejorar la

calidad, lo que permite al desarrollador documentar y modelar una aplicación desde los requisitos y el diseño hasta la implementación y pruebas [2].

1.1.3 Desarrollo Dirigido por Modelos

El desarrollo de software basado en modelos (MDD, *Model-Driven Software Development*) es un método de trabajo que crea grandes expectativas como una alternativa excepcional a los métodos tradicionales de producción de software. MDD proporciona una forma de facilitar el desarrollo y mantenimiento de sistemas o aplicaciones de software mediante el uso de modelos como el trabajo principal del proceso de desarrollo. MDD es la evolución natural de la Ingeniería de Software basada en modelos, enriquecida mediante el agregado de transformaciones automáticas entre modelos. MDD utiliza los modelos para avanzar en las tareas de comprensión, diseño, construcción, pruebas, implementación y modificación de sistema. Algunas de las ventajas que tiene MDD son:

- Incremento en la productividad.
- Adaptación a los cambios tecnológicos.
- Adaptación a los cambios en los requisitos.
- Consistencia.
- Reutilización, mejoras en la comunicación entre los desarrolladores.
- Captura de la experiencia.
- Los modelos son productos de larga duración[3].

1.1.4 Lenguaje de Modelado de Flujo de Interacción (IFML)

IFML (*Interaction Flow Modeling Language*, Lenguaje de Modelado de Flujo de Interacción) es un lenguaje de modelado gráfico que se centra en la estructura y el comportamiento de una aplicación tal y como la ve el usuario final, es decir, representa el contenido del *front-end* y los eventos disponibles en la interfaz de usuario.

IFML cubre aspectos de la interfaz gráfica como son la estructura de la vista, el contenido y las transiciones, todo concentrado dentro del diagrama navegacional. Por otra parte, también se considera el diagrama de dominio, en el que se realiza la especificación de la información relevante de la aplicación modelada, su importancia está en que la información del dominio coincida con algunos aspectos de la interfaz, por ejemplo, en los formularios. IFML no considera el aspecto gráfico en lo relacionado con colores, tipos de letra, distribución entre otros.

IFML modela aplicaciones para los siguientes dominios:

- Aplicaciones Web tradicionales basadas en HTML+ HTTP.
- Aplicaciones enriquecidas de Internet soportadas por el estándar HTML5.
- Aplicaciones para dispositivos móviles.
- Aplicaciones cliente-servidor.
- Aplicaciones de escritorio.
- Interfaces usuario-máquina integradas para el control de las aplicaciones.
- Aplicaciones multicanales y contextualizadas[4].

1.1.5 Lenguaje Unificado de Modelado (UML)

UML es el Lenguaje Unificado de Modelado del OMG (*Object Management Group*, Grupo de Administración de Objetos) que especifica, visualiza y documenta modelos de sistemas en general, no sólo de sistemas de software, aunque estos últimos son su principal objetivo.

La versión 2.0 de UML cuenta con trece diagramas agrupados en tres tipos:

1. Diagramas estructurales:
2. Diagrama de clases.
   2. Diagrama de componentes.
   3. Diagrama de despliegue.
   4. Diagrama de objetos.
   5. Diagrama de paquetes.
   6. Diagrama de estructura compuesta.

3. Diagramas de comportamiento:
    7. Diagrama de actividades.
    8. Diagrama de casos de uso.
    9. Diagrama de máquina de estados.
4. Diagramas de interacción:
    10. Diagrama general de interacciones.
    11. Diagrama de comunicación.
    12. Diagrama de secuencia.
    13. Diagrama de tiempos.

Con los diagramas ya mencionados, es posible modelar cualquier tipo de aplicación independientemente del tipo de *hardware* o plataforma o lenguaje de programación empleado para el desarrollo [5].

Diagrama de clases

El diagrama de clases es un tipo de diagrama de UML que muestra las clases de la aplicación y sus relaciones; se considera como el modelo más importante para el desarrollo de una aplicación o sistema orientado a objetos y los demás diagramas UML se toman como complementos [6].

El diagrama de clases es un diagrama de estructura UML que muestra el sistema diseñado a nivel de clases e interfaces, características, restricciones y relaciones: asociaciones, dependencias y generalizaciones entre otras [7].

1.1.6 WebRatio Web Platform

WebRatio Web Platform es un entorno de desarrollo que usa el estándar IFML para modelar el flujo de interacción entre el usuario y la aplicación a desarrollar (*front-end*) y modelar los conceptos del negocio (parte del *back-end*), logrando una combinación entre la simplicidad y funcionalidad. Si bien IFML se utiliza en el modelado de requisitos, WebRatio incluye como parte de sus capacidades la relación con la base de datos, pues su objetivo principal es obtener código funcional a partir del modelado. La herramienta cuenta con una versión comunitaria sin costo, una versión académica sin costo asociada a una

cuenta de correo educativa y versiones profesionales con costo; sólo la última deja disponible el código generado para su posterior edición.

Algunos de los aspectos en que se enfoca la herramienta son:

- **Modelado del dominio:** Soporta el diseño haciendo uso de las propiedades de los diagramas de clases de UML.
- **Modelado navegacional:** Permite diseñar el modelo navegacional con una implementación propia de los elementos de IFML[4].

1.1.7 Lenguaje de Marcado Extensible (XML)

XML (*eXtensible Markup Language*, Lenguaje de Marcado Extensible) es un lenguaje de marcado que define reglas para la codificación de documentos; se cataloga como un lenguaje extensible, ya que a los desarrolladores se les permite definir sus elementos y su objetivo principal es ayudar a los sistemas de información a compartir datos de manera estructurada [8].

1.1.8 XML de Intercambio de Metadatos (XMI)

XML de Intercambio de Metadatos (XMI) es un formato XML para permitir el intercambio de diagramas entre distintas herramientas de modelado, fue creado inicialmente para UML, pero también soporta diagramas de otros lenguajes de modelado como IFML; define los siguientes aspectos:

- La representación de objetos en elementos y atributos XML.
- Los mecanismos estándar para vincular objetos dentro del mismo archivo o entre archivos.
- La validación de documentos XMI utilizando esquemas XML.

XMI describe soluciones a los puntos anteriormente mencionados mediante la especificación de reglas de producción EBNF (*Extended Backus–Naur Form*) para crear documentos XML y esquemas que comparten objetos de forma coherente [9].

1.1.9 ITO's IFML *Based Rich Internet Applications Generator* (ITO's_iBRAIN)

Es un generador de Aplicaciones Enriquecidas de Internet, aplicaciones para dispositivos móviles y aplicaciones de escritorio desarrollado por Estévez[10], Rosales [11] y otros que, partiendo de modelos IFML, obtiene código funcional con varias combinaciones de tecnologías como PrimeFaces + JavaServer Faces, jQuery + PHP, Angular + PHP y React + Django + Python, JavaFX + Java, Apache Cordova + PHP entre otras. Las aplicaciones resultantes del uso del generador cuentan con características estipuladas por el usuario en lo que respecta a la distribución de su contenido, temas y navegabilidad; el contenido de la interfaz gráfica, la interacción permitida con el usuario y los conceptos principales del negocio provienen de modelos IFML diseñados en WebRatio Web Platform exportados como un archivo XML [11].

1.2 Situación tecnológica económica y operativa de la empresa

El Instituto Tecnológico de Orizaba es una institución de educación superior orientada al desarrollo de la ciencia y la tecnología mediante la formación de recursos humanos y la investigación de dichas áreas para el fortalecimiento académico y social. Dicha institución es parte del Tecnológico Nacional de México.

Dentro del Instituto Tecnológico de Orizaba se encuentra la División de Estudios de Postgrados e Investigación (DEPI), que se encarga de formar profesionales altamente preparados, capaces de aplicar las tecnologías de cómputo para resolver problemas y aprender de forma autónoma el nuevo conocimiento que se genera diariamente.

1.3 Planteamiento del problema

Por su naturaleza, IFML se utiliza en las etapas tempranas del desarrollo; tanto WebRatio como IFMLEdit.org lo implementan considerando un desarrollo dirigido por modelos dado que ambas herramientas permiten transformar el o los modelos en código, lo que permite que el desarrollador valide con el usuario final que el modelo efectivamente refleja sus

requerimientos, el complemento de modelado para Eclipse sólo permite hacer el diseño, pero no genera código. En este sentido, en [2] se realizó un análisis de las herramientas para IFML con los siguientes criterios de evaluación: 1) representa el concepto de IFML siguiendo completamente la notación estándar, 2) representa el concepto de IFML con variación en el nombre o representación gráfica, pero con similitud con el estándar, 3) representa el concepto de IFML de una manera poco clara y 4) no tiene soporte para modelar el concepto de IFML o modela de una forma que no corresponde al estándar; con estos criterios, se identificó que WebRatio es la herramienta más completa de las tres existentes para el soporte a IFML pero no respeta completamente el estándar (en varios casos los símbolos que utiliza son diferentes al estándar establecidos o incluso no se consideran, por ejemplo las ventanas modales de IFML se indican como llamada AJAX en WebRatio).

WebRatio tiene una versión gratuita que, aunque permite ver el resultado del modelo en ejecución, no entrega el código resultante para que el equipo de desarrollo lo complemente, la obtención del código fuente sólo es posible en las versiones de paga, lo que descarta su uso en PYMEs. En el caso de [12], sí se obtiene el código resultante, pero sólo es posible generar el modelo navegacional y no el modelo de dominio.

En los trabajos de tesis [10] y [11] se creó la herramienta ITO's_iBRAIN (Instituto Tecnológico de Orizaba's *IFML Based Rich Internet Applications Generator*) que permite generar aplicaciones enriquecidas de Internet, aplicaciones de escritorio y aplicaciones para dispositivos móviles modeladas con IFML, recibe el modelo generado en WebRatio (diagrama navegacional y diagrama de dominio) y lo transforma en el código de la aplicación equivalente de acuerdo a la selección de lenguajes realizada. Como se menciona en secciones anteriores, WebRatio ha tenido actualizaciones a lo largo del tiempo, lo que pone en riesgo a ITO's_iBRAIN debido a la dependencia del modelo, por lo que es importante independizarla incluyendo ahora el módulo de modelado.

1.4 Objetivo general y objetivos específicos

En las dos sub-secciones siguientes se detalla el objetivo general para el proyecto propuesto a desarrollar, así como sus objetivos específicos.

1.4.1 Objetivo general

Desarrollar un módulo que permita realizar los modelos navegacional y de dominio IFML para complementar la herramienta ITO's_iBRAIN.

1.4.2 Objetivos específicos

- Analizar las definiciones formales de IFML (*Interaction Flow Modeling Language*) publicadas por el OMG para identificar los requerimientos del módulo de modelado.
- Analizar las herramientas existentes de modelado IFML para identificar las mejores prácticas e incorporarlas al módulo de modelado.
- Analizar la definición formal de XMI para UML, específicamente lo relacionado con diagrama de clases, para garantizar la exportación del modelo de dominio.
- Modificar la arquitectura de la herramienta ITO's_iBRAIN para incorporar el módulo de modelado con el mínimo de afectaciones.
- Codificar el módulo de modelado para la herramienta ITO's_iBRAIN.
- Realizar al menos dos casos de estudio para validar el comportamiento del módulo en la herramienta.

1.5 Justificación

IFML (*Interaction Flow Modeling Language*, Lenguaje de Modelado de Flujo de Interacciones) es un estándar del OMG (*Object Management Group*, Grupo de Administración de Objetos) que permite modelar el *front-end* de una aplicación que tenga una interfaz gráfica de alta interacción con el usuario, independientemente de los detalles técnicos de su implementación, es decir, es independiente de la plataforma y lenguaje de codificación. La versión 1.0 de IFML se liberó en 2015, el lenguaje es poderoso, sin embargo estudios como [13] y el propio sitio oficial sólo muestran la existencia de 3 herramientas de modelado: *plugin* para Eclipse, WebRatio e IFMLEdit.org; de ellas, únicamente WebRatio ha tenido actualizaciones frecuentes.

MDD se basa en la capacidad de transformar modelos hasta llegar al código para agilizar el desarrollo sin sacrificar la calidad. En [14] y [15] se describe la importancia de usar modelos que se transformen en código con el fin de aumentar la productividad de los equipos de desarrollo y disminuir el tiempo de entrega al usuario final.

Dadas las ventajas de IFML pero su limitada implementación en la industria al contar con sólo tres herramientas documentadas para su modelado, es importante incrementar el número de herramientas que soporten tanto IFML como la idea de transformación de modelos en código, preferentemente sin costo o con costo muy bajo para apoyar a las PYMEs dedicadas al desarrollo de sistemas; de ahí la relevancia de ITO's_iBRAIN y la necesidad de independizarla de WebRatio para mantener su funcionalidad.

Capítulo 2 Estado de la práctica

A continuación, se presenta una breve descripción de los trabajos más importantes consultados para este proyecto.

2.1 Trabajos relacionados

En [2] se indicó que el diseño del *front-end* es un proceso costoso y que la situación se agrava debido a la escasez de métodos automatizados de producción orientados al software que sean convencionales, lo cual provoca una baja en la reutilización de artefactos de software por medio de interfaces y con una sobrecarga que asegure la posibilidad de que las aplicaciones se generen en multiplataforma. El estándar IFML, un lenguaje de modelado de interacción a un nivel independiente de la plataforma, proporciona un conjunto estable de conceptos utilizables para caracterizar los aspectos esenciales de la interacción del usuario con la interfaz de una aplicación de software y es una respuesta a la problemática de diseño del *front-end*. Los principales elementos a modelar en un diagrama de flujo de interacción y cubiertos por el estándar son: la estructura para la vista, el contenido para la vista, eventos que representan sucesos generados para la interacción del usuario, transiciones de eventos que describen las afectaciones en la interfaz de usuario de un evento, referencias a acciones desencadenadas por eventos del usuario y enlace de parámetros. En Ingeniería de Software se han establecido varios procesos, metodologías y herramientas para estandarizar y facilitar el proceso de desarrollo; entre estas últimas se encuentran las herramientas CASE para mejorar la productividad y automatizar la producción, junto con la reducción de tiempos y costos en el desarrollo de un proyecto de software. Para el caso de IFML, como herramientas CASE se considera a WebRatio, el *plugin* de Eclipse e IFMLEdit.org.

En [13] se indicó que las aplicaciones web han ganado un notable crecimiento en los últimos años, estas aplicaciones con interfaces de usuario refinadas van desde la visualización simple de datos hasta aplicaciones interactivas extremadamente complejas

como sistemas de información, motores de búsqueda o redes sociales entre otras. La ingeniería dirigida por modelos (MDE) busca simplificar la arquitectura de un sistema mediante la introducción de modelos que proporcionan diferentes niveles de abstracción. El estándar de la OMG, *Interaction Flow Modeling Language* (IFML) se presentó como primera versión en marzo de 2013. IFML presenta interacciones, produce modelos de cualquier contenido de aplicación de software *front-end*, captura su comportamiento de control sin ninguna referencia a plataformas específicas. IFML se utiliza en el modelado de la interfaz de usuario de aplicaciones de escritorio, web móvil y cliente-servidor. En este estudio se consideraron 22 investigaciones reportadas de 2014 a 2017 para realizar la revisión sistemática de la literatura (SLR) y se formularon las siguientes preguntas de investigación: 1) ¿Dónde se aplica IFML? R: las aplicaciones se clasificaron en 4 categorías, a) aplicaciones web (8), b) aplicaciones para dispositivos móviles (9), c) aplicaciones de escritorio (1) y d) otras (4); 2) ¿Con qué otros estándares de modelado se ha integrado IFML? R: se ha reportado la integración con UML, ODM, SOA ML, Mob ML y Web ML; 3) ¿Cuáles son los principales dominios/áreas donde se practica con frecuencia IFML? R: El comercio electrónico es el dominio más reportado junto con el desarrollo de la interfaz de usuario; 4) ¿Cuáles son las herramientas disponibles para el modelado, transformación y validación del modelo IFML? R: como herramientas de modelado se identificaron el *plugin* de Eclipse, IFMLEdit.org y WebRatio, como herramientas de transformación de modelado: plantillas Xtend, QVTO, Acceleo, WebRatio, IFMLEdit.org y ATL, como herramientas de validación del modelado: WebRatio e IFMLEdit.org; 5) ¿Es IFML maduro para utilizarlo en la industria? R: Su madurez se midió mediante el soporte que existe en sus herramientas y su aplicación en diversos dominios.

En [14] se describió que el desarrollo web y móvil, la amplia gama de plataformas de codificación así como los diversos tamaños de pantallas de dispositivos requiere la capacidad de evolucionar rápidamente y evaluar múltiples versiones de tecnologías. En el ambiente móvil se cuenta con marcos de trabajo o entornos de desarrollo que obtienen como salida aplicaciones multiplataforma como *Appcelerator Titanium*, IBM *MobileFirst Platform Foundation, PhoneGap* y *AppBuilder*. Un enfoque alternativo aprovecha el

paradigma del desarrollo dirigido por modelos (MDD), éste se diferencia del desarrollo multiplataforma y los IDE visuales en que traslada el esfuerzo de desarrollo del código a los modelos y transformaciones, requiere una implementación de prototipos tempranos para mitigar la divergencia de especificaciones mediante la implementación típica. Los enfoques recientes de MDD específicos se basan principalmente en DSL textuales, transformaciones y lenguajes de modelado especificados informalmente y se encuentran en un estado prototípico. En ese sentido, IFMLEdit.org, es una herramienta MDD en línea, gratuita, de código abierto, para la especificación y generación automática de prototipos rápidos de aplicaciones web y aplicaciones para dispositivos móviles a partir de diagramas IFML. El Lenguaje de Modelado de Flujo de Interacción (IFML) es un estándar del OMG que permite una descripción de interfaces gráficas de usuario, independiente de la plataforma, para dispositivos como computadoras de escritorio, portátiles, teléfonos móviles y tabletas. IFMLEdit.org soporta que los desarrolladores modifiquen o importen modelos IFML, admite una transformación de mapeo semántico de IFML a PCN, realiza la generación de modelo a texto de prototipos de aplicaciones móviles y web totalmente funcionales.

En [15] se mostraron los resultados que brindó una encuesta aplicada a 47 instructores relacionados con temas de Ingeniería de Software. Las preguntas abordaron el contenido del curso, las herramientas y tecnologías utilizadas así como los factores positivos y negativos que afectan los resultados del aprendizaje. Se analizaron los resultados y se resumieron los hallazgos clave con el potencial de mejorar la práctica de la enseñanza y el aprendizaje de tales temas. El trabajo estaba destinado a contribuir a la definición del estado de la práctica de la enseñanza de modelado y la ingeniería dirigida por modelos, desde el punto de vista del profesor. Los objetivos de esa encuesta fueron analizar el estado actual de la enseñanza de modelos, comprender mejor los aspectos distintivos que caracterizan un curso exitoso y lo que no funciona, e identificar los resultados de aprendizaje típicos. La información general se utilizó para describir el contexto de cada curso e incluyó: nombre, nivel de los estudiantes, si es optativo u obligatorio, el número de veces que se ha impartido el curso, el formato del curso, número de créditos, número de estudiantes, modos de evaluación y resultados del aprendizaje. Esto identificó qué

actividades específicas llevan a cabo los estudiantes con los modelos, por ejemplo, crear un diagrama de estado a partir del lenguaje natural, verificar las propiedades del sistema, entre otros. Los resultados de la encuesta mostraron que el 52.8% de los cursos fueron optativos y 47.2% obligatorios. La mayoría de los cursos usan los modelos de Ingeniería de Software e Ingeniería de Lenguajes como complemento a otros dominios de aplicación como robótica, IoT, *e-learning*, hogar inteligente y salud. Los modelos se utilizan para el diseño, la comunicación y la documentación iniciales. Por lo general, se crean a partir de requisitos, incluidos escenarios de lenguaje natural. Se crean tanto diagramas estructurales (por ejemplo, de clases) como diagramas de comportamiento (por ejemplo, estado o secuencia) y se genera código a partir de modelos. Los estudiantes normalmente aprenden cómo desarrollar un ecosistema de modelado que consta de meta-modelos e instancias de dichos meta-modelos, uno de los puntos más importantes es la transformación de modelo a modelo, como también las transformaciones de modelo a archivos de textos (generación de código). La creación de perfiles UML se utiliza ampliamente en los cursos de ingeniería del lenguaje, mientras que UML se utiliza en cursos de ingeniería de software (por ejemplo, arquitectura de software, diseño de software y diseño orientado a objetos). Se encontró que aproximadamente el 30% de los cursos que usaban EMF, OCL o ATL y casi el 50% de los cursos que usaban Xtext, Acceleo o Xtend identificaron como problema que las herramientas no son lo suficientemente maduras. Estos resultados parecen coherentes con el hecho de que las herramientas más antiguas serían más maduras.

En [16] se planteó que la ingeniería basada en modelos (MDE) aborda la complejidad del software elevando el nivel de abstracción del mismo y facilitando la automatización de generación de código y verificación de software; también indica que a los modeladores a menudo le resulta cognitivamente difícil crear, editar y depurar modelos. Las herramientas normalmente no se adoptan o satisfacen las necesidades de los usuarios porque los diseñadores de herramientas: a) no han identificado ni comprendido las dificultades y desafíos de los usuarios, b) no han tenido en cuenta los factores de cognición humana que explicara las dificultades y desafíos de los usuarios y c) han realizado pocas evaluaciones empíricas de la eficiencia de sus herramientas para apoyar a los usuarios. Se realizó un estudio para conocer los desafíos más severos de los modeladores cuando se usan

herramientas de modelado; específicamente, se enfocaron en identificar los desafíos cognitivos que enfrentan los modeladores cuando diseñan modelos estructurales y de comportamiento de sistemas de software, como lo ejemplifican los diagramas UML. Se reclutó a 18 sujetos que tenían el suficiente conocimiento sobre UML y experiencia en el uso de al menos una herramienta de modelado. Los resultados revelaron los desafíos predominantes de los modeladores al usar herramientas de modelado, entre los cuales están a) recordar información contextual e b) identificar y corregir errores e inconsistencias, tales son los aspectos más críticos y son los que más necesitan la consideración de los proveedores de herramientas. De acuerdo con los desafíos del modelado, pocos trabajos apuntan a solucionar los desafíos de la comprensión del modelo, algunos otros trabajos proponen nuevas técnicas de herramientas para ayudar a los usuarios a desarrollar modelos. En otros trabajos se entrevistan a varios sujetos para comprender su percepción de los desafíos cognitivos en la resolución de inconsistencias e identificar los factores cognitivos determinantes, pero no analizan a los sujetos cuando no utilizan ninguna herramienta. La mayoría de los modelos que se están desarrollando en la práctica industrial se basan en las notaciones UML o similares a UML, ya que UML se ha convertido en el estándar para modelar sistemas de software.

También en [17] se describió que la ingeniería dirigida por modelos (MDE) aborda la complejidad del software elevando el nivel de abstracción que especifica el sistema y agiliza la autorización para generar código. Los modelos son los artefactos centrales en MDE que utilizan notaciones gráficas o textuales para aumentar el nivel de comprensión y razonamiento en un sistema de software. Sin embargo, a los modeladores a menudo les resulta cognitivamente difícil editar y depurar modelos y dedican mucho esfuerzo a estas tareas. Aunque las herramientas MDE existentes proponen interfaces de usuario (UI) útiles para abordar cuestiones técnicas como la semántica del lenguaje, carecen de una consideración sistemática de las métricas de UX, como las preferencias de los usuarios y los desafíos cognitivos. Los editores gráficos proporcionan diferentes elementos gráficos y notaciones (por ejemplo, iconos) que reflejan el estado interno de un elemento del modelo. Varias herramientas de modelado emplean estas técnicas de visualización para ayudar a las capacidades visuales y espaciales de los modeladores. Algunas herramientas de

investigación introducen interfaces orientadas a tareas para ayudar a los usuarios a recordar la información contextual relevante para su tarea de modelado actual. La mayoría de las herramientas existentes brindan a los usuarios características que mejoran la comprensibilidad y navegabilidad de los artefactos sin tener en cuenta la semántica de los artefactos o las relaciones entre ellos. Aún existen desafíos cognitivos que las herramientas no abordan (por ejemplo, la memoria de trabajo del usuario y el estilo de aprendizaje). Las herramientas MDE actuales van a mejorarse para reducir el esfuerzo de editar y depurar modelos, así como mejorar la calidad de los modelos mediante la incorporación de técnicas que tienen en cuenta los factores cognitivos humanos. Las contribuciones generales que se esperaron son: a) un análisis de los modeladores y sus tareas, para conocer las principales dificultades y desafíos que enfrentan al usar herramientas de modelado similares a UML, b) avances en herramientas que tienen como objetivo abordar las dificultades y desafíos de los modeladores mejorando las métricas de UX de las herramientas MDE existentes, c) proporcionar más información sobre cómo interactúan los modeladores con las herramientas de modelado y d) cómo explotar las referencias hacia la psicología cognitiva con base en las métricas que se aplican de UX para mejorar las aplicaciones de interacción.

En [18] se describió que el diseño de sistemas interactivos se apoya a través de personas, modelado de tareas, modelado de interacción e interfaz. En los criterios de calidad de uso, la usabilidad es un factor que contribuye al desarrollo de soluciones interactivas. Según la norma ISO/IEC 25010, la usabilidad se define como la "capacidad del software para entenderlo, utilizarlo y aprenderlo, y también su capacidad para complacer al usuario, cuando se utiliza en condiciones específicas". Es importante considerar la usabilidad correctamente durante los pasos del diseño de un sistema y posteriormente propagarlo en la interfaz de usuario. Por medio de IFML, los encargados de desarrollar la aplicación modelan el contenido y comportamiento esperado de la interfaz pero no se refleja directamente la usabilidad. El estudio fue planificado con el objetivo principal de verificar si los mecanismos de usabilidad son percibidos por los participantes en los modelos IFML y si estos mecanismos se propagan a los prototipos; en él participaron 14 estudiantes de pregrado que ya habían tomado cursos de interacción humano-computadora, introducción a la ingeniería de software, estaban tomando análisis y diseño de sistemas y tenían

conocimientos previos sobre modelado con UML; quienes participaron en la creación de los modelos también lo hicieron en la construcción del prototipo correspondiente. Los resultados del estudio determinaron que el prototipo no reflejó los mecanismos de usabilidad esperados porque no se reflejan claramente con IFML; por lo tanto, es deseable complementar, durante el diseño, los modelos IFML originales con otros elementos para especificar determinados requisitos de usabilidad.

En [19] describen que el ciclo de vida de desarrollo de software (SDLC), el análisis de requerimientos es una fase crítica. Los requisitos analizados inicialmente sirven como entrada para las siguientes fases del SDLC. Por lo general, los requisitos se obtienen de los usuarios en lenguaje natural, ya que promueve la libertad de expresión. Sin embargo, también crea muchos problemas para diferentes partes interesadas debido a su naturaleza propensa a errores y ambigüedades porque los requisitos escritos en lenguaje natural sencillo se interpretan de diferentes maneras. Para gestionar ese tipo de cuestiones, se desarrollaron diferentes modelos y/o prototipos para realizar la validación inicial de requisitos. En ese sentido, IFML está mostrando características prometedoras para desarrollar modelos/prototipos iniciales de interfaces de usuario a partir de requisitos. El presente trabajo presenta un marco novedoso para generar automáticamente modelos IFML a partir de requisitos iniciales de texto sin formato mediante el empleo de características de la PLN. El marco propuesto minimiza el proceso desde la obtención de requisitos hasta la validación temprana, ya que los modelos IFML se generan instantáneamente a partir de los requisitos del lenguaje natural. Las principales contribuciones del trabajo son las siguientes: diseña un algoritmo completo utilizando técnicas de NLP para extraer elementos IFML de los requisitos del lenguaje natural. En particular las reglas NPL se desarrollaron para extraer automáticamente contribuciones IFML como *ViewComponent*, *ViewContainer*, *Event* y *Action* del texto en lenguaje natural. La herramienta “texto a IFML” (T2IF) se implementa en Visual Studio utilizando el lenguaje C# sobre la base del algoritmo propuesto y las reglas de NPL. T2IF proporciona una interfaz de usuario sencilla para la generación de elementos IFML a partir de requisitos textuales. Se evalúa la aplicación del marco propuesto a través de dos casos de estudio de referencia. IFML tiene como objetivo describir el comportamiento y la estructura de la aplicación según la

perspectiva del usuario. IFML admite su integración con otros modelos para especificar completamente las características de la aplicación. A partir de la literatura se identifica que IFML es un paso significativo para el modelado y transformación de las interfaces de usuario. Por lo tanto, se investigan con frecuencia en varias áreas como web, móviles y escritorio, para el desarrollo de interfaces de usuario. El requerimiento obtenido del usuario está escrito en lenguaje natural. El proceso de extraer información adecuada de los datos recopilados preliminarmente es, de hecho, una tarea muy errónea y requiere intervención humana. En particular para esta cuestión se desarrollaron reglas, basadas en la revisión de la literatura, para obtener los requisitos de la interfaz de usuario en texto de lenguaje natural adecuado, es decir en inglés. Las reglas generales para escribir requisitos son: las oraciones deben ser cortas y concisas. Se deben escribir los requisitos en oraciones simples y activas. El requisito debe estar libre de oraciones no funcionales y negativas. El requisito debe redactarse desde el punto de vista del usuario. El requisito de la interfaz de usuario debe escribirse vita por vista o página por página. Estas reglan ayudan a obtener los requisitos de las interfaces de usuario en un idioma inglés adecuado para su posterior procesamiento. Los resultados prueban que el marco propuesto es capaz de generar modelos IFML con alta precisión, aunque el tamaño de los casos de estudio es ligeramente pequeño para realizar una evaluación completa y objetiva. En realidad, se seleccionaron casos de estudio del documento en línea de la especificación del estándar IFML para evitar sesgos. El objetivo fue demostrar la efectividad del marco propuesto y se logró de manera justa a través de casos de estudio dados. El marco propuesto es el primer y significativo paso hacia la automatización del modelado IFML a partir de los requisitos del lenguaje natural. Dado que IFML ahora se aplica con frecuencia para simplificar el desarrollo de interfaces de usuario para diferentes sistemas, el marco propuesto es muy beneficioso para tanto la industria y lo académico.

2.2 Análisis comparativo

En la ***Tabla 1*** se presenta una comparativa de los trabajos analizados anteriormente, con el fin de identificar las diferencias, similitudes y resultados obtenidos.

Tabla 1 Análisis comparativo de los artículos relacionados

ARTÍCULO	PROBLEMA	OBJETIVO	RESULTADO	TECNOLOGÍAS/ HERRAMIENTAS	ESTADO
S. Estevéz *et al* [2]	La elaboración de Aplicaciones Enriquecidas de Internet (RIAs) es una tarea compleja, por lo cual su diseño e implementación consumen una gran cantidad de tiempo	Analizar las herramientas de modelado para IFML actualmente disponibles en el mercado: WebRatio Web Platform, el complemento de modelado IFML para Eclipse e IFMLEdit.org.	Comparativa de herramientas de modelado IFML que permitió elegir a WebRatio como herramienta para trabajos posteriores.	WebRatio Web Platform, el complemento para Eclipse e IFMLEdit.org	Finalizado
M. Hamdani, W. H. Butt, M. W. Anwar, and F. Azam [13]	El diseño de interfaces *front-end* en aplicaciones de software es un proceso complejo.	Examinar las aplicaciones de IFML.	Se identificaron cuatro áreas principales en las que IFML se aplica con frecuencia (aplicaciones móviles, web, otras y escritorio) y herramientas IFML que soportan modelado (3) y transformación de modelos (6).	No aplica	Finalizado

ARTÍCULO	PROBLEMA	OBJETIVO	RESULTADO	TECNOLOGÍAS/ HERRAMIENTAS	ESTADO
C. Bernaschina, S. Comai, and P. Fraternali [14]	La amplia gama de diferentes plataformas para web y el desarrollo de aplicaciones móviles requiere la creación rápida de prototipos y la evaluación de múltiples versiones	Describir IFMLEdit.org, una herramienta MDD en línea, de código abierto, para la especificación y generación automática de prototipos rápidos de aplicaciones web y móviles a partir de especificaciones IFML.	Describir un enfoque MDD para la generación automática de aplicaciones móviles y web multiplataforma a partir de especificaciones IFML	IFMLEdit.org	Finalizado
F. Ciccozzi *et al* [15]	Identificar los métodos de enseñanza orientados al modelado desde el punto de vista del instructor	Analizar el estado actual de la enseñanza de modelos, comprender mejor los aspectos distintivos que caracterizan un curso exitoso y lo que no funcionara, e identificar los resultados de aprendizaje típicos y discutirlos con respecto a los necesarios.	Una descripción estructurada del estado de la práctica dentro de la enseñanza del modelado y la ingeniería basada en modelos (desde el punto de vista del instructor).	No aplica	Finalizado
P. Pourali and J. M. Atlee [16]	A los modeladores a menudo les resulta cognitivamente difícil crear, editar y depurar modelos, y dedican mucho esfuerzo a estas tareas	Identificar las dificultades más destacadas que los usuarios enfrentaran al desarrollar diagramas de clase y de máquina de estado utilizando herramientas	Se identificó que las mayores dificultades de los usuarios están en (1) recordar información contextual e (2) identificar y corregir errores e	StarUML, ArgoUML, Visual Paradigm, MagicDraw, Umple, Papyrus, Astah	Finalizado

ARTÍCULO	PROBLEMA	OBJETIVO	RESULTADO	TECNOLOGÍAS/ HERRAMIENTAS	ESTADO
		de modelado UML.	inconsistencias.		
P. Pourali [17]	La mayoría de las herramientas se han centrado en abordar aspectos técnicos (por ejemplo, comprobar la semántica del lenguaje) más que en la usabilidad.	Observar, identificar y comprender los desafíos cognitivos de los usuarios al realizar sus tareas utilizando herramientas de modelado.	Los desafíos más severos de los modeladores al usar herramientas de modelado son: 1) Recordar la información contextual y 2) Localizar, comprender y corregir errores en los modelos.	Focus + Context Transition Editor	Finalizado
R. Queiroz, A. B. Marques, A. Lopes, E. Oliveira, and T. Conte[18]	No se aborda el modelado de interfaces junto con la usabilidad por lo que para de esta última se pierde antes de implementarse.	Realizar un estudio empírico para evaluar cómo los usuarios perciben la usabilidad a través de modelos IFML y si la usabilidad se propaga de los modelos a un prototipo de interfaz.	No todos los aspectos de la usabilidad se perciben y propagan fácilmente en la interfaz a través de modelos IFML	No aplica	Finalizado
M. Hamdani, W. H. Butt, M. W. Anwar, I. Ahsan, F. Azam, and M. A. Ahmed[19]	Reducir los tiempos de modelado en IFML utilizando el lenguaje natural.	Realizar una herramienta que sea capaz de automatizar el modelado en IFML.	Una herramienta que lleva por nombre “T2IF” que genera modelos IFML a partir de textos sin formato.	C#, SQL Server y Visual Studio	Finalizado

De acuerdo al análisis presentado se concluye que, para IFML, existen pocas herramientas de modelado y que la más completa (WebRatio) tiene un costo para acceder a todas las funciones disponibles, la principal función y más importante en la versión de paga es que entrega código. Además, estudios empíricos realizados en los trabajos analizados muestran que los usuarios tienen dificultad en la parte de recordar información contextual en las herramientas de modelado existentes y que la forma en que se enseña el modelado en las escuelas a veces no es la indicada.

Las principales diferencias entre la iniciativa de este proyecto y las herramientas existentes son: el complemento de eclipse se apega completamente al estándar de IFML pero sólo genera un archivo PNG, IFMLEdit.org [12] también se apega al estándar pero únicamente modela el diagrama navegacional pero el de dominio no y, por su parte, WebRatio no respeta completamente el estándar y se necesita una versión de paga para obtener el código fuente generado. Esta propuesta tiene como objetivos implementar el estándar IFML, almacenar los diagramas resultantes como archivos XMI e incorporarla a la herramienta ITO's_iBRAIN para generar código.

Para finalizar, el desarrollo de este proyecto tiene potencial ya que, con base en lo que se analizado, IFML es un lenguaje de modelado con grandes capacidades para describir la complejidad de las interfaces gráficas de usuario, independientemente de la tecnologías o plataforma, ayudando a los desarrolladores a entender mejor la interacción del sistema; además, un modelo potencializa su uso en la medida en que es fácilmente transformable a código; por tal motivo se llegó a la conclusión que hacen falta herramientas para modelado IFML y su transformación en código.

2.3 Propuesta de solución

Para dar solución al problema planteado en el capítulo uno se propone un módulo de modelado IFML formado por tres sub-módulos y siete funciones para hacer totalmente independiente la herramienta ITO's_iBRAIN. En la *figura 1* se presenta un esquema que representa los sub-módulos y cómo se relacionan entre sí.

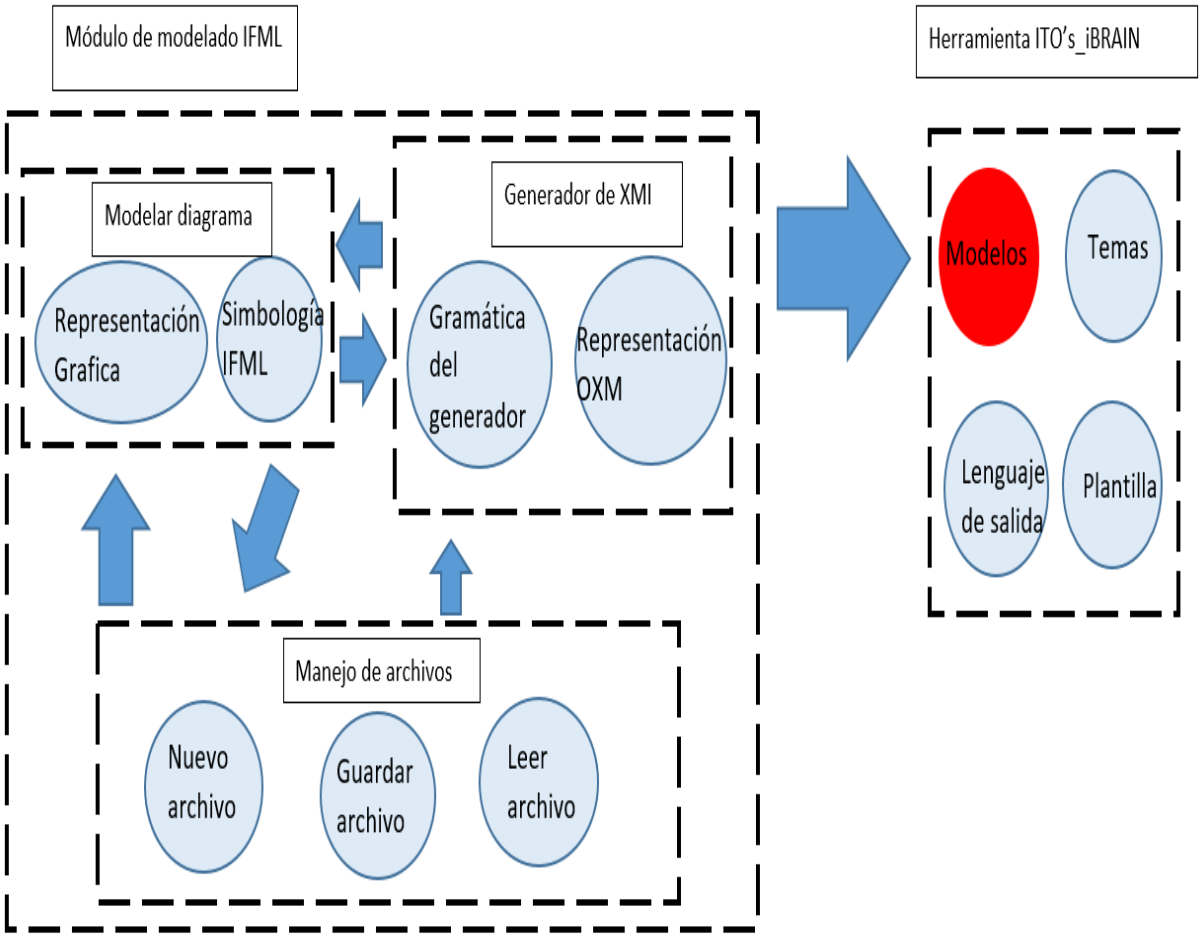

Figura 1 Esquema de la propuesta de solución

2.3.1 Sub-módulo Modelar diagrama

Este sub-módulo implica dos funciones: a) soportar el modelado del diagrama de dominio y b) soportar el modelado del diagrama navegacional con la simbología correspondiente de IFML.

2.3.2 Sub-módulo generador de XMI

Este sub-módulo tiene la funcionalidad de transformar los modelos en documentos XML y viceversa, mediante el estándar XMI definido por el OMG.

2.3.3 Sub-módulo manejo de archivos

La funcionalidad de esta parte corresponde a la gestión de archivos XML que almacenarán los diagramas modelados en la herramienta: crear nuevo archivo, guardar un archivo (almacenar un archivo XML que contiene la representación del modelo en pantalla) y leer un archivo (leer un archivo XML y representar su modelo equivalente en pantalla).

El círculo marcado en rojo de la ***Figura 1*** muestra el elemento de la herramienta ITO's_iBRAIN que será afectado para comunicarse con el módulo de modelado IFML, esto debido a que se basaba en un archivo XML con la estructura definida por WebRatio y ahora dicho archivo reflejará el estándar XMI.

2.3.4 Tecnologías a utilizar

Ya que el proyecto es el módulo de una herramienta existente, se va utilizar la tecnología JavaFX y, por lo tanto, el lenguaje de programación Java.

2.3.4.1 JavaFX

JavaFX es una colección de paquetes de gráficos y multimedia que permiten a los desarrolladores diseñar, construir, probar, depurar e implementar aplicaciones de cliente enriquecidas que funcionan de manera uniforme en todas las plataformas.

Las API de JavaFX están disponibles como una función totalmente integrada de *Java SE Runtime Environment* (JRE) y *Java Development Kit* (JDK) hasta la versión 8. Posteriormente Oracle donó el proyecto para volverlo abierto y ahora se encuentra por separado.

JavaFX Permite a los desarrolladores integrar contenido web de gráficos vectoriales, animación, audio y video en una aplicación enriquecida, atractiva e interactiva.

Extiende la tecnología al permitir que cualquier biblioteca Java se use en una aplicación JavaFX. Permite mantener un flujo de trabajo eficiente entre diseñadores y desarrolladores, ya que los diseñadores pueden trabajar en las herramientas que deseen mientras colaboran con los desarrolladores [20].

2.3.5 Definición de la gramática del generador

De igual manera que en el lenguaje de programación, ITO's_iBRAIN utiliza ANTLR 4 para la traducción de un archivo XML fuente en un árbol de representación intermedia que utiliza el módulo de generación de código, mismo que no va a variar por la implementación del módulo de modelado.

2.3.5.1 ANTLR 4

ANTLR (*ANother Tool for Language Recognition*, Otra Herramienta para el Reconocimiento de Lenguajes) es un potente generador de aplicaciones de lenguajes para leer, procesar, ejecutar o traducir texto estructurado o archivos binarios. Se usa ampliamente para crear lenguajes, herramientas y marcos de trabajo. A partir de una gramática BNF, ANTLR genera clases para realizar el análisis léxico, el análisis gramatical, la construcción de árboles de sintaxis y su recorrido. Su autor es Terence Parr.

ANTLR está en la categoría de meta-programas, por ser un programa que escribe otros programas. Partiendo de la gramática de un lenguaje, ANTLR genera un programa que determina si una sentencia o palabra pertenece a dicho lenguaje. Es gratuito y trabaja bajo licencia BSD, funciona en plataformas Linux, Windows y Mac OS[21].

2.3.6 Mapeo de objetos a XML

Se describe a continuación la tecnología seleccionada para el mapeo de objetos a XML.

2.3.6.1 JAXB

Es una tecnología de Java que provee un API para ligar un esquema XML a una representación en código y viceversa.

Ventajas:

- Es rápida: JAXB es más rápido que la API SAX debido a que el análisis se basa en clases generadas, que son pre-compiladas. En esta parte se suprime la fase de interpretación que realiza API SAX. También almacena la información como API DOM, pero no incluye la gran cantidad de funciones adicionales, simplificando la complejidad y aumentando la velocidad. Con esto se consigue un uso de memoria más eficiente.
- Restringir datos: tanto en el código que se genera como en la estructura [22].

2.3.7 Representación gráfica de modelos

A continuación, se describe la tecnología para la representación gráfica de los modelos.

2.3.7.1 *Group*

En JavaFX, un grupo (*group*) es un componente contenedor que no aplica ningún diseño especial para los elementos secundarios. *Group* supone los límites colectivos de sus nodos y no permite cambiar sus dimensiones de forma directa. Aquí, cada componente o nodo secundario se mantendrá en la posición 0,0. Si se desea dimensionar los nodos de un grupo es necesario acceder a cada nodo creado para cambiar el tamaño[23].

2.3.8 Metodología

En esta sección se presenta la metodología para el desarrollo de este proyecto.

2.3.8.1 Incremental

El modelo incremental tiene como objetivo un crecimiento progresivo de la funcionalidad. El producto va evolucionando con cada una de las entregas previstas hasta que se forma a lo requerido por el cliente.

Este enfoque se usó inicialmente para proyectos de software, aunque más tarde se aplicó a otros sectores, establece entregas parciales mediante un calendario de plazos. En cada una de ellas, el producto muestra una evolución con respecto a la fecha anterior; por lo que nunca es igual.

Una de las claves en el éxito es la evaluación de las etapas. Los responsables analizan si los resultados parciales son los esperados y si apuntan al objetivo principal. De lo contrario, implementarán las soluciones que la situación requiera.

Ventajas

- Mediante este modelo se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software.
- Es un modelo más flexible, por lo que se reduce el costo en el cambio de alcance y requisitos.
- Es más fácil probar y depurar en una iteración más pequeña.
- Es más fácil gestionar riesgos.

Desventajas

- Cada fase de una iteración es rígida y no se superponen con otras.
- Hay probabilidad de que se presenten problemas referidos a la arquitectura del sistema porque no todos los requisitos se han reunido [22].

2.3.8 Justificación de la solución propuesta

La tecnología JavaFX es fija debido a que el proyecto es complemento de una herramienta existente.

Se eligió JAXB como tecnología a utilizar porque es parte del JDK de Java y no necesita de configuraciones extras para ser utilizado.

En la metodología Incremental se seleccionó ya que el trabajar con iteraciones y desarrollo de prototipos asegura un mejor crecimiento de proyecto

Capítulo 3 Aplicación de la metodología

En este capítulo se muestra el desarrollo del Módulo de Modelado siguiendo las fases que componen la Metodología Incremental explicada en el Capítulo 2, los diagramas de clases, paquetes y casos de uso se realizaron en la herramienta Visual Paradigm [25].

3.1 Requerimientos generales

A continuación, se enlistan los requisitos funcionales y no funcionales del Módulo de Modelado, posteriormente se realizó el diagrama de casos de usos que se muestra en la *Figura 2*.

Requisitos funcionales

- Crear nuevo modelo
- Guardar modelo
- Leer modelo
- Editar modelo
- Generar código a partir de los modelos

Requisitos no funcionales

- El módulo mostrará mensajes de error claros y concretos al usuario final
- El módulo será intuitivo tomando en cuenta la forma de uso de otras herramientas de modelado, principalmente para UML
- Mantener la representación del modelo en memoria para convertirlo a representación intermedia
- El modelo navegacional se relaciona con los elementos del modelo de dominio
- Menú contextual
- Acercar y alejar contenido de diagrama
- Redimensionar la figura
- Intercambiar vista de diagramas
- Mantener sincronía en los diagramas (prioridad modelo de dominio)

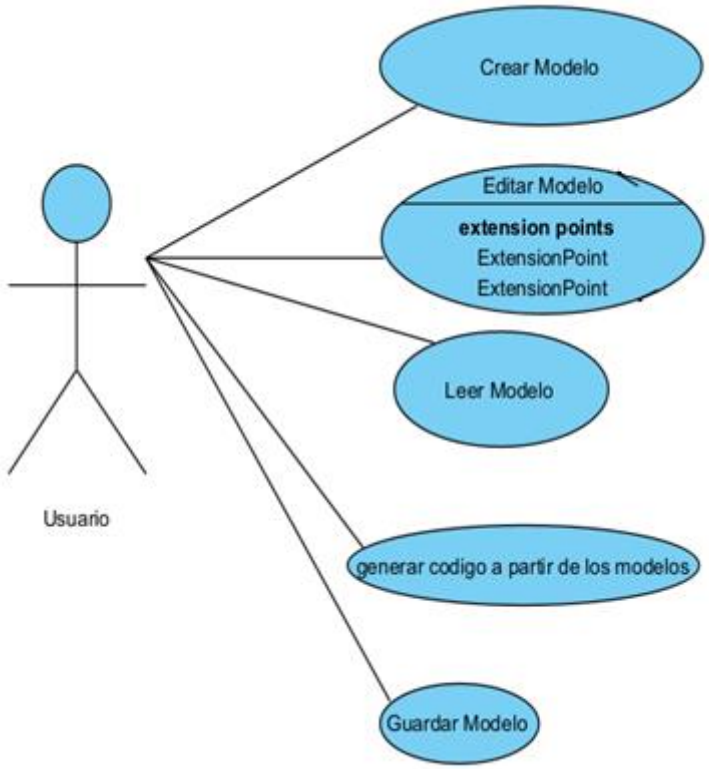

Figura 2 Diagrama casos de uso

A continuación, se describen los casos de uso por medio de narrativas (tablas 2 a 5).

Tabla 2 Caso de uso "Crear modelo".

Caso de uso	Crear modelo
Precondición	Ninguno
Propósito	Espacio limpio en el área de modelado
Tipo	Principal y esencial
Flujo básico	
Actor	**Sistema**
1. Crear nuevo diagrama	
	2. Verifica si no hay modelos activos. De lo contrario ver flujo alterno “Existencia de modelo”
	3. Limpia el área de modelado
Flujo alterno “Existencia de modelo”	
	1. Muestra mensaje “existencia de un modelo “
2. Guardar modelo existente	
	3. Limpia el área de modelado

Tabla 3 Caso de uso "Editar modelo"

Caso de uso	Editar modelo
Precondición	Crear un modelo
Propósito	Modificar un modelo ya guardado
Tipo	Principal y esencial
Flujo básico	
Actor	**Sistema**
	1. Muestra área de modelado
	2. Muestra elementos para utilizar en una barra
3. Arrastra y suelta elemento de la barra (formulario)	
	4. Agrega elemento al área de modelado
5. Ingresa elementos para un formulario	
	6. Verifica consistencia de elementos entre el diagrama nacional y de dominio. De lo contrario ver flujo alterno "Error de consistencia de datos"
7. Borra elemento gráfico del área de modelado	
	8. Quita elemento del área de modelado
Flujo alterno "Error de consistencia de datos"	
	1. Muestra mensaje con los elementos que no coinciden
2. Corrige agregando o quitando los elementos que no coinciden	

Tabla 4 Caso de uso "Leer modelo"

Caso de uso	Leer modelo
Precondición	Tener archivo XMI
Propósito	Mostrar modelo proporcionado por el usuario
Tipo	Principal y esencial
Flujo básico	
Actor	**Sistema**
1. Ingresa archivo XMI	
	2. Verifica si no hay modelos activos. De lo contrario ver flujo alterno "Existencia de modelo"
	3. Muestra modelo en el área de modelado
Flujo alterno "Existencia de modelo "	
	1. Muestra mensaje "existencia de un modelo "

Tabla 4 Continuación Caso de uso "Leer modelo"

2. Guardar modelo existente	
	3. Muestra modelo del archivo XMI en el área de modelado

Tabla 5 Caso de uso "Guardar modelo"

Caso de uso	Guardar modelo
Precondición	Existencia de elementos en el área de modelado.
Propósito	Posible exportación de diagramas en imagen Exportación de diagramas en archivo XMI
Tipo	Principal y esencial
Flujo básico	
Actor	**Sistema**
1. Selecciona guardar modelo	
	2. Verifica la existencia de elementos en el área de modelado. De lo contrario ver flujo alterno “error de guardado”
	3. Mostrar opción de guardado. Si elige “guardar imagen” ver flujo alterno ”guardar en imagen”
4. Selecciona opción guardar en archivo XMI	
5. Asigna nombre al modelo.	
	6. Genera archivo XMI a partir de los elementos del área de modelado con los estándares de la OMG con el nombre proporcionado por el usuario
Flujo Alterno ”Error de Guardado”	
	1. Muestra mensaje de error “sin elementos en el área de modelado”
2. Selecciona elementos para agregar en el área de modelado	
Flujo alterno “”guardar imagen”	
1. Seleccionar guardar “guardar en imagen”	
	2. Guarda los elementos del área de modelado en imagen

3.2 Incremento I Formato XMI para UML

En IFML el modelo de dominio se refleja en un diagrama de clases de UML, se utiliza XMI para serializar como archivo XML un diagrama de cualquier tipo de modelo. Uno de los requisitos del proyecto es exportar los modelos en un formato estándar como lo es XMI, por lo que se eligió como primer incremento el formato de archivo XMI para el modelo de dominio ya que son menos elementos que se requieren y existen herramientas que lo leen, lo cual comprueba la funcionalidad.

3.2.1 Análisis de requerimientos

Para determinar los elementos necesarios para el diagrama de dominio se analizó la documentación oficial de la OMG para asegurar su correcta implementación en ITO's_iBRAIN y la posibilidad de importación a otras herramientas de modelado.

Los elementos considerados para el modelo de dominio son los siguientes:

`XMI`: es la etiqueta principal para asegurar la importación a otras herramientas de modelado.

`uml:Model:` esta etiqueta anida los elementos del diagrama de dominio.

`OwnedMember:` esta etiqueta define el tipo de dato a utilizar por la etiqueta *OwnedAtributte*

`PackageElement:` aquí se define si va ser una clase, una asociación o una clase de asociación.

`Generalization:` esta etiqueta es especial para representar las herencias en UML.

`OwnedAtributte`: esta etiqueta es para representar los atributos de las clases.

`DefaultValue`: si el atributo de la clase tiene algún valor se define en esta etiqueta.

`LowerValue`: el valor mínimo de la multiplicidad se representa en esta etiqueta.

`UpperValue`: el valor máximo de la multiplicidad se define en esta etiqueta

`MemberEnd`: esta etiqueta es la que se encarga de realizar las relaciones entre las clases de uml.

`OwnedOperation`: en esta etiqueta se definen las operaciones que va a contener la clase.

`OwnedParameter`: los parámetros de las operaciones se definen en esta etiqueta.

`Xmi:Extension`: esta etiqueta se necesita para trabajar clases de asociación.

`AssociationClass`: esta etiqueta define las clases de asociación.

`Association:` esta etiqueta es solo para el uso de las clases de asociación.

`Uml:Diagram:` esta etiqueta anida las etiquetas necesarias para la representación gráfica del diagrama de dominio.

`Uml:Diagram.element`

`Uml:DiagramElement:` esta etiqueta define como se va a ser la representación gráfica de los elementos del diagrama de dominio.

3.2.2 Diseño

De acuerdo a la sección anterior, las etiquetas identificadas para armar un archivo XMI que represente el diagrama de dominio se reflejan en un diagrama de clases parcial, la *Figura 3* muestra el diagrama de clases para el incremento 1.

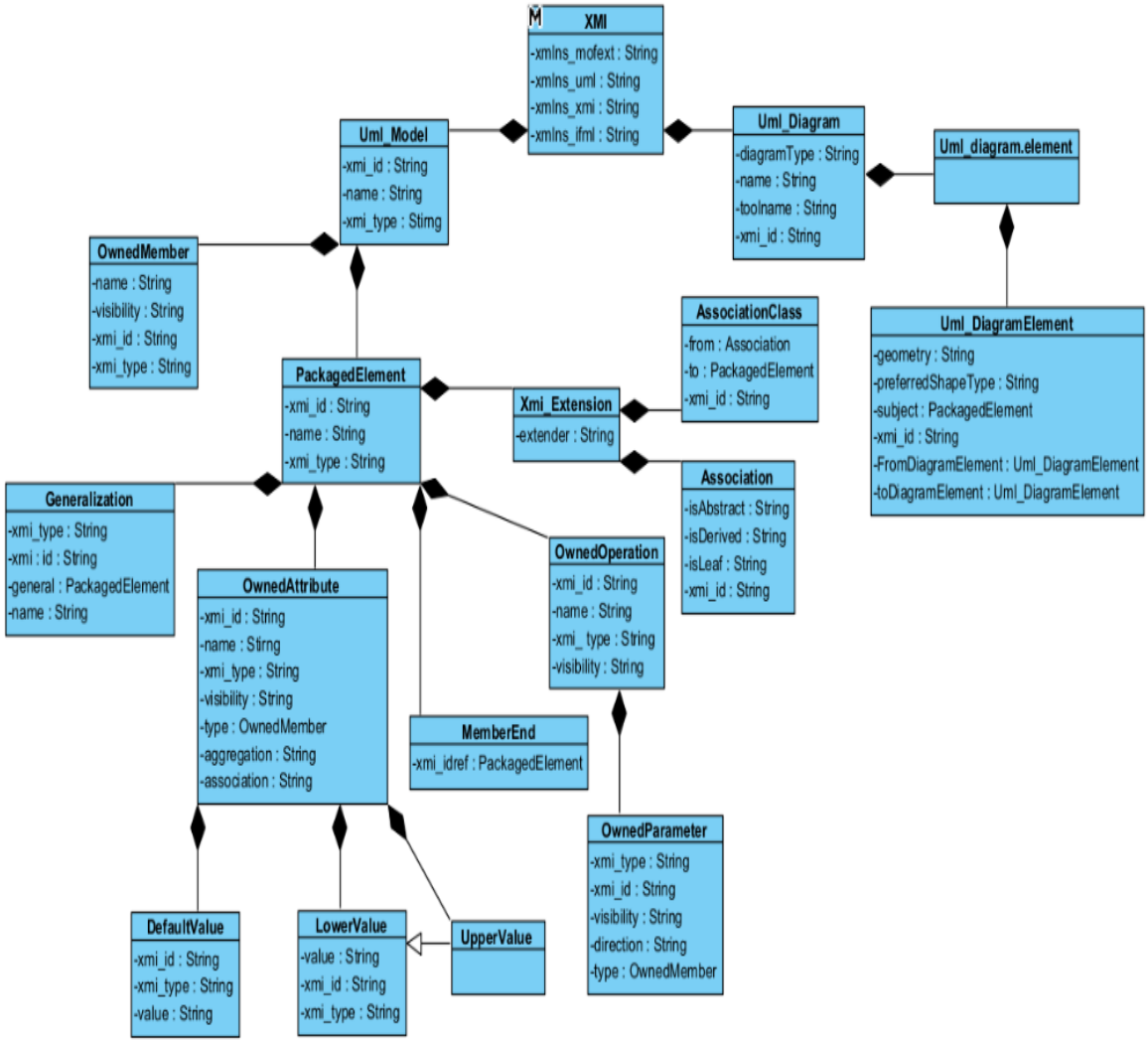

Figura 3 Diagrama de clases para el formato XMI del modelo de dominio.

3.2.3 Construcción

Como primer paso se creó un proyecto en Netbeans, separado de ITO's_iBRAIN, como se muestra en las *Figura 4*, el proyecto ya cuenta con la biblioteca de JAXB que es necesaria para realizar las pruebas correspondientes y se utilizó el JDK 17.

Para el formato XMI de UML se programaron 18 clases con sus respectivas anotaciones con las que trabaja JAXB.

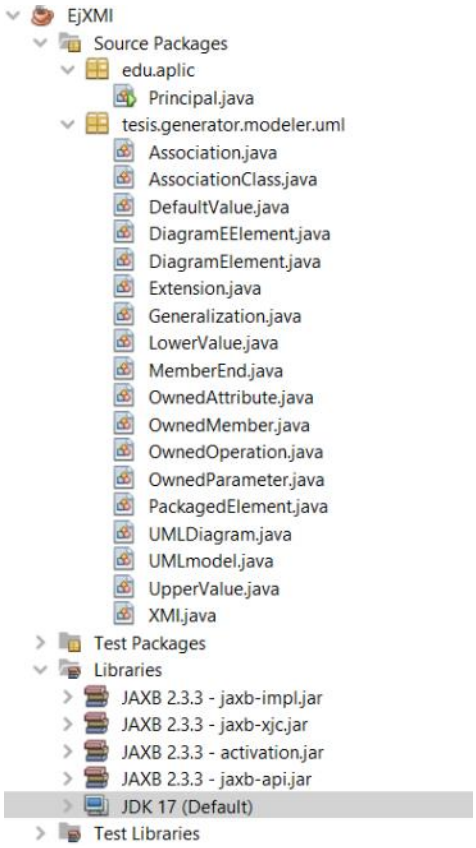

Figura 4 Estructura del proyecto para el primer incremento.

En el *Listado 1* se muestra un fragmento del código de la clase XMI; esta clase hace uso de la biblioteca de JAXB que trabaja con anotaciones, las tres primeras líneas configuran el comportamiento para la serialización y deserialización de la clase (clase a XML y viceversa). La anotacion `XmlRootElement` especifica que la clase va a serializarse como una etiqueta XML, como parámetro recibe el nombre de la etiqueta XML y el espacio de nombres, que está situado en el archivo *package-info.java* que contiene los espacios de nombres que se utilizan como prefijos de las etiquetas (`uml`, `ifml`, `xmi` y `mofext`), si `XmlRootElement` no recibe el parametro *name* entonces se serializa con el mismo nombre de la clase, el parametro *namespace* sí es obligatorio para que se construya correctamente el archivo `.xmi` con el prefijo que corresponde en la etiqueta principal; `XmlAccessorType` (línea 2) es una anotación que describe el método de acceso de los atributos de la clase al generar archivos XML, en este caso significa que los atributos serán privados y se accederá a ellos mediante métodos `get/set`; la anotación `XmlType` (línea 3) se utiliza cuando existen etiquetas anidadas para indicar el orden de serialización; la

anotación `XmlAttribute` (línea 11) permite indicar el nombre con que se va a serializar un atributo XML y el espacio de nombres para que se construya la etiqueta con el prefijo que corresponde, en este caso va a tener el prefijo `uml`, como en este caso se indicó en la línea 2 que los atributos de la clase se acceden mediante métodos, la anotación se coloca arriba del método *get* para que tome el nombre del método como nombre del atributo de la etiqueta XML; en la línea 20, la anotación `XmlElement` indica que el atributo de la clase se serializará como etiqueta anidada a la etiqueta principal (representada por la clase), el parámetro permite generar un nombre distinto de etiqueta y de igual manera se asigna el *namespace* para la construcción del prefijo, ya que por omisión se toma el nombre del método.

Listado 1 Fragmento de código de la clase XMI

```
1  @XmlRootElement(name = "XMI",
   namespace="http://www.omg.org/spec/XMI/20131001")
2  @XmlAccessorType(XmlAccessType.PROPERTY)
3  @XmlType(propOrder = {"model", "diagram"})
4  public class XMI{
5
6  private List<Model> Model;
7  private List<Diagram> Diagram;
10 …
11 @XmlElement(name = "Model" ,
   namespace="http://www.omg.org/spec/UML/20131001")
12     public List<Model> getModel() {
13         return Model;
14     }
15
16     public void setModel(List<Model> model) {
17         this.Model = model;
18 }
19
20 @XmlElement(name = "Diagram" ,
   namespace="http://www.omg.org/spec/UML/20131001")
21     public List<Diagram> getDiagram() {
22         return Diagram;
23     }
24
25     public void setDiagram(List<Diagram> Diagram) {
26         this.Diagram = Diagram;
27     }
```

Las 18 clases que se programaron hacen uso de las mismas anotaciones que se observan en el *Listado 1*.

En el *Listado 2* se muestra el uso de algunas clases por medio de una clase para pruebas (`Principal`), en las líneas 3 – 9 se definen las variables que se van a utilizar, en las líneas 10 -17 se definen las listas que van a utilizar, las instancias en las listas 18 y 19 se refieren a clases propias de JAXB para dar formato de salida XML, de la línea 20 a la línea 77 se definen los atributos de las etiquetas que se van a utilizar, en las líneas 78 – 96 se agregan los atributos definidos a las listas de igual manera ya definidas y en las líneas 97 – 106 se encuentra un bloque *try-catch* que serializa la clase XMI auxiliándose de JAXB.

Listado 2 Clase para prueba "Principal"

```
1   public class Principal {
2   public static void main(String[] args) {
3           PackagedElement clas1, clas2;
    …
9           XMI xm;
10          ArrayList<UMLmodel> Lmodelos = new ArrayList<>();
    …
17          ArrayList<OwnedOperation> Lope2 = new ArrayList<>();
18          JAXBContext jaxbContext = null;
19          Marshaller jaxbMarshaller = null;
20          xm = new XMI();
    …
24          mod = new UMLmodel();
25          mod.setId("_1");
26          mod.setType("uml:Model");
27          mod.setName("model");
    …
77          gen1.setTypee("uml:Generalization");
78          Lmodelos.add(mod);
    …
96          clas2.setGeneralization(Lge);
97          try {
98              jaxbContext = JAXBContext.newInstance(XMI.class);
99              jaxbMarshaller = jaxbContext.createMarshaller();
100 jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
    true);
101             jaxbMarshaller.marshal(xm, new
    File("./ejemplo10.xmi"));
102         } catch (JAXBException e) {
```

```
103             Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
                null, e);
104                 }
105             }
106         }
```

3.2.4 Despliegue

En esta sección se muestran las pruebas realizadas para el primer incremento. Al ejecutar la aplicación de prueba el resultado es la estructura de un archivo XMI que contiene un diagrama de clases UML, cada clase tiene una operación y un atributo su relación está formada por una herencia, en la *Figura 5* se observa la estructura del archivo XMI.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmi:XMI xmlns:mofext="http://www.omg.org/spec/MOF/20131001"
         xmlns:uml="http://www.omg.org/spec/UML/20131001"
         xmlns:xmi="http://www.omg.org/spec/XMI/20131001">
    <uml:Model xmi:id="_1" name="model" xmi:type="uml:Model">
        <packagedElement xmi:id="_2" name="Casa1" xmi:type="uml:Class">
            <ownedAttribute xmi:id="_3" name="color" type="string_id" xmi:type="uml:Property" visibility="private"/>
            <ownedOperation xmi:id="_4" name="operacion1" xmi:type="uml:Operation" visibility="public"/>
        </packagedElement>
        <packagedElement xmi:id="_5" name="Casa2" xmi:type="uml:Class">
            <ownedAttribute xmi:id="_6" name="color2" type="string_id" xmi:type="uml:Property" visibility="private"/>
            <ownedOperation xmi:id="_7" name="operacion1" xmi:type="uml:Operation" visibility="public"/>
            <generalization general="_2" xmi:id="_11" name="H-C1-C2" xmi:type="uml:Generalization"/>
        </packagedElement>
        <ownedMember xmi:id="string_id" name="string" xmi:type="uml:DataType" visibility="public"/>
    </uml:Model>
</xmi:XMI>
```

Figura 5 Ejemplo de archivo XMI para el modelo de dominio

Una de las características de XMI es que permite que distintas herramientas lean el mismo diagrama, aunque no se haya desarrollado ahí; por lo tanto, al importar el archivo XMI en la herramienta *MagicDraw* [26] (*Figura 6*), se observa que hace la correcta importación de los elementos antes mencionados, lo que permite su reutilización. Cabe mencionar que la mayoría de las herramientas no incluyen los elementos importados directamente dentro de un diagrama (los datos gráficos específicos no son parte de XMI) pero sí permiten

incluirlos en un diagrama, una vez importados, sin perder la información propia del modelo (nombre, visibilidad, alcance, restricciones entre otras cosas).

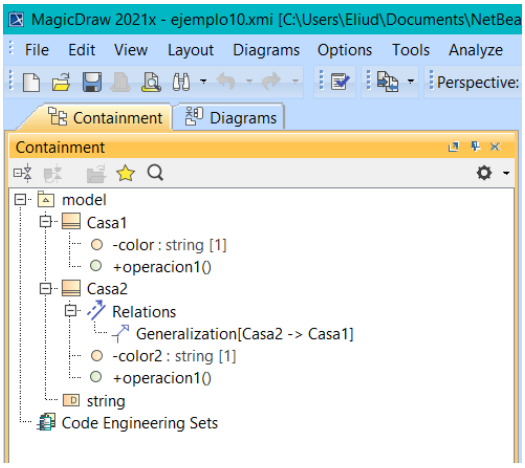

Figura 6 Importación de los elementos del diagrama de dominio en MagicDraw

3.3 Incremento II Formato XMI del diagrama navegacional de IFML

En IFML el diagrama navegacional es el más importante y se optó como segundo incremento ya que son más elementos y dado que no existen herramientas registradas que sean capaces de leer archivos XMI para el diagrama navegacional las pruebas son distintas.

3.3.1 Análisis de requerimientos

De igual manera que en el incremento anterior, para determinar los elementos necesarios para el diagrama navegacional se realizó un extenso análisis de las definiciones formales de la OMG para IFML que son el Metamodelo y DI (*Diagram Interchange*), también se revisó extensivamente el libro de IFML [4] para asegurar su correcta implementación en ITO's_iBRAIN.

Para realizar los archivos XMI se tomó en cuenta el análisis antes mencionado y, debido a la inexistencia de herramientas capaces de leer archivos XMI de IFML, se revisaron dos herramientas en línea (la primera está aprobada por la OMG) que se relacionan con este

tema; dichas herramientas no validan, pero dan una mejor idea de las condiciones que debe cubrir el archivo XMI que represente el modelo navegacional de IFML.

Los elementos considerados para el diagrama navegacional son los siguientes:

`XMI`: es la misma etiqueta que en el incremento I, al que se agregó un atributo que hace referencia a la documentación de IFML.

`Ifml_Model`: esta etiqueta anida todos los elementos del diagrama navegacional de IFML.

`Context`: determina cómo configurar la interfaz del usuario resultante, basada en roles; trabaja en conjunto con la etiqueta `ViewPoint`.

`UserRole`: esta etiqueta trabaja en conjunto con `Context` y representa un rol desempeñado por un usuario humano o sistema.

`ViewPoint`: esta etiqueta facilita la comprensión del sistema por medio de roles de usuario.

`InteractionFlowModel`: esta etiqueta anida la interacción de los elementos con el usuario.

`ViewContainer`: este elemento representa las "pantallas" en el diagrama navegacional.

`ViewElement`: solo es una clase de la que heredan las clases `ViewContainer`, `Form`, `Detail`, `List` y no se refleja en el archivo XMI.

`Action`: este elemento representa la ejecución de una funcionalidad del sistema; dentro de la herramienta ITO's_iBRAIN, el elemento se implementa como una llamada a la misma página sin navegación.

`Form`: este elemento representa formularios de entrada donde el usuario envía información, este elemento trabaja en conjunto con `ViewContainer`.

`Detail`: de igual manera que `Form`, trabaja en conjunto con los elementos `ViewContainer` y `DataBinding`, es el encargado de mostrar los detalles de una instancia.

`List`: este elemento se utiliza para mostrar una lista de instancias de `DataBinding`.

`GenericEvent`: de igual manera que `ViewElement` solo es una clase de la que heredan y no se refleja de manera visual en el archivo XMI.

`Event`: este elemento representa una situación externa que afecta el estado de la aplicación, dado que la respuesta del sistema es la ejecución de una funcionalidad, es posible que este elemento quede anidado dentro de un elemento `Action`.

`SubmitEvent`: desencadena paso de parámetros por medio de un `DataFlow`.

`SelectionEvent`: desencadena paso de parámetros por medio de `NavigationFlow`.

`SelectionField`: es un campo permite la selección de uno o varios valores definidos en `Slots`.

`Slot`: en este elemento se definen valores para utilizarlos como opciones fijas en un `SelectionField`.

`SimpleField`: este elemento muestra un valor o captura la entrada de texto por el usuario.

`NavigationFlow`: hace paso de paramentos con navegación entre `ViewElements`.

`DataFlow`: este elemento pasa los parámetros, pero no hace navegación entre `ViewElements`.

`ParameterBindigGroup`: agrupa los elementos `Parameter`.

`Parameter`: sus instancias tienen valores.

`DataBinding`: este elemento enlaza el diagrama navegacional con el diagrama de dominio.

`ValidationRule`: determina si el contenido de un campo es válido o no y de esta clase heredan varios elementos que definen las validaciones más comunes (tipo de dato, rango, si permite vacíos o no, entre otros).

`TypeVR`: se especifica que tipo de dato debe ser aceptado en el campo.

`ValueLengthVR`: si hay un límite de caracteres en el campo se especifica en esta etiqueta.

`RegularExpressionVR`: las expresiones regulares se especifican en esta etiqueta.

`MandatoryVR`: especifica si el campo es obligatorio.

`EmailVR`: especifica si el campo va a tener validación para ingresar un correo electrónico.

`CreditCardVR`: al ingresar una tarjeta de crédito se valida en esta etiqueta.

`LikeVR`: define si un valor es aceptado o en qué posición debe ir.

`CollectionVR`: se define una lista de valores aceptables para el campo.

`CaptchaVR`: medida de seguridad como autenticación pregunta respuesta.

`IfmlDiagram`: esta anida las etiquetas necesarias para la representación gráfica del diagrama de navegacional.

`IfmlDiagram.element:` esta etiqueta anida las etiquetas `IfmlDiagramElement`.

`IfmlDiagramElement:` esta etiqueta define como se va a ser la representación gráfica de los elementos del diagrama de navegacional.

3.3.2 Diseño

De acuerdo a la sección anterior, las etiquetas identificadas para armar un archivo XMI que represente el diagrama navegacional se representa en un diagrama de clases, la Figura 7 muestra dicho diagrama.

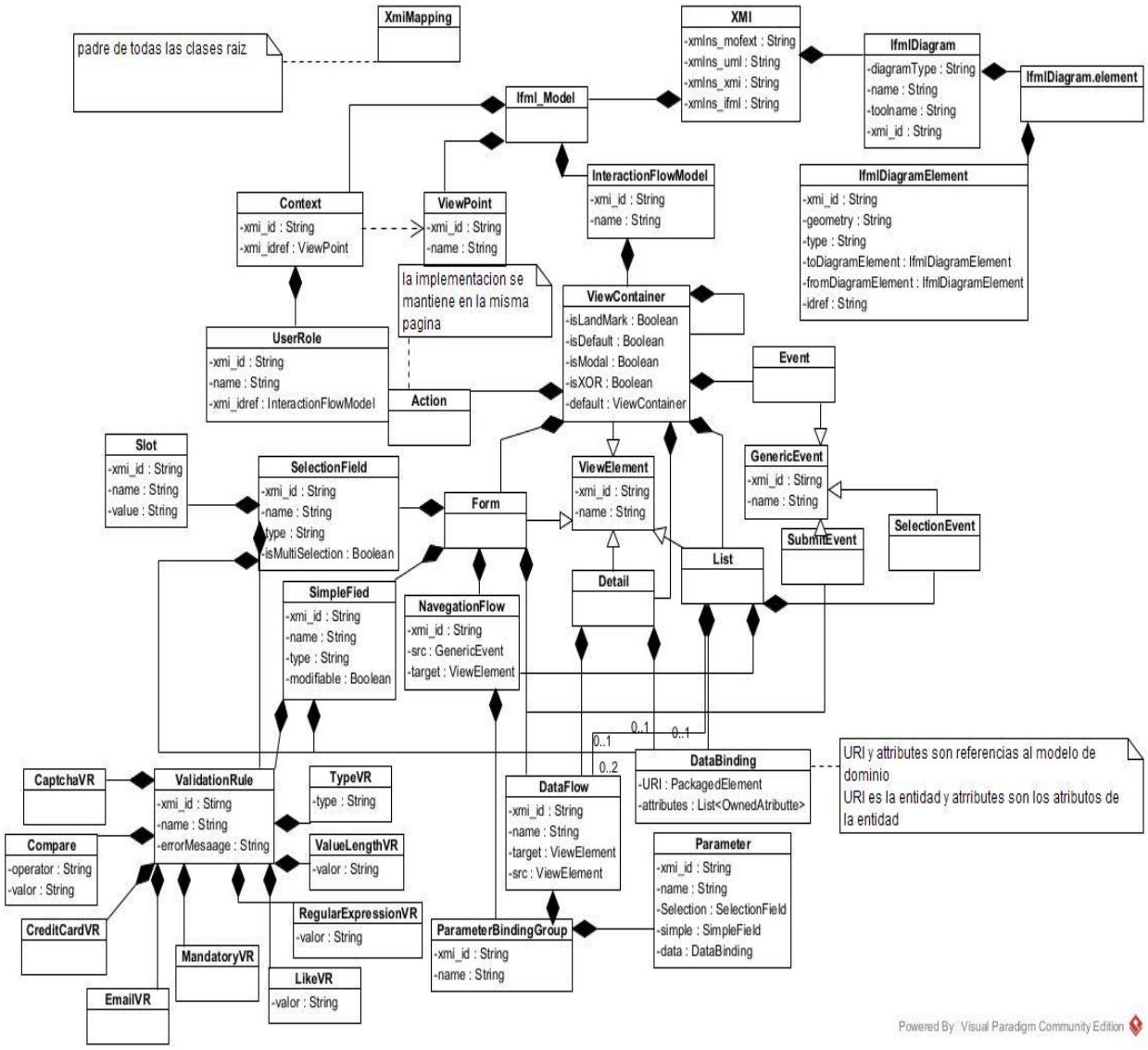

Figura 7 Diagrama de clases para el formato XMI para el diagrama navegacional

3.3.3 Construcción

A la estructura de la aplicación que se mostró en la *Figura 4* se le añadió un nuevo paquete `tesis.generator.modeler.ifml`, donde se encuentran todas las clases para representar los elementos de IFML a serializar en archivo, la estructura resultante se muestra en la *Figura 8.*

Para el formato XMI correspondiente al diagrama navegacional de IFML se programaron 35 clases con sus respectivas anotaciones de JAXB.

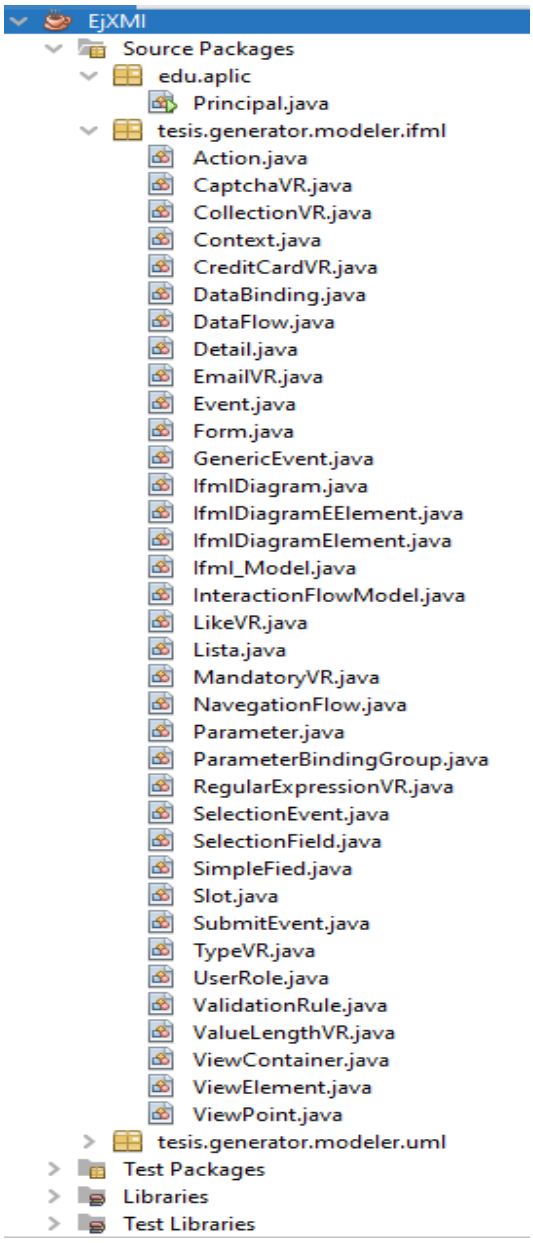

Figura 8 Estructura del proyecto para el segundo incremento.

A continuación, en el *Listado 3* se muestran fragmentos de código más relevantes, haciendo usos de la biblioteca de JAXB.

La clase XMI sufrió algunas afectaciones al código mostrado en el *Listado 1*, ya que se anexaron nuevas líneas de código, en la línea 6 y 7 se definió la variable `Ifml_model y DiagramIfml` que hace referencia a la clase `Ifml_Model` que es la raíz que anida los elementos de IFML y `DiagramIfml` que es la raíz que anida la información para la representación gráfica de los elementos de IFML, en la línea 11 y 20 se hace uso de la

anotación `XmlElement` que se describió en el incremento I y en las líneas 12 – 26 se definen los métodos *get* y *set* de las variables definidas en la líneas 6 y 7.

Listado 3 Fragmento de código de la clase XMI afectado en el incremento II.

```
1  @XmlRootElement(name = "XMI",
   namespace="http://www.omg.org/spec/XMI/20131001")
2  @XmlAccessorType(XmlAccessType.PROPERTY)
3  @XmlType(propOrder = {"diagramIfml","ifmlModel"})
4  public class XMI{
5
6  private List<DiagramIfml> DiagramIfml;
7  private List<Ifml_Model> IfmlModel;
8  …
9
10
11 @XmlElement(name = "Diagram" ,
   namespace="https://www.omg.org/spec/IFML/20140301")
12     public List<DiagramIfml> getDiagramIfml() {
13         return DiagramIfml;
14     }
15
16     public void setDiagramIfml(List<DiagramIfml>
   DiagramIfml) {
17         this.DiagramIfml = DiagramIfml;
18     }
19
20 @XmlElement(name = "Model" ,
   namespace="https://www.omg.org/spec/IFML/20140301")
21     public List<Ifml_Model> getIfmlModel() {
22         return IfmlModel;
23     }
24
25     public void setIfmlModel(List<Ifml_Model> Ifml) {
26         this.IfmlModel = Ifml;
27     }
28 …
```

Las 35 clases que se programaron hacen uso de las mismas anotaciones que se observan en el *Listado 3* y se explicaron en el incremento I.

En el Listado 4 se muestra el uso de algunas clases por medio de una clase para pruebas (`Principal`) de las líneas 3 – 18 se definen las variables que se van a utilizar, de las líneas 20 – 36 se definen las listas, las líneas 41 y 42 son clases de JAXB que se necesitan para dar formato XML al archivo de salida, de las líneas 41 – 148 se definen los atributos

de las etiquetas, de las líneas 150 – 203 se agregan los atributos definidos en las listas definidas en las líneas 20 – 36, y al final en las líneas 204 – 209 en un bloque *try-catch* contiene el uso de las clases de JAXB que es donde se le asigna un nombre al archivo.

Listado 4 Fragmento de código de la clase para prueba "Principal"

```
1   public class Principal {
2       public static void main(String[] args) {
3           XMI xm;
    …
18          Detail dt;
19
20          ArrayList<Ifml_Model> LIfml = new ArrayList<>();
    …
36          ArrayList<Detail> Ldet = new ArrayList<>();
37
38          JAXBContext jaxbContext = null;
39          Marshaller jaxbMarshaller = null;
40
41          xm = new XMI();
    …
145          nvf2 = new NavegationFlow();
146          nvf2.setXmi_id("_20");
147          nvf2.setSrcc(sle);
148          nvf2.setTargett(dt);
149
150         LIfml.add(ifmlmodel);
151         xm.setIfmlModel(LIfml);
    …
203         vc3.setoDetail(Ldet);
204         try {
205             jaxbContext = JAXBContext.newInstance(XMI.class);
206             jaxbMarshaller = jaxbContext.createMarshaller();
207 jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
    true);
208             jaxbMarshaller.marshal(xm,                    new
    File("./modeloNavegacional.xmi"));
209
210         } catch (JAXBException e) {
211 Logger.getLogger(Principal.class.getName()).log(Level.SEVERE,
    null, e);
212         }
213 }
214 }
```

3.3.4 Despliegue

En esta sección se muestran las pruebas realizadas para el segundo incremento. Al ejecutar la aplicación de prueba el resultado es la estructura de un archivo XMI que contiene el diagrama navegacional de IFML, este archivo contiene un rol de administrador, con 3 páginas, una página tiene un formulario con dos campos uno simple y otro de selección, este formulario hace el envío de los parámetros con navegación a una lista que se encuentra en otra página y cada elemento de la lista tiene eventos que se dirigen a otra página que muestra los detalles de los elementos de la lista. En la *Figura 9* se muestra la estructura del archivo XMI.

Es importante destacar que la OMG provee un validador en línea para XMI, siempre que se cuente con el Metamodelo correspondiente [27] en el caso de IFML, el validador no reconoce el archivo de Metamodelo [28] publicado.

Debido a que no existen herramientas registradas para la importación del diagrama navegacional de IFML en archivos XMI, la prueba de importación a otras herramientas de modelado no se realizó, sólo se verificó que el archivo cumpliera con las reglas de XML y XMI.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmi:XMI xmlns:mofext="http://www.omg.org/spec/MOF/20131001"
         xmlns:uml="http://www.omg.org/spec/UML/20131001"
         xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
         xmlns:ifml="https://www.omg.org/spec/IFML/20140301">
    <ifml:Model>
        <ViewPoint id="vp1" name="areaAdministrador"/>
        <Context id=""  idref="vp1">
            <UserRole id="ctx1" name="Administrador" idref="ifml"/>
        </Context>
        <InteractionFlowModel id="ifml" name="ModelAdm" >
            <ViewContainer id="vc1" name="Bienvenido" isDefault="true" isLandMark="true" isModal="false">
                <Form xmi:id="1FA" name="Formulario Alumnos">
                    <SimpleField xmi:id="1sf" name="id" type="int" modifiable="false"/>
                    <SimpleField xmi:id="2sf" name="nombre" type="String" modifiable="true"/>
                    <SelectionField xmi:id="3sf" name="sexo" type="String" IsMultiSelection="true">
                        <Slot xmi:id="1s" name="Maculino" value="M" />
                        <Slot xmi:id="2s" name="Femenino" value="F"/>
                    </SelectionField>
                    <SubmitEvent xmi:id="1SuE" name="Enviar"/>
                    <NavegationFlow xmi:id="1NF" src="1SuE" target="1LA"/>
                </Form>
            </ViewContainer>

            <ViewContainer id="vc2" name="Segunda" isDefault="false" isLandMark="true" isModal="false">
                <List xmi:id="1LA" name="Lista Alumnos">
                    <SelectionEvent id="1SeE" name="select" />
                    <NavegationFlow xmi:id="2NF" source="1SeE" target="1DA"/>
                </List>
            </ViewContainer>

            <ViewContainer id="vc3" name="Tercera" isDefault="false" isLandMark="false" isModal="false">
                <Details xmi:id="1DA" name="Alumnos">
                </Details>
            </ViewContainer>
        </InteractionFlowModel>
    </ifml:Model>
</xmi:XMI>
```

Figura 9 Ejemplo de archivo XMI para el diagrama navegacional de IFML.

3.4 Incremento III Anexar Incremento I y II a ITO's_iBRAIN

Al haber terminado y realizado las pruebas pertinentes para los dos incrementos, se procedió a implementarlos en ITO's_iBRAIN por medio de una pantalla con un botón y haciendo uso de elementos estáticos.

3.4.1 Análisis de requerimientos

Mediante la creación de la pantalla del Modelador y el botón "Guardar" en el ambiente de ITO's_iBRAIN, es necesario implementar la función de exportar un diagrama navegacional con elementos estáticos de IFML. Dividiendo el ambiente en 3 secciones, en la primera sección para mostrar los elementos de IFML en la segunda sección mostrar los elementos UML y en la tercera sección el área de modelado para la representación gráfica.

3.4.2 Diseño

Para este incremento se hizo un maquetado en la herramienta *Balsamiq Mockups* para dar una idea clara de lo que se describió en la sección anterior. En la *Figura 10* se muestra el maquetado resultante para el módulo de modelado de ITO's_iBRAIN.

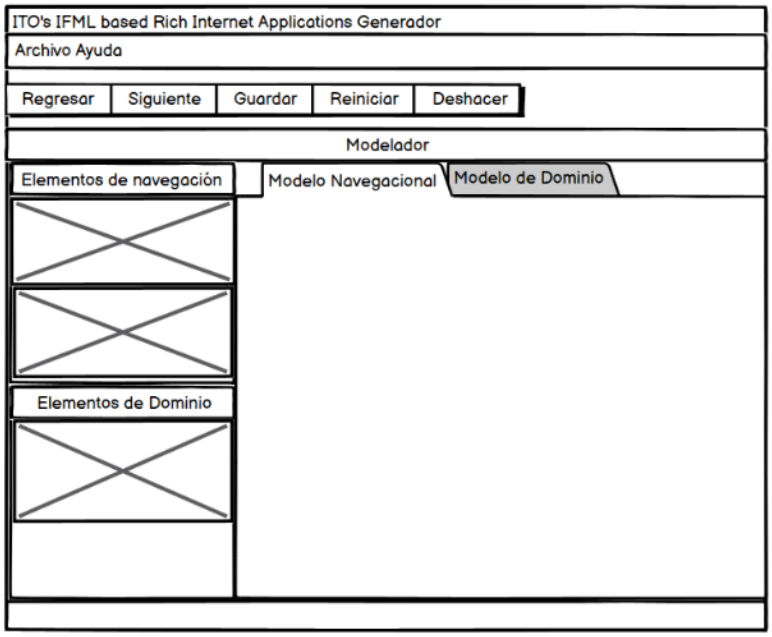

Figura 10 Diseño abstracto del ambiente de ITO's_iBRAIN para el ambiente de modelado

3.4.3 Construcción

En la *Figura 11* se muestra la estructura de ITO's_iBRAIN, los elementos remarcados son los que se anexaron para este incremento, los cuales son los paquetes *tesis.generator.modeler.ifml* que es el encargado del ambiente visual, *tesis.generator.modeler.uml y tesis.generator.enviroment.modeler* son los que se mostraron en los incrementos I y II.

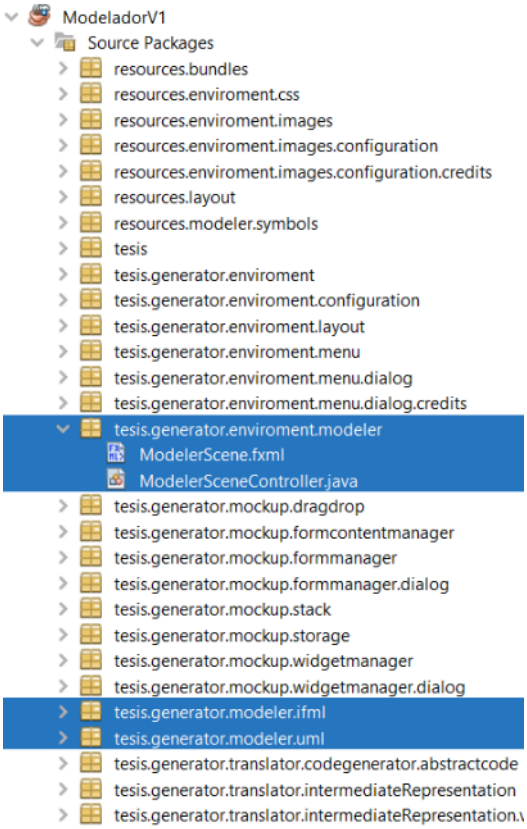

Figura 11 Nueva estructura de la herramienta ITO's_iBRAIN.

JavaFX trabaja con archivos FXML para mostrar el ambiente visual de sus aplicaciones, en el *Listado 5* se muestra el contenido del archivo *ModelerScene*. En la línea 4 esta una etiqueta `BorderPane` la cual se utilizó para seccionar el ambiente visual, en la línea 6 está la etiqueta *`top`* es la sección de los botones, en la línea 11 está la etiqueta *`left`* está es la sección de los elementos para el modelo navegacional y de dominio, en la línea 21 está la etiqueta *`center`* donde se podrán las etiquetas necesarias para la representación gráfica de los elementos y en la línea 22 es un `TabPane` para seleccionar el modelo de dominio o navegacional, para efectos de este incremento en la línea 24 se utilizó la etiqueta `ListView` para mostrar una imagen que representa un modelo navegacional estático, posteriormente esta etiqueta se eliminará.

Listado 5 Contenido de archivo ModelerScene.fxml

```
1  <AnchorPane xmlns:fx="http://javafx.com/fxml/1" id="AnchorPane"
2          prefHeight="400.0" prefWidth="600.0"
3  fx:controller="tesis.generator.enviroment.modeler.ModelerSceneContr
   oller">
4      <BorderPane fx:id="oRoot" prefWidth="1000"
   xmlns:fx="http://javafx.com/fxml/1" >
5
6       <top>
7          <HBox  alignment="BOTTOM_CENTER" styleClass="upperTitle" >
8              <Label text="%mod.tit" styleClass="upperTitleText"/>
9          </HBox>
10      </top>
11         <left>
12    <VBox fx:id="oToolBar" prefHeight="450" styleClass="left">
13         <Label text="%mod.navElements" styleClass="tit"/>
14           <ListView fx:id="oNavElemList" styleClass="listcell"
   minHeight="225"/>
15
16          <Label text="%mod.domElements" styleClass="tit"/>
17            <ListView fx:id="oDomElemList" styleClass="listcell"
   minHeight="225"/>
18
19     </VBox>
20         </left>
21         <center >
22           <TabPane tabClosingPolicy="UNAVAILABLE" fx:id="oTabPane">
23                 <tabs>
24                     <Tab fx:id="navModelTab" text="%mod.navModel">
25                         <ListView fx:id="dgNav"/>
26                     </Tab>
27                     <Tab fx:id="domModelTab" text="%mod.domModel">
28                         <ListView fx:id="dgDom"/>
29                     </Tab>
30                 </tabs>
31           </TabPane>
32         </center>
33         <bottom>
34           <GridPane styleClass="disclaimer" alignment="center"
   hgap="10" vgap="10">
35
36             <Label GridPane.columnIndex="1" GridPane.rowIndex="1"
37                GridPane.rowSpan="3" GridPane.valignment="TOP"
38                        text="%txt.label" />
39           </GridPane>
40         </bottom>
```

```
41      </BorderPane>
42  </AnchorPane>
```

En el listado 6 se muestra el código de la clase *ModelerSceneController* que controla el comportamiento de la pantalla *ModelerScene.fxml*. En las líneas 4 – 5 se hace referencia a los identificadores (atributo `id`) de las etiquetas de la pantalla por medio de variables; en el método `initialize`, que empieza en la línea 10, se define en dónde se encuentran las imágenes que se utilizaron en este incremento (5 imágenes); de las líneas 33 – 47 se define la activación y desactivación de los elementos de la lista según la pestaña (*tab*) seleccionada.

Listado 6 Fragmento de código del archivo ModelerSceneController.java

```
1   public class ModelerSceneController implements Initializable,
2           GeneratorSceneController {
3
4       @FXML
5       TabPane oTabPane;
6       @FXML
7       ListView oNavElemList, oDomElemList, dgNav;
8
9       @Override
10      public void initialize(URL url, ResourceBundle rb) {
11          Image oImg;
12          ImageView oImgView;
13
14          oImg = new
    Image("resources/modeler/symbols/viewcontainer.png");
15          oImgView = new ImageView(oImg);
16          oNavElemList.getItems().add(oImgView);
17          oImg = new
    Image("resources/modeler/symbols/xorviewcontainer.png");
18          oImgView = new ImageView(oImg);
19          oNavElemList.getItems().add(oImgView);
20          oImg = new
    Image("resources/modeler/symbols/landviewcontainer.png");
21          oImgView = new ImageView(oImg);
22          oNavElemList.getItems().add(oImgView);
23
24          oImg = new Image("resources/modeler/symbols/si.png");
25          oImgView = new ImageView(oImg);
26          dgNav.getItems().add(oImgView);
27
```

```
28          oImg = new Image("resources/modeler/symbols/class.png");
29          oImgView = new ImageView(oImg);
30          oDomElemList.getItems().add(oImgView);
31          oDomElemList.setDisable(true);
32
33  oTabPane.getSelectionModel().selectedItemProperty().addListener(
34                  new ChangeListener<Tab>() {
35              @Override
36              public void changed(ObservableValue<? extends Tab> ov,
37                      Tab prev, Tab curr) {
38                  if (curr.getId().equals("navModelTab")) {
39                      oNavElemList.setDisable(false);
40                      oDomElemList.setDisable(true);
41                  } else {
42                      oNavElemList.setDisable(true);
43                      oDomElemList.setDisable(false);
44                  }
45              }
46          });
47      }
```

En el *Listado 7* se muestra un fragmento de código del método *save* que es el encargado de que funcione el botón guardar de ITO's_iBRAIN que permite exportar el modelo estático en un archivo XMI. De las líneas 2-210 es el mismo código del *Listado 4* ya que se utilizaron los mismos elementos de IFML, pero de la línea 211 – 216 se anexa una alerta para que el usuario vea que el archivo se creó correctamente.

Listado 7 Fragmento de código del método save del archivo ModelerSceneController.

```
1   public void save(ActionEvent ae) {
    …
205 try {
206       jaxbContext = JAXBContext.newInstance(XMI.class);
207          jaxbMarshaller = jaxbContext.createMarshaller();
208 jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
209 jaxbMarshaller.marshal(xm, new File("./modeloNavegacional.xmi"));
210
211             Alert msj = new Alert(Alert.AlertType.INFORMATION);
212             msj.setTitle("Resultado");
213             msj.setHeaderText(null);
215             msj.setContentText("Archivo XMI creado");
216             msj.showAndWait();
217         } catch (JAXBException e) {
218
```

```
                Logger.getLogger(ModelerSceneController.class.getName()).log(Level.SEVERE,
null, e);
219             }
220
221         }
```

3.4.4 Despliegue

En esta sección se muestran las pruebas realizadas para el tercer incremento. En la *Figura 12* se muestra el ambiente de ITO's_iBRAIN, seccionado como se especificó en el diseño. Como se ha mencionado, en este punto la imagen del diagrama es fija, todavía no lo genera el modelador.

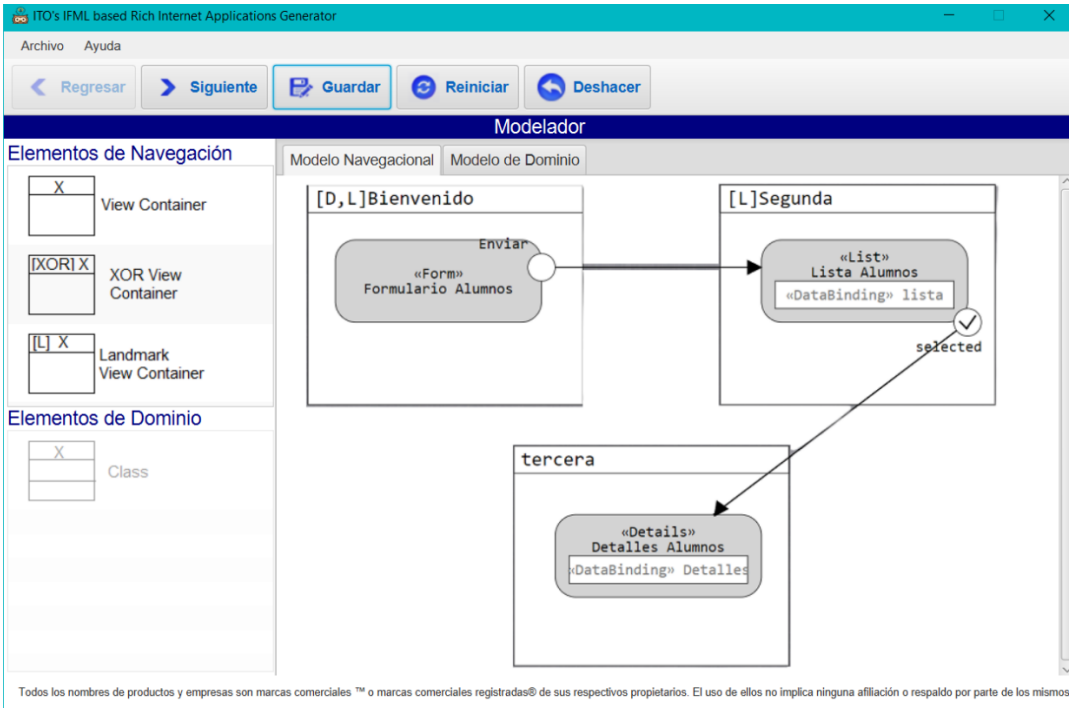

Figura 12 Prototipo del ambiente de modelado en ITO's_iBRAIN.

El resultado de oprimir el botón Guardar se muestra en la *Figura 13*. Es una alerta que aparece en pantalla con un mensaje "Archivo XMI creado".

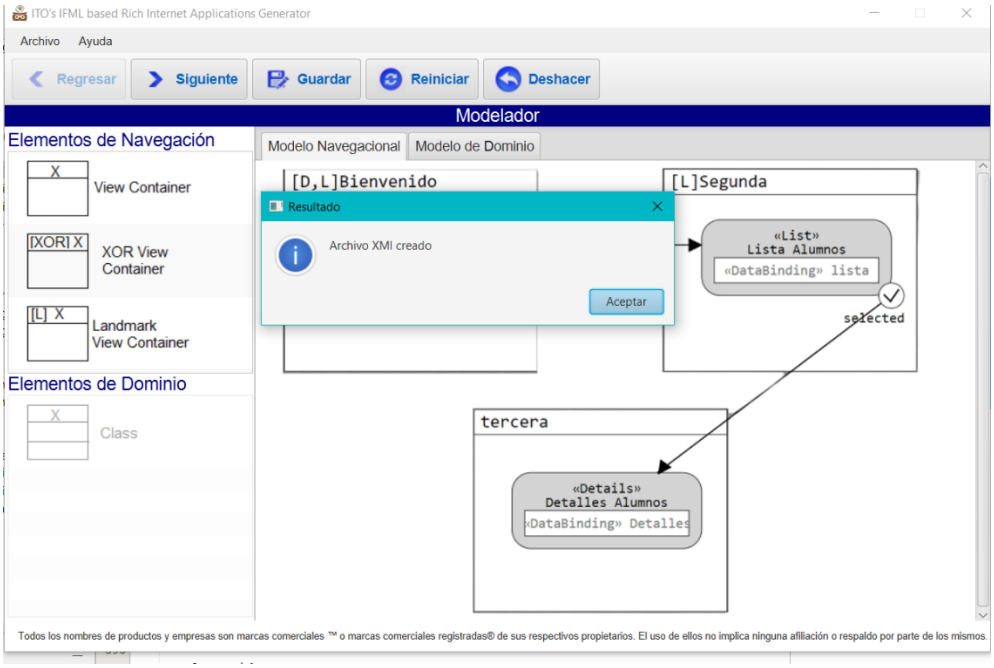

Figura 13 Alerta de archivo XMI creado

El archivo XMI creado es el mismo que el de la *Figura 9* que se encuentra en el despliegue del segundo incremento.

3.5 Incremento IV Área de modelado para el diagrama navegacional en ITO´s_iBRAIN

Posterior del incremento III que se realizó con un modelo estático, en este incremento se agregan a una lista los elementos de IFML para modelar de manera dinámica, atendiendo las restricciones de cada elemento.

3.5.1 Análisis de requerimientos

En la primera sección que se diseñó en el incremento III, se anexaron los 12 elementos de IFML que tiene representación gráfica, posteriormente el usuario selecciona qué elemento desea insertar en el área de modelado que es la pestaña Modelo Navegacional.

3.5.2 Diseño

Por cada elemento de IFML que tiene representación gráfica se crearon clases específicas que se muestran en los diagramas de clases parciales de *Figura 14* y *Figura 15*. Cabe

mencionar que no son todas las clases que incluyen los elementos de IFML, en el último incremento se anexan los diagramas finales.

En la *Figura 14* se muestra las clases necesarias para anexar los elementos de IFML en la lista (*ListView*); cada clase representa una opción en la lista de elementos seleccionable, con las características particulares que le corresponden (imagen específica del elemento, validaciones respecto a la colocación del elemento en el diagrama, figura gráfica asociada para el diagrama).

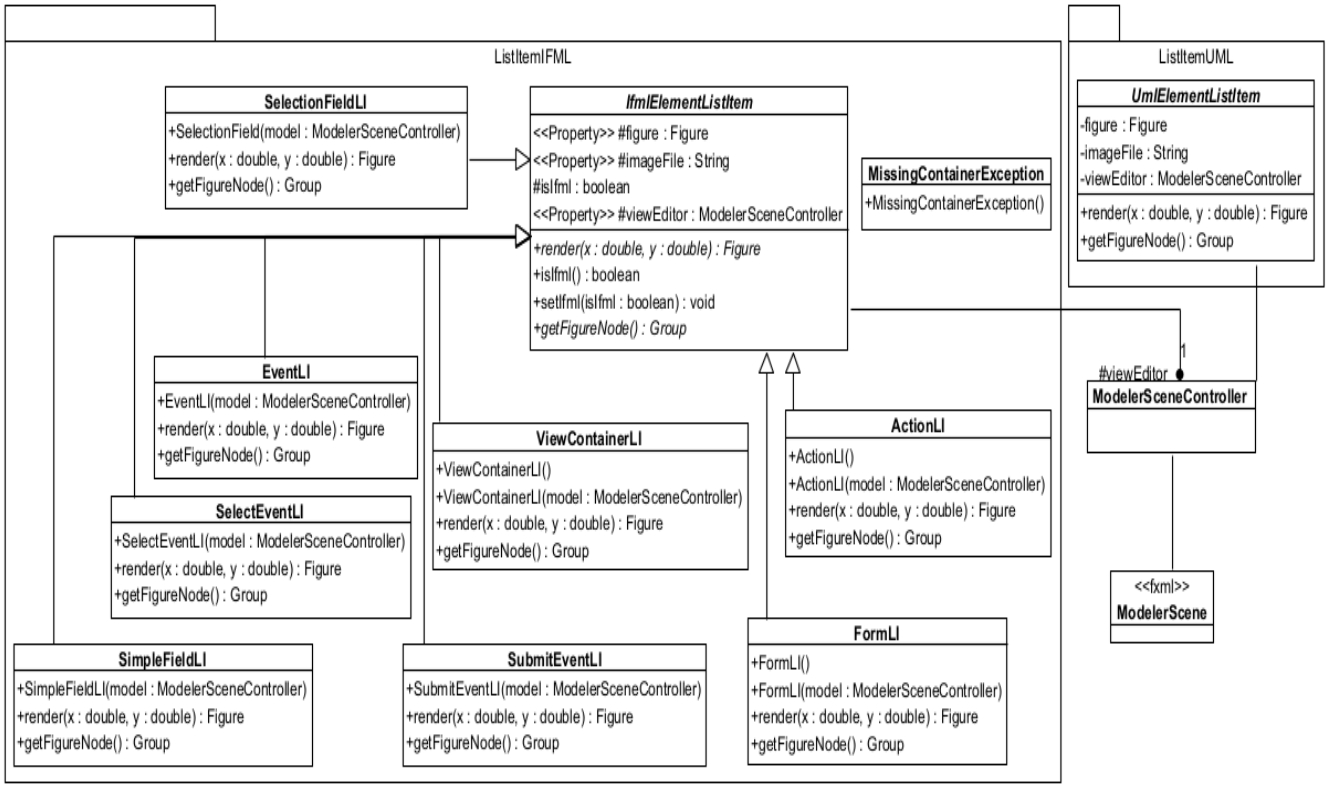

Figura 14 Diagrama de clases para agregar a la lista los elementos de IFML

En la *Figura 15* se muestran las clases más importantes para la representación gráfica de los elementos de IFML. La clase *Figure* es la más importante y es *abstracta*, de esta clase heredan *CustomRectangle*, *CustomPolygon*, *CustomCircle* y *CustomLine* que son las clases base para generar figuras que representan los elementos de IFML. Cada clase concreta equivale a la representación gráfica de un elemento IFML dentro del diagrama y las características visuales asociadas (si es un rectángulo, círculo o hexágono, qué movimientos permite, qué elementos anidados permite, si permite asociarse mediante líneas a otros elementos, entre otras).

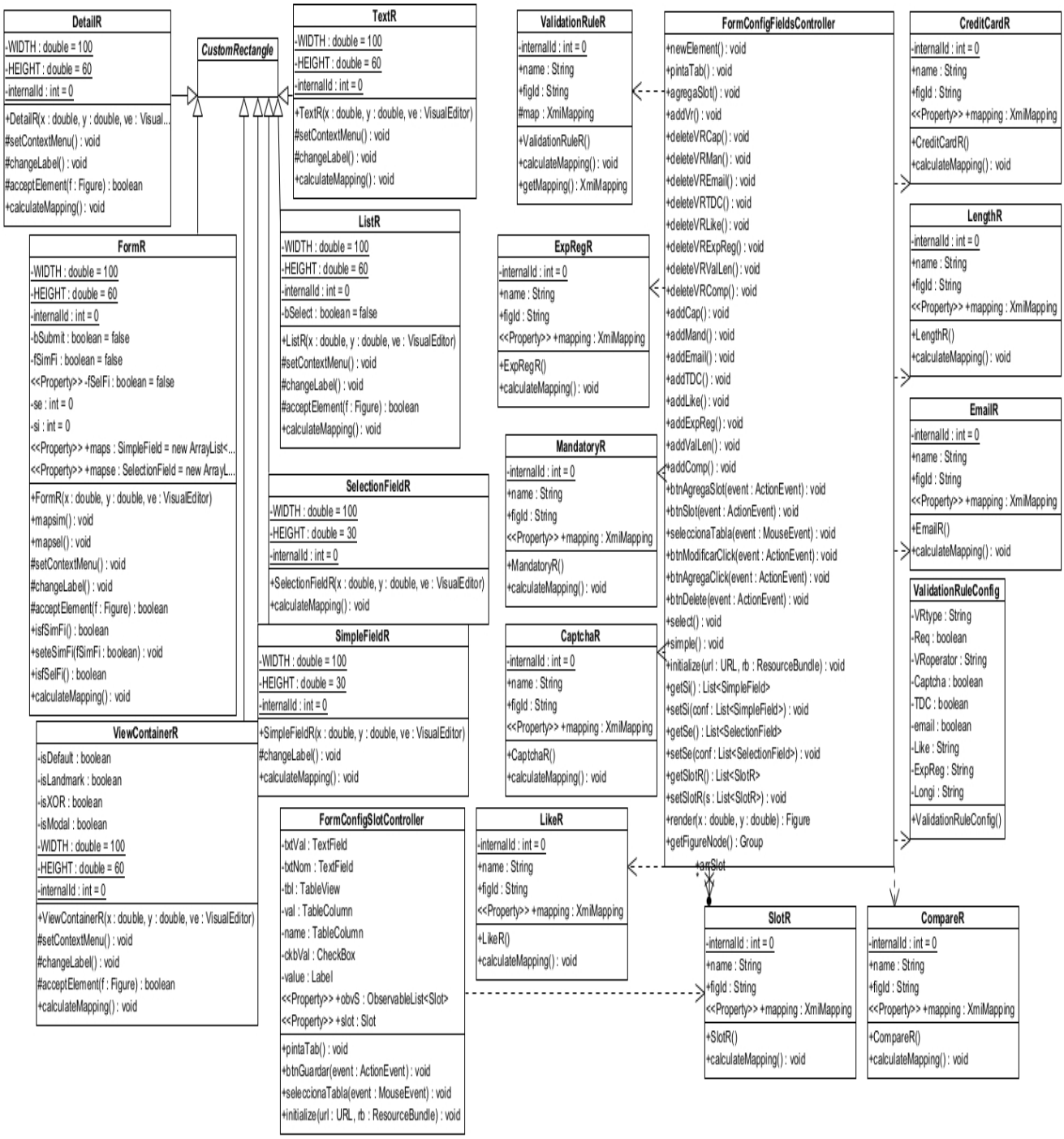

Figura 15 Diagrama de clases para representación gráfica de los elementos de IFML

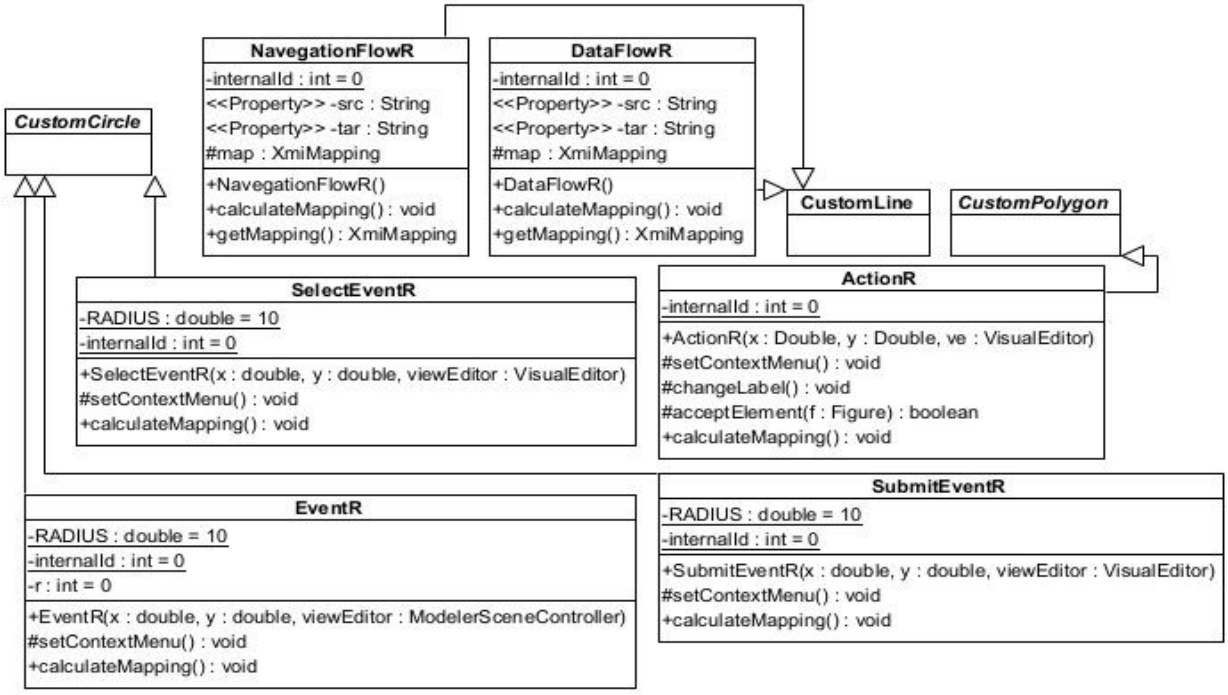

Figura 15 Continuación del diagrama de clases para representación gráfica de los elementos de IFML

3.5.3 Construcción

En la *Figura 16* se muestra la estructura actualizada de ITO's_iBRAIN, los elementos remarcados son los nuevos paquetes y clases que se agregaron para este incremento, los cuales son *resource.modeler.symbols* este paquete es el encargado de almacenar las imágenes que se agregan a la lista, *tesis.generator.enviroment.modeler.lisitemifml* este paquete contiene las clases representan las opciones en la lista de selección y que se relacionan con clases del paquete *tesis.generator.modeler.render*, donde se encuentran las clases encargadas de mostrar gráficamente los elementos de IFML.

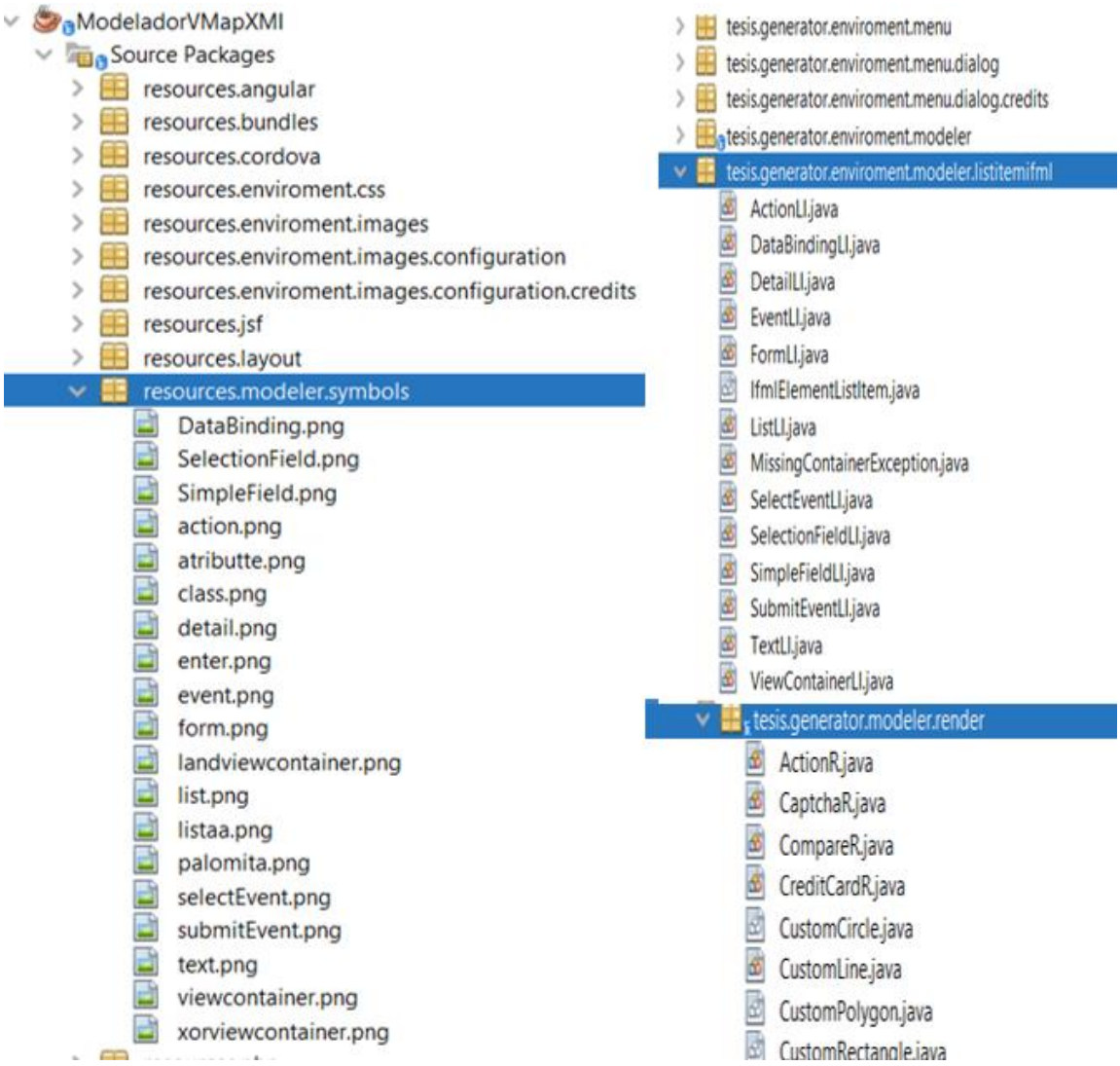

Figura 16 Estructura actualizada de la herramienta ITO´s_iBRAIN

A continuación, se detalla el funcionamiento a nivel de código para mostrar un elemento de IFML en el área de modelado.

En el *Listado 8* se muestra el método para agregar los elementos de IFML a una lista, en la línea 2 se define la lista observable (tipo especial de JavaFX), en las líneas 4 -15 se definen los elementos de la lista, en la línea 17 se agregan los elementos a la lista, en la línea 19 se define que sólo se puede seleccionar un elemento de la lista. En las líneas 29 y 30 se define la posición de los elementos en el área de modelado. Posteriormente se presenta un bloque `try-catch` a partir de la línea 31 en donde se definen ciertas validaciones como, por ejemplo, “El elemento elegido no es válido como elemento interno del elemento seleccionado en el editor” que se encuentra en la línea 35 y en la línea 32 y 33 se manda a

llamar el método *render* con las coordenadas donde se van a posicionar las figuras en el área de modelado.

Listado 8 Método fillNavigationElements() de la clase ModelerSceneController.java

```
1     public void fillNavigationElements() {
2         ObservableList<IfmlElementListItem> oNav
3                 = FXCollections.observableArrayList();
4         oNav.add(new ViewContainerLI(this));
5         ……..
15        oNav.add(new DataBindingLI(this));
16
17        oNavElemList.setItems(oNav);
18
19    oNavElemList.getSelectionModel().setSelectionMode(SelectionMode.SIN
      GLE);
20    ……..
23        oNavElemList.setOnMouseReleased(new
      EventHandler<MouseEvent>() {
24            @Override
25            public void handle(MouseEvent t) {
26                double x, y;
27                Figure f = null;
28                oNavElemList.setCursor(Cursor.HAND);
29                x = t.getSceneX() * (1.0 / getScale().getX());
30                y = (t.getSceneY() - 30) * (1.0 /
      getScale().getY());
31                try {
32                    f =
      oNavElemList.getSelectionModel().getSelectedItem()
33                            .render(x, y);
34                } catch (InvalidInnerElementException iiex) {
35                    showError("mod.error.invalidInnerElement");
36                } catch (UnaceptableInnerElementsException uiex) {
37    showError("mod.error.unacceptableInnerElement");
38                } catch (MissingContainerException mce) {
39                    showError("mod.error.externalElementMissing");
40                } catch (Exception ex) {
41                    showError("msj.unknownError");
42    Logger.getLogger(ModelerSceneController.class.getName()
43                    ).log(Level.SEVERE, null, ex);
44                }
45
46                if (f != null) {
47                    arrNavFigures.add(f);
```

```
48
rootNav.getChildren().add(oNavElemList.getSelectionModel()
49
.getSelectedItem().getFigureNode());
50                    }
51                }
52            });
53        }
```

Para mostrar un elemento (*ViewContainer*) de IFML en el área de modelado, se necesitan las siguientes clases: (1) *ViewContainerLI*, que representa la opción de un contenedor IFML dentro de la lista de elementos, esta clase hereda de la clase abstracta *IfmlElementListItem* que, a su vez, hereda de la clase *ImageView* (propia de JavaFX y que se relaciona con la presentación de imágenes en pantalla, provenientes de archivos JPG o PHP) y (2) *ViewContainerR*, que representa al contenedor IFML en el editor gráfico, es un subtipo de la clase *CustomRectangle* (rectángulos con la capacidad de moverse con el ratón, cambiar el tamaño, contener otros elementos dentro) y ésta hereda de la clase abstracta *Figure* (una figura que sabe en qué posición se encuentra dentro del editor gráfico, si tiene elementos anidados o no, si permite anidar elementos y de qué tipo, si tiene un menú contextual o no y, en su caso, las opciones básicas del menú como la asignación del nombre por ejemplo).

En el *Listado 9* se muestra la clase abstracta *IfmlElementListItem*, contiene métodos importantes como en la línea 5, el método *render*, que se relaciona con elementos del paquete `tesis.generator.modeler.render` y solicita como parámetros las coordenadas donde está el ratón y que es donde se colocará la figura asociada, el método retorna dicha figura. En la línea 36 retorna la figura como parte de un grupo. Los métodos que están en las líneas 9 – 34 son métodos *get* y *set* de las variables definidas en las líneas 2 – 5.

Listado 9 Clase abstracta IfmlElementListItem

```
1  public abstract class IfmlElementListItem extends ImageView{
2      protected Figure figure;
3      protected String imageFile;
4      protected boolean isIfml;
5      public ModelerSceneController viewEditor;
6
```

```
7       public abstract Figure render(double x, double y) throws
Exception;
8
9       public Figure getFigure() {
10          return figure;
11      }
        ……
32      public void setViewEditor(ModelerSceneController viewEditor) {
33          this.viewEditor = viewEditor;
34      }
35
36      public abstract Group getFigureNode();
37
38  }
```

Respecto al *Listado 10*, en la línea 1 indica que *ViewContainerLI* hereda de *IfmlElementListItem*, en las líneas 2 y 6 se definen los constructores de la clase *ViewContainerLI* uno sin parámetros y otro con parámetros, el constructor más importante es el de la línea 6 ya que se invoca en la clase *ModelerSceneController* (listado 8), en la línea 7 se indica la ruta de la imagen asociada a esta opción.

En la línea 13 está la implementación del método mencionado en *Listado 9*; en IFML es posible que un contenedor (*ViewContainer*) contenga a otros contenedores del mismo tipo, por lo tanto, en este método se asegura de que, si existe una figura ya seleccionada dentro del editor y se trata de un contenedor, el nuevo contenedor se colocará anidado al seleccionado (línea 15, además el método `addInnerElement` de `Figure` se asegura que sólo se aniden elementos válidos de acuerdo al lenguaje, lo que es una validación semántica); de otro modo, si no hay elementos seleccionados, entonces el nuevo contenedor se colocará en el editor en el punto indicado por el ratón (línea 14).

Listado 10 Clase ViewContainerLI

```
1   public class ViewContainerLI extends IfmlElementListItem{
2       public ViewContainerLI() {
3           this(null);
4       }
5
6       public ViewContainerLI(ModelerSceneController model){
7           this.imageFile =
"resources/modeler/symbols/viewcontainer.png";
```

```
8           this.setImage(new Image(this.imageFile));
9           this.viewEditor = model;
10      }
11
12      @Override
13      public Figure render(double x, double y) throws Exception{
14          if (viewEditor.getSelected() == null)
15              figure = new ViewContainerR(x ,y, viewEditor);
16          else{
17              viewEditor.getSelected().addInnerElement(
18                  new ViewContainerR(x ,y, viewEditor)
19              );
20              figure = null;
21          }
22          return figure;
23      }
24      @Override
25      public Group getFigureNode() {
26          return ((CustomRectangle)this.figure).getG();
27      }
28  }
```

Para entender el comportamiento de *ViewContainerR* se explican partes de las clases antecesoras *Figure* y *CustomRectangle*.

La clase *Figure*, tiene métodos importantes, se muestran algunos de los más relevantes en el *Listado 11*.

En las líneas 1 y 3 se encuentran métodos para seleccionar y mover las figuras, las líneas 7 y 9 son para saber qué figura esta seleccionada (el `overlay` es un rectángulo formado por pequeños cuadros que permite al usuario distinguir, en el editor, un elemento seleccionado respecto a los que no lo están), en las líneas 11 y 15 están las firmas de métodos que se llaman en los métodos relacionados con el ratón, la línea 19 tiene la firma para incluir un elemento dentro de otro mientras que la línea 21 tiene la firma para eliminar un elemento interno y en la línea 23 está la firma para realizar el mapeo de los elementos gráficos con su equivalente para los archivos XMI.

Listado 11 Métodos abstractos de la clase Figure

```
1       protected abstract void onMousePressed();
2
```

```
3     protected abstract void onMouseDragged();
4
5     protected abstract void changed();
6
7     protected abstract void iniOverlay();
8
9     protected abstract void updateOverlay();
10
11    protected abstract void moveTo(double x, double y);
12
13    protected abstract void handleMouse(Shape figure);
14
15    protected abstract void select();
16
17    protected abstract void handleMouseCorner(Rectangle rectangle);
18
19    public abstract void addInnerElement(Figure f) throws
      Exception;
20
21    public abstract void removeInnerElement(Figure f) throws
      Exception;
22
23    public abstract void calculateMapping();
```

En el *Listado 12* se muestra parte del código clase *CustomRectangle* en la línea 1 está el constructor de la clase que tiene como parámetros las coordenadas en donde se va a mostrar la figura (`x` y `y`, si es un elemento nuevo las origina el ratón, si es un elemento anidado dentro de otro las calcula y proporciona el elemento exterior), `width y height` son el ancho y la altura del rectángulo a mostrar puesto que varía según el elemento particular de IFML, también se recibe el color de la figura (en IFML, los elementos internos como los formularios tienen fondo gris), `lin` y `tex` son parámetros booleanos que indican si la figura del rectángulo tendrá una línea en la parte superior del rectángulo y si tendrá texto en la parte superior del rectángulo respectivamente (en IFML, los contenedores tienen una línea superior, pero los campos no; los contenedores tienen el nombre en la parte superior y los campos en el centro) , `text2` es el parámetro que lleva el texto a mostrar en el rectángulo y, por último, el parámetro *bArc* indica si las esquinas son redondeadas o no (los contenedores tienen esquinas redondeadas, los campos tienen esquinas normales).

En la línea 63 está el método que permite saber si se admiten elementos anidados o no; por omisión no se permite, pero este método se sobrescribe en clases como *ViewContainerR* que permite elementos internos pero sólo de cierto tipo.

Listado 12 Fragmento de código de la clase CustomRectangle

```
1  public CustomRectangle(double x, double y, double width, double height,
2                  Color color, VisualEditor ve,
3                  boolean lin, boolean tex, String tex2, boolean bArc) {
4              viewEditor = ve;
5              g = new Group();
6              initialHeight = height;
7              initialWidth = width;
8              previousX = x;
9              previousY = y;
10             arrRelationships = new ArrayList<>();
11             innerElements = new ArrayList<>();
12
13             if (tex) {
14                 label = new Label();
15                 label.setText(tex2);
16             }
17
18             if (x > 0 && y > 0) {
19                 rectangle = new Rectangle(x, y, initialWidth,
initialHeight);
20                 rectangle.setFill(color);
21                 rectangle.setCursor(Cursor.MOVE);
22
23                 if (x == 1 && y == 1) {
24                     rectangle.setStroke(Color.TRANSPARENT);
25                 } else {
26                     rectangle.setStroke(Color.BLACK);
27                 }
28
29                 g.getChildren().add(rectangle);
30
31                 if (bArc) {
32                     rectangle.setArcHeight(35);
33                     rectangle.setArcWidth(30);
34                 }
35
36                 if (lin) {
37                     line = new Line();
38                     line.setStartX(x);
```

```
39                  line.setStartY(y + LINE_GAP_Y);
40                  line.setEndX(x + rectangle.getWidth());
41                  line.setEndY(y + LINE_GAP_Y);
42                  line.startXProperty().bind(rectangle.xProperty());
43  line.startYProperty().bind(rectangle.yProperty().add(LINE_GAP_Y));
44                  g.getChildren().add(line);
45              }
46
47              if (tex) {
48                  label = new Label();
49                  label.setText(tex2);
50                  label.setLayoutX(x);
51                  label.setLayoutY(y);
52  label.layoutXProperty().bind(rectangle.xProperty().add(5));
53  label.layoutYProperty().bind(rectangle.yProperty().add(5));
54                  g.getChildren().add(label);
55              }
56
57              onMousePressed();
58              onMouseDragged();
59              changed();
60              iniOverlay();
61          }
62      }
63  protected boolean acceptElement(Figure f) {
64          return false;
65      }
```

A continuación, en el *Listado 13* se muestran fragmentos de código de la clase `ViewContainerR` que es la encargada de la representación gráfica del elemento *ViewContainer* de IFML.

En la línea 1 se define el nombre de la clase y que está heredando de `CustomRectangle`, en las líneas 3 – 8 se definen los atributos propios del contenedor que no tienen otros elementos de IFML, en la línea 11 está el constructor con las especificaciones de la figura a mostrar en el editor, en este caso se solicita una figura sin bordes redondeados, color blanco, con una línea en la parte superior y texto, el contenido del texto es "*ViewContainer*", posteriormente en la línea 23 está el método para el menú contextual que tiene como elementos *Default*, *Landmark*, *XOR* y *Modal*, estos valores son

estereotipos para el elemento *ViewContainer*; en la línea 32 está el método para cambiar el nombre del contenedor y, si algún estereotipo está seleccionado, se agrega su texto equivalente en la parte superior de la figura. Y, por último, en la línea 44, está el método para aceptar elementos internos, en el caso de *ViewContainer* se aceptan *ViewContainer*, *Form*, *Details*, *List*, *Event*, *Text* y *Action*.

Listado 13 Fragmentos de código de la clase ViewContainerR

1	**public class ViewContainerR extends CustomRectangle {**
2	
3	private boolean isDefault;
	…..
8	private static int internalId = 0;
10	
11	public ViewContainerR(double x, double y, VisualEditor ve) {
12	super(x, y, WIDTH, HEIGHT, Color.WHITE, ve,
13	true, true, "ViewContainer", false);
14	internalId++;
15	this.isNavigational = true;
16	this.figId = "vc_" + internalId;
17	this.name = "ViewContainer";
18	
19	this.label.setFont(new Font("Arial", 10));
20	this.expectedInnerPosition = HORIZONTAL_POSITION;
21	}
22	@Override
23	protected void setContextMenu() {
24	MenuItem mnuDefault = new MenuItem("Default"),
25	mnuLandmark = new MenuItem("Landmark"),
26	mnuXOR = new MenuItem("XOR"),
27	mnuModal = new MenuItem("Modal");
28	super.setContextMenu();
	…….
29	contextMenu.getItems().addAll(mnuDefault, mnuLandmark, mnuXOR, mnuModal);
30	}
31	@Override
32	protected void changeLabel() {
33	String sVal = "";
34	sVal = (this.isModal ? "<<Modal>> " : "")
35	+ (this.isDefault ? "[D] " : "")
36	+ (this.isLandmark ? "[L] " : "")
37	+ (this.isXOR ? "[XOR] " : "")
38	+ this.name;
39	this.label.setText(sVal);

```
40   if (this.rectangle.getWidth() < this.label.getWidth()) {
41               setHSize(this.getLeftX() + this.label.getWidth(),
     false);
42           }   }
43   @Override
44       protected boolean acceptElement(Figure f) {
45           boolean bRet = false;
46
47           if (f instanceof ViewContainerR
48                   || f instanceof FormR || f instanceof DetailR || f
     instanceof ListR
49                   || f instanceof EventR || f instanceof TextR || f
     instanceof ActionR) {
50               bRet = true;
51           }
```

La misma relación entre un elemento de la lista (*`ElementoLI`*) y su representación gráfica (*`ElementoR`*) explicados anteriormente para el caso de los contenedores (*ViewContainer* en IFML) se replica para el resto de los elementos del modelo navegacional (*Form, List, Details, Text, Action, Event, SelectEvent, SubmitEvent, SimpleField, SelectionField y DataBinding*). Es importante destacar que algunos elementos, como los campos, requieren una configuración particular pues, si se agregan de la misma manera que los otros (anidados dentro de un formulario en este caso), el diagrama resulta demasiado grande y complejo de leer. Herramientas como IFMLEdit.org y WebRatio también emplean la misma solución (sólo se muestra un elemento y la configuración se hace por separado, aunque se trate de muchos campos).

3.5.4 Despliegue

En esta sección se muestran las pruebas realizadas para el cuarto incremento. En la *Figura 17* se muestra el ambiente de modelado en ITO's_iBRAIN, en la lista se muestra seleccionado el elemento *ViewContainer* (como instancia de la clase `ViewContainerLI`) y en el área de modelado se muestra dos contenedores anidados (instancias de `ViewContainerR`), de igual manera se observa que está seleccionado el contenedor externo.

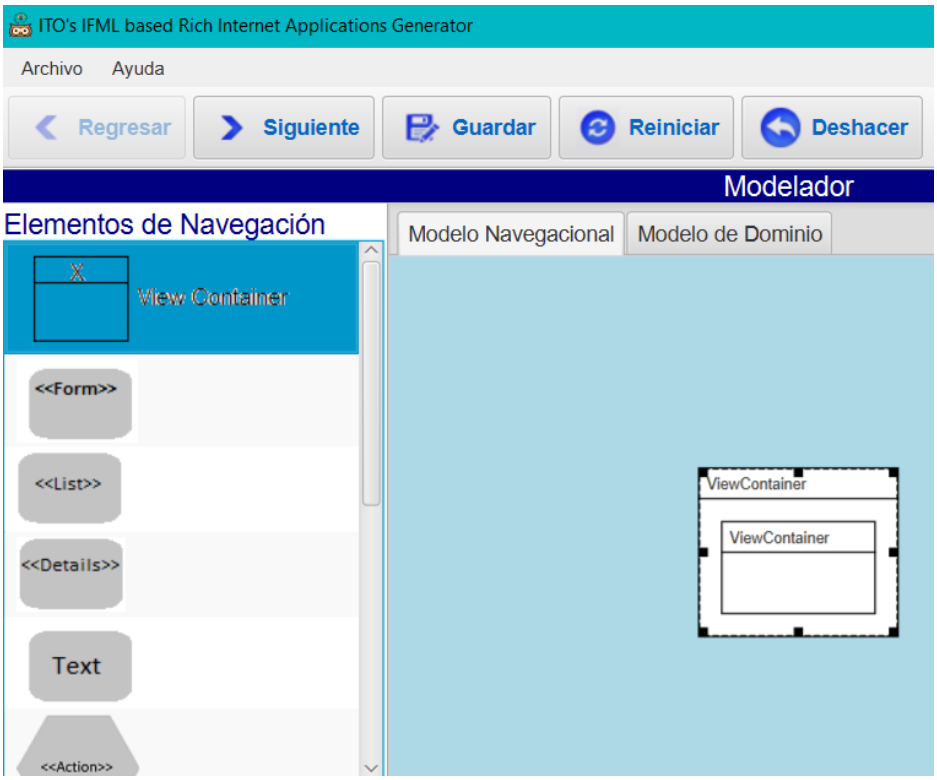

Figura 17. Representación gráfica del elemento ViewContainer

En este punto, el modelador no sólo es capaz de mostrar gráficamente el elemento *ViewContainer*, también es capaz de mostrar los elementos de *Form*, *List*, *Details*, *Text*, *Action*, *Event*, *SelectEvent*, *SubmitEvent*, *SimpleField*, *SelectionField* y *DataBinding*. (*Figura 18*).

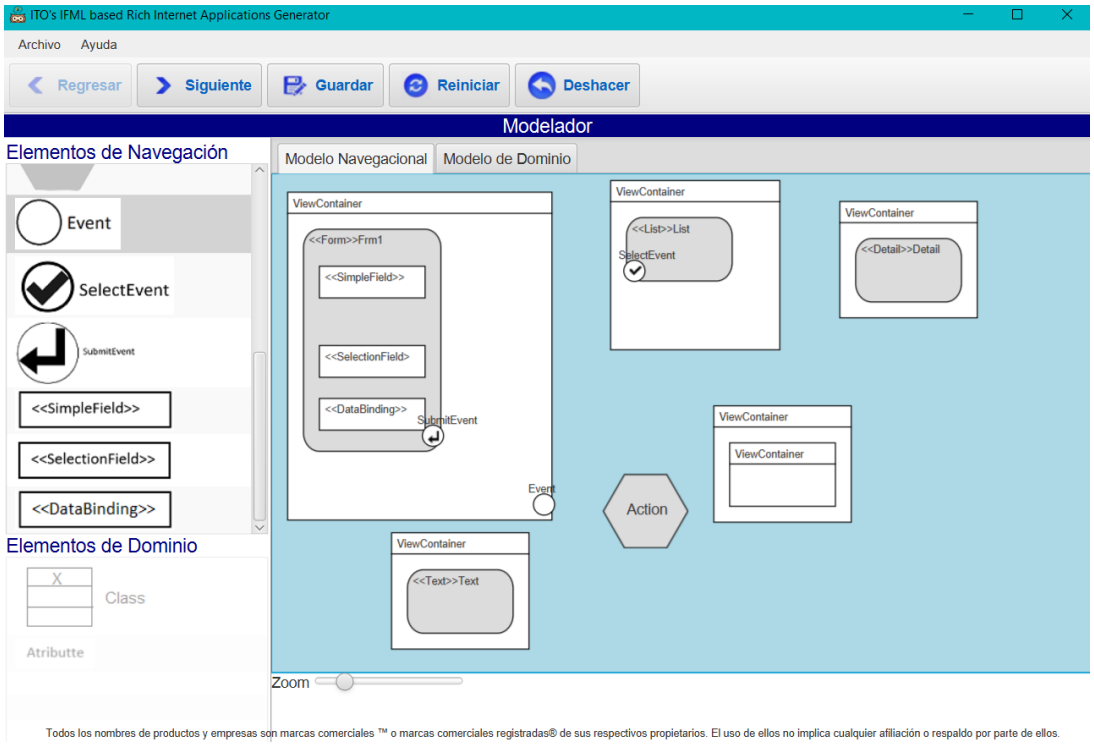

Figura 18. Representación gráfica de los elementos de IFML

Los elementos *Action* se instancian con la clase `ActionR` que hereda de la clase `CustomPolygon` y los elementos *Event*, *SubmitEvent* y *SelectionEvent* se instancian con las clases `EventR`, `SubmitEventR` y `SelectionEventR` respectivamente, que heredan de la clase `CustomCircle`.

3.6 Incremento V Ventana modal para el elemento *Form* de IFML

Debido a que el elemento *Form* de IFML acepta como elementos internos a *SelectionField* y *SimpleField*, éstos necesitan de ciertas configuraciones para el nombre, tipo y validaciones asociadas (clase `ValidationRules`) y, en el caso de *SelectionField*, además, maneja opciones internas (*Slots*).

3.6.1 Análisis de requerimientos

Los campos sencillos (*SimpleField*) necesitan configurar reglas de validación para asegurar que los códigos resultantes validen la captura de información en ellos; las validaciones soportadas son:

- Campo requerido
- Comparación contra un valor (comparativa) menor, mayor, igual, distinto, entre otros
- *Captcha*
- Tarjeta de crédito (formato de 16 dígitos)
- Formato de correo electrónico (*email*)
- Comparación contra un valor semejante (*like*)
- Expresión regular
- Longitud en número de caracteres
- Tipo de dato específico (si se relaciona con un atributo del Modelo de Dominio, el tipo de dato es consistente con él)

Para los campos de selección única o múltiple (*SelectionField*) las reglas de validación aceptables son: campo requerido y comparación contra un valor (comparativa). Además, tienen dos acciones adicionales que son: agregar opciones (*slots*, inician con un nombre y valor por omisión) y ajustar las opciones (nombre y valor).

3.6.2 Diseño

Igual que en el Incremento III, se utilizó la herramienta *Balsamiq Mockups* para maquetar la forma visual de la configuración mencionada, tomando en cuenta que en el editor gráfico sólo se muestra, a lo más, un campo sencillo y un campo de selección para evitar sobrecargar el diagrama. En la *Figura 19* se muestra el maquetado de la ventana modal que se presentará al elegir la opción *SimpleField* del menú contextual del elemento *Form*, esta opción de menú sólo es válida si el formulario tiene previamente anidado un campo simple (*SimpleField*).

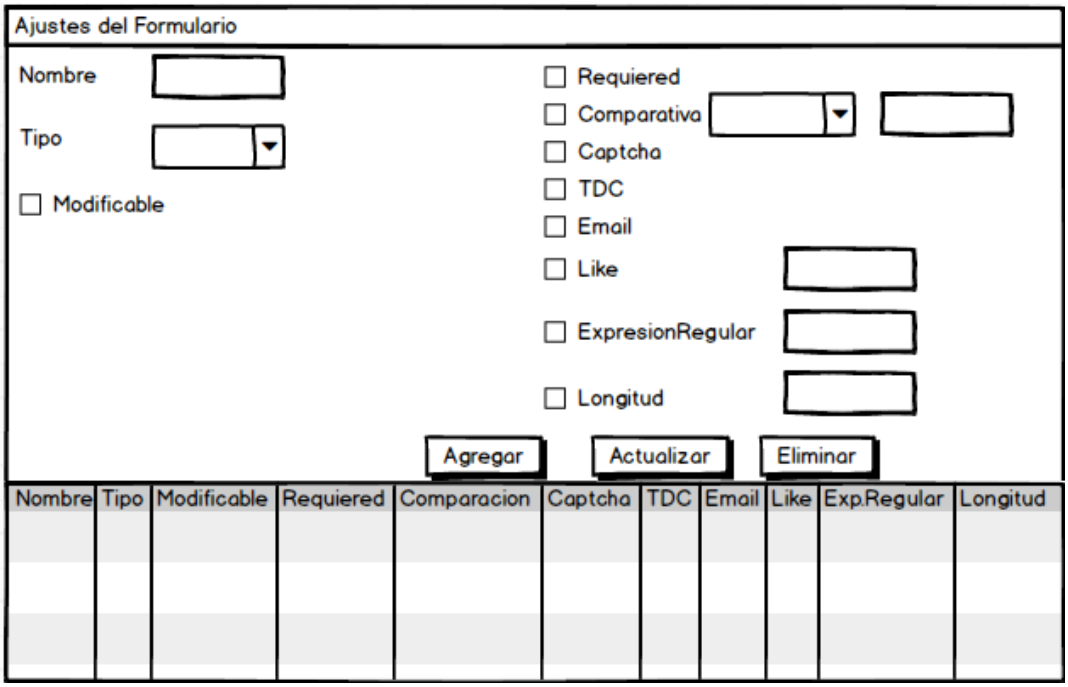

Figura 19. Diseño de ventana modal para SimpleField

En la *Figura 20* se muestra el maquetado de la ventana modal para la opción *SelectionField* en el menú contextual de *Form*; igual que en el caso anterior, esta opción de menú sólo es válida si previamente se agregó al formulario un campo de selección.

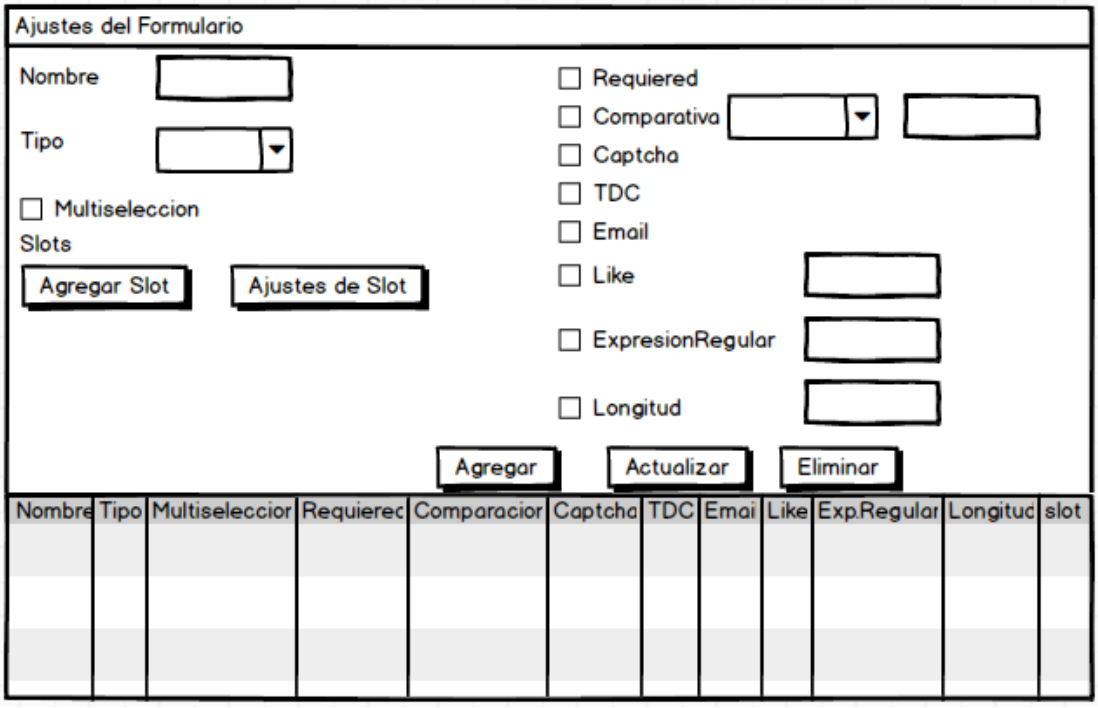

Figura 20. Diseño de ventana modal para SelectionField

En la *Figura 21* se muestra el maquetado de la ventana modal para las opciones dentro del campo de selección (*slots*), misma que aparece como resultado del botón Ajustes de *Slot* que se encuentra en la ventana modal de la *Figura 20.*

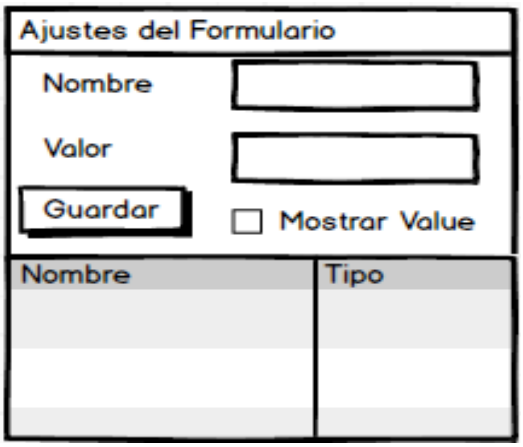

Figura 21. Diseño de ventana modal para Slots

3.6.3 Construcción

En la *Figura 22* se observa parte de la estructura de ITO's_iBRAIN una vez agregadas las clases para soportar los diseños mencionados en los puntos anteriores; los elementos que aparecen con fondo azul son los encargados de que funcionen correctamente las ventanas modales.

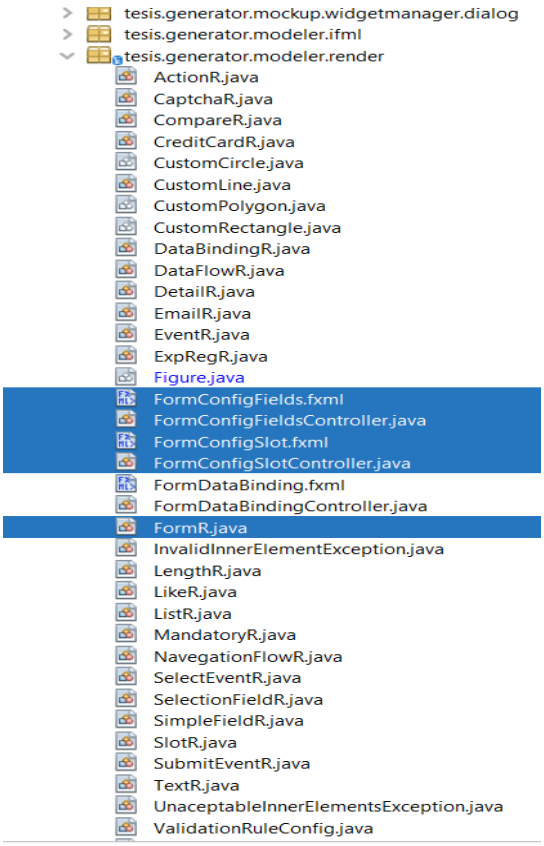

Figura 22. Clases para las ventanas modales

En el *Listado 14* se muestran fragmentos de código de la clase `FormR` con los métodos para abrir las ventanas modales por medio de un menú contextual. En la líneas 5 y 6 se especifica la ruta donde se encuentra el archivo *FXML* que contiene los elementos de la ventana modal, en la línea 9 se llama al método que recibe la lista de elementos anidados que existen en el formulario; en la línea 12 se llama a un método de la clase `FormR` que inicializa el mapeo de los elementos de tipo `SimpleField` del formulario con su equivalente en XMI; en la línea 14 se llama al método de la ventana modal antes de que se muestre en pantalla, para configurar el comportamiento específico para campos sencillos (debido a la semejanza en la configuración de campos, se construyó sólo una ventana modal que se ajusta dependiendo si se trata de campos sencillos o campos de selección). En la línea 15 se invoca un método de la ventana modal para enviarle los elementos de tipo `SimpleField` asociados al formulario para su edición en dicha ventana modal. Los métodos utilizados en las líneas 34 – 65 son similares a los ya mencionados, pero para el caso de los campos de selección.

Listado 14 Fragmentos de código de la clase FormR

```
1   mnuEditarSim.setOnAction(new EventHandler<ActionEvent>() {
2               @Override
3               public void handle(ActionEvent event) {
4                   try {
5                       FXMLLoader fxmlLoader = new
FXMLLoader(getClass().
6   
getResource("/tesis/generator/modeler/render/FormConfigFields.fxml"
), ITOs_IBRAIN.getBundle());
7                       AnchorPane oWorkspace = (AnchorPane)
fxmlLoader.load();
8                       FormConfigFieldsController controller =
(FormConfigFieldsController) fxmlLoader.getController();
9                       controller.setConf(getInneElements());
10                      si++;
11                      if (si == 1) {
12                          mapsim();
13                      }
14                      controller.simple();
15                      controller.setSi(getMaps());
16                      controller.setOpcion(1);
17                      controller.pintaTab();
18  
19                      Scene scene = new Scene(oWorkspace);
20                      Stage stage = new Stage();
21                      stage.initModality(Modality.APPLICATION_MODAL);
22                      stage.setScene(scene);
23  
stage.setTitle(ITOs_IBRAIN.getBundle().getString("mod.dialog.settin
g"));
24                      stage.getIcons().add(new
Image("resources/enviroment/images/settings.png"));
25                      stage.showAndWait();
26  
27                  } catch (IOException e) {
28                      Logger.getLogger(MenuItem.class
29                              .getName()).log(Level.SEVERE, null, e);
30                  }
31              }
32          });
33  
34   mnuEditarSele.setOnAction(new EventHandler<ActionEvent>() {
35              @Override
36              public void handle(ActionEvent event) {
37                  try {
38                      FXMLLoader fxmlLoader = new
FXMLLoader(getClass().
```

```
39  getResource("/tesis/generator/modeler/render/FormConfigFields.fxml"
    ), ITOs_IBRAIN.getBundle());
40                      AnchorPane oWorkspace = (AnchorPane)
    fxmlLoader.load();
41                      FormConfigFieldsController controller
42                              = (FormConfigFieldsController)
    fxmlLoader.getController();
43                      controller.setConf(getInneElements());
44                      se++;
45                      if (se == 1) {
46                          mapsel();
47                      }
48                      controller.select();
49                      controller.setSe(getMapse());
50                      controller.setOpcion(2);
51                      controller.pintaTab();
52
53                      Scene scene = new Scene(oWorkspace);
54                      Stage stage = new Stage();
55                      stage.initModality(Modality.APPLICATION_MODAL);
56                      stage.setScene(scene);
57  stage.setTitle(ITOs_IBRAIN.getBundle().getString("mod.dialog.settin
    g"));
58                      stage.getIcons().add(new
    Image("resources/enviroment/images/settings.png"));
59                      stage.showAndWait();
60
61                  } catch (IOException e) {
62                      Logger.getLogger(MenuItem.class
63                              .getName()).log(Level.SEVERE, null, e);
64                  }
65              }
    });
```

En el *Listado 15* se muestran fragmentos del código de vista `FormConfigFields.fxml` donde se encuentran los elementos gráficos, propios de JavaFX, que componen a la ventana modal; las restricciones respecto al ajuste visual relacionado con campos sencillos o campos de selección se encuentran en la clase `FormConfigFieldsController` que se presenta en el *Listado 16*.

Listado 15 Fragmento de código de FormConfigFields.fxml

```
1  <AnchorPane id="AnchorPane" prefHeight="475.0" prefWidth="970.0"
   xmlns:fx="http://javafx.com/fxml/1"
2  fx:controller="tesis.generator.modeler.render.FormConfigFieldsContr
   oller">
3      <BorderPane fx:id="oRoot" prefWidth="875.0" minHeight="475.0"
4                  AnchorPane.bottomAnchor="1.0"
5                  AnchorPane.leftAnchor="1.0"
6                  AnchorPane.rightAnchor="1.0"
7                  AnchorPane.topAnchor="1.0">
8          <left>
9              <GridPane minWidth="400" hgap="5" vgap="5">
10                 <children>
11                     <Label text="%mod.ctxmnu.name"
   GridPane.columnIndex="0"
12                            GridPane.rowIndex="0"/>
13                     <TextField fx:id="txtNomb"
   GridPane.columnIndex="1"
14                                GridPane.rowIndex="0"/>
   …….
26                     <Button fx:id="btnSlott"
   mnemonicParsing="false"
27                             onAction="#btnSlot"
   text="%mod.dialog.slot.setting"
28                             GridPane.columnIndex="1"
   GridPane.rowIndex="6"/>
29                 </children>
30             </GridPane>
31         </left>
32 <center>
33             <GridPane minWidth="400" hgap="9" vgap="9"
   fx:id="oValidaciones"  centerShape="false" >
34                 <children>
35                     <CheckBox text="Requiered" fx:id="ckbReq"
   GridPane.columnIndex="0"
   …….
36                      <Button fx:id="btnDelete"
   mnemonicParsing="false"
37                             onAction="#btnDelete" text="Eliminar"
38                             GridPane.columnIndex="2"
   GridPane.rowIndex="9"/>
39                 </children>
40             </GridPane>
41         </center>
42  <bottom>
43             <TableView fx:id="tbl" prefHeight="200.0"
   prefWidth="810.0"     onMousePressed="#seleccionaTabla">
44                 <columns>
45                     <TableColumn prefWidth="125.0"
```

```
      text="%mod.ctxmnu.name"
46                                        minWidth="25" fx:id="nom"/>
      ……
47                          <TableColumn prefWidth="50.0" text="Atributo"
48                                        minWidth="60" fx:id="att"/>
49                    </columns>
50                </TableView>
51            </bottom>
52        </BorderPane>
53  </AnchorPane>
```

En el *Listado 16* se muestran los atributos relacionados con la interfaz gráfica de la ventana modal (anotados como @FXML, líneas 3 a 18), los atributos sin representación gráfica (líneas 20 a 33) y los nombres de los métodos de la clase `FormConfigFieldsController` (no son abstractos, no se muestra el contenido por falta de espacio); en la línea 1 muestra que hereda de `IfmlElementListItem`, debido al comportamiento que permite agregar nuevas instancias *SimpleField*, *SelectionField* o *Slots.* En la línea 35 está el inicio del método para agregar nuevos elementos, este método se manda a llamar en la línea 66 como resultado del evento Clic del botón Agregar; en la línea 36 está el método `pintaTab`, que se invoca desde la clase `FormR` y que permite mostrar los datos de los campos ya agregados al formulario. En la línea 37 está el método agregar nuevos *slots*, este método solo se manda a llamar cuando se muestra la ventana modal para los elementos de *SelectionField*, en la línea 39 – 46 esta los métodos para eliminar las reglas de validación que se han agregado, de las líneas 48 – 55 están los métodos para agregar las reglas de validación, estos métodos se mandan a llamar como resultado del evento Clic de los botones correspondientes.

Listado 16 Fragmentos de código de la clase FormConfigFieldsController

```
1  public class FormConfigFieldsController extends IfmlElementListItem
   implements Initializable {
2
3      @FXML
4      private ResourceBundle bundle;
5      @FXML
6      private ComboBox cmbTip, cmbComp;
7      @FXML
8      private CheckBox ckbMod, ckbReq, ckbComp, ckbCapt, ckbTDC,
   ckbEmail, ckbLike,   ckbExp, ckbLong;
9      @FXML
```

```
10      private TextField txtNomb, txtLike, txtExpReg, txtLong,
        txtComp;
11      @FXML
12      private TableView tbl;
13      @FXML
14      private TableColumn nom, tip, mod, req, com, cap, tdc, email,
        like, expReg, longi, slot1;
15      @FXML
16      private Button btnSlott, btnAgregaSlot;
17      @FXML
18      private Label lSlot;
19
20      private ObservableList<SimpleField> datosSim;
21      private ObservableList<SelectionField> datosSele;
22      private List<SimpleField> arrInnerSim;
23      private List<SelectionField> arrInnerSelect;
24
25      private List<SlotR> arrSlot = new ArrayList<>();
26      private List<Slot> slot = new ArrayList<>();
27
28      private ArrayList<Figure> conf;
29      private int opcion = 0;
30      private SelectionField obj;
31      private SimpleField objs;
32      private  int i = 0;
33      private ArrayList<Figure> copia;
34
35  public void newElement()
36  public void pintaTab()
37  public void agregaSlot()
38  public void addVr()
39  public void deleteVRCap()
40  public void deleteVRMan()
41  public void deleteVREmail()
42  public void deleteVRTDC()
43  public void deleteVRLike()
44  public void deleteVRExpReg()
45  public void deleteVRValLen()
46  public void deleteVRComp()
47
48  public void addCap()
49  public void addMand()
50  public void addEmail()
51  public void addTDC()
52  public void addLike()
53  public void addExpReg()
54  public void addValLen()
```

```
55  public void addComp()
56
57  @FXML
58      public void btnAgregaSlot(ActionEvent event)
59  @FXML
60      public void btnSlot(ActionEvent event)
61  @FXML
62      public void seleccionaTabla(MouseEvent event)
63  @FXML
64      public void btnModificarClick(ActionEvent event)
65  @FXML
66      public void btnAgregaClick(ActionEvent event) throws Exception
67  @FXML
68      public void btnDelete(ActionEvent event) throws Exception
69
70  public void select()
71  public void simple()
```

3.6.4 Despliegue

En esta sección se encuentran las pruebas realizadas para el quinto incremento. En la *Figura 23* se muestra el ambiente de modelado de ITO's_iBRAIN, con cuatro elementos de IFML: *ViewContainer*, *Form*, *SelectionField* y *SimpleField*. Se muestra el menú contextual con la opción seleccionada Ajustes *SimpleField*.

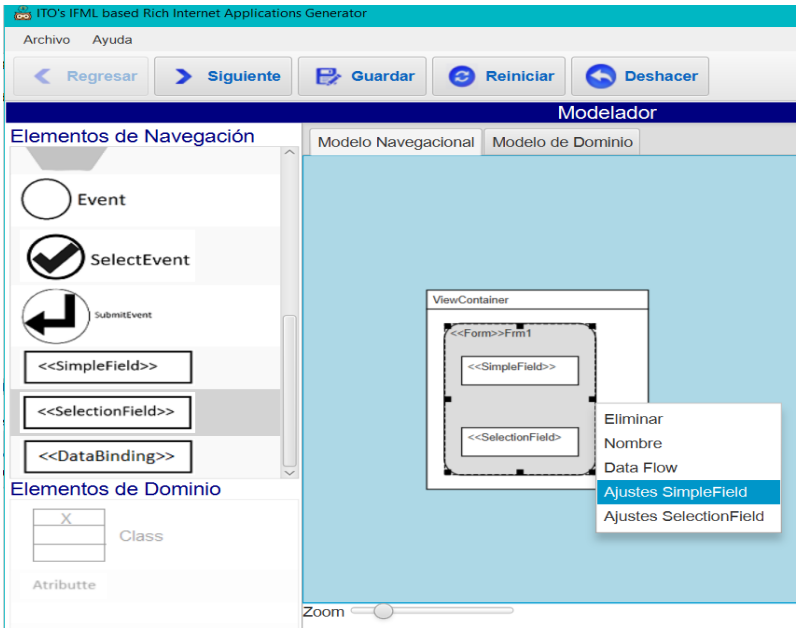

Figura 23. Menú Contextual del elemento Form

Al dar clic en la opción del menú, se muestra la ventana modal como en la *Figura 24*, misma que es capaz de agregar, actualizar o eliminar campos sencillos (*SimpleField*); cada que se oprime un botón se realizan ciertas operaciones; por ejemplo, cuando se agrega un campo sencillo, además de afectar la lista de elementos anidados al formulario, se muestra una ventana emergente mostrando el texto "elemento agregado"; cuando se desea eliminar un elemento, éste se selecciona con doble clic sobre la tabla y al oprimir el botón Eliminar, además de remover el campo de la lista de elementos anidados al formulario, se muestra una ventana emergente con el texto "Elemento Eliminado"; el botón Actualizar, permite modificar la información de un campo que ya se encuentra en la tabla. En el caso de reglas de validación, se verifican antes de asociarlas a un campo, si no son correctas se eliminan del campo.

Cada que se agrega, actualiza o elimina un campo, se actualiza la tabla que se ubica en la parte inferior de la ventana modal (*Figura 24*).

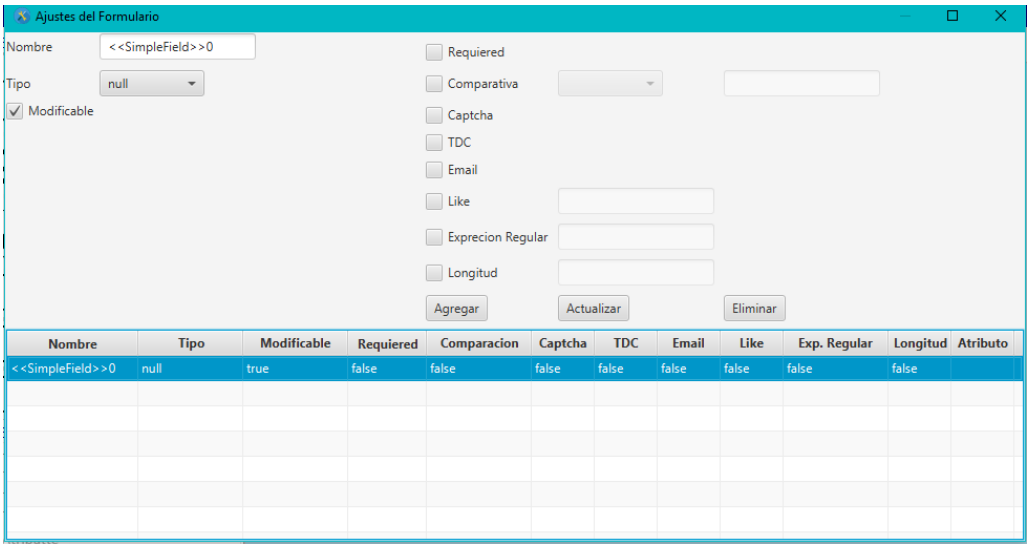

Figura 24. Ventana modal SimpleField

Al dar clic en la opción Ajustes *SelectionField* del menú contextual del `Form`, se muestra la ventana modal de la *Figura 25*; no es muy diferente de la ventana modal de *SimpleField*, los únicos cambios son los botones "Agregar *Slot*" y "Ajustes de *Slot*" así como la inhabilitación de las reglas de validación no aplicables a este tipo de campos, dejando activas únicamente las dos primeras. Al agregar un nuevo *slot*, se muestra una ventana emergente con el texto "elemento agregado (nombre del elemento)". El funcionamiento de

los botones agregar, actualizar o eliminar son iguales a la ventana modal de *SimpleField*, pero ahora con los elementos de *SelectionField*.

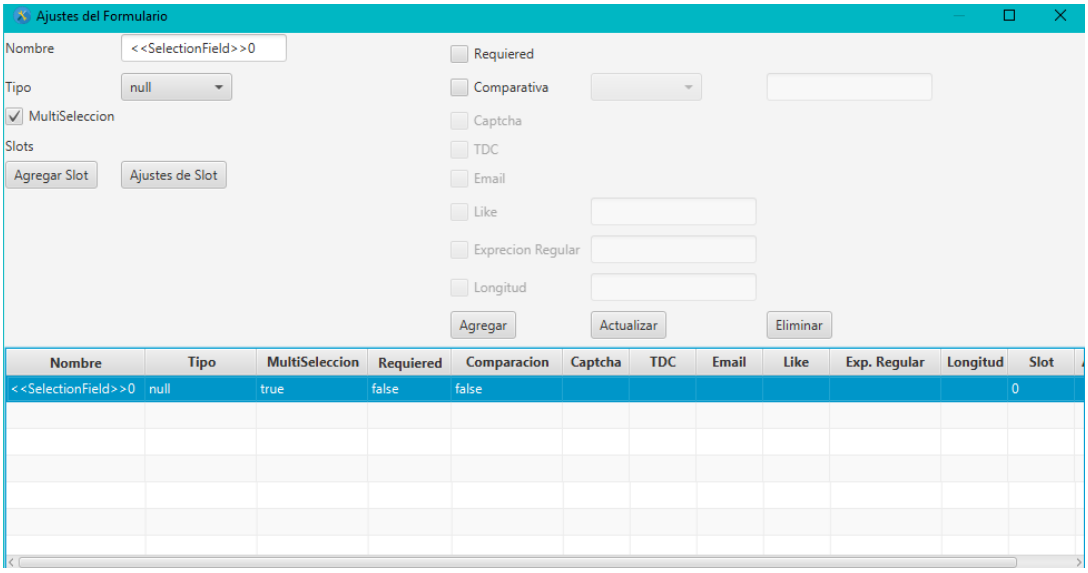

Figura 25. Ventana modal SelectionField

Al dar *clic* en el botón "Ajustes de Slot" se muestra una ventana modal como en la *Figura 26*. Ahí se presentan los *slots* que estén agregados; al dar doble *clic* sobre un elemento de la tabla, el contenido seleccionado pasa a los campos que están en la parte superior de la ventana modal; si el campo de marcado (*CheckBox*) está seleccionado entonces se incluye tanto el valor como el nombre de la opción; esto es porque en los códigos de campos de selección se distingue entre la información que se muestra en pantalla (en este caso el Nombre) y la información que maneja internamente la aplicación resultante (en este caso el Valor).

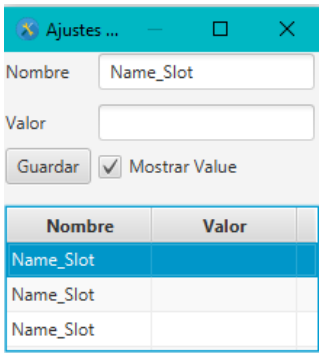

Figura 26. Ventana modal para Slots

3.7 Incremento VI Representación gráfica de las conexiones (*DataFlow* y *NavigationFlow*)

Al terminar la representación gráfica de los elementos de IFML, solo faltan las relaciones entre elementos; la que define IFML son *DataFlow* y *NavigationFlow*, en este incremento ambas se integran al modelador de ITO's_iBRAIN.

3.7.1 Análisis de requerimientos

Los elementos origen válidos para *NavigationFlow* son *Event*, *SelectEvent* y *SubmitEvent* que muestran en un menú contextual la opción de generar un elemento *NavigationFlow*, los elementos destino válidos son cualquier elemento de IFML excepto los elementos origen.

Para el elemento *DataFlow*, los posibles orígenes son *Form*, *List y Action*, sus destinos válidos son los elementos *Form* y *List*.

Los *DataFlow* están representados gráficamente por una línea punteada que al final (destino) tiene una cabeza de flecha cerrada y los *NavigationFlow* se representan con una línea continua con el mismo tipo de final en el destino que *DataFlow*.

3.7.2 Modelado

En la *Figura 27* se muestra la clase *CustomLine*, misma que contiene los metodos necesarios para representar líneas punteadas, continuas y la flecha al final de la línea.

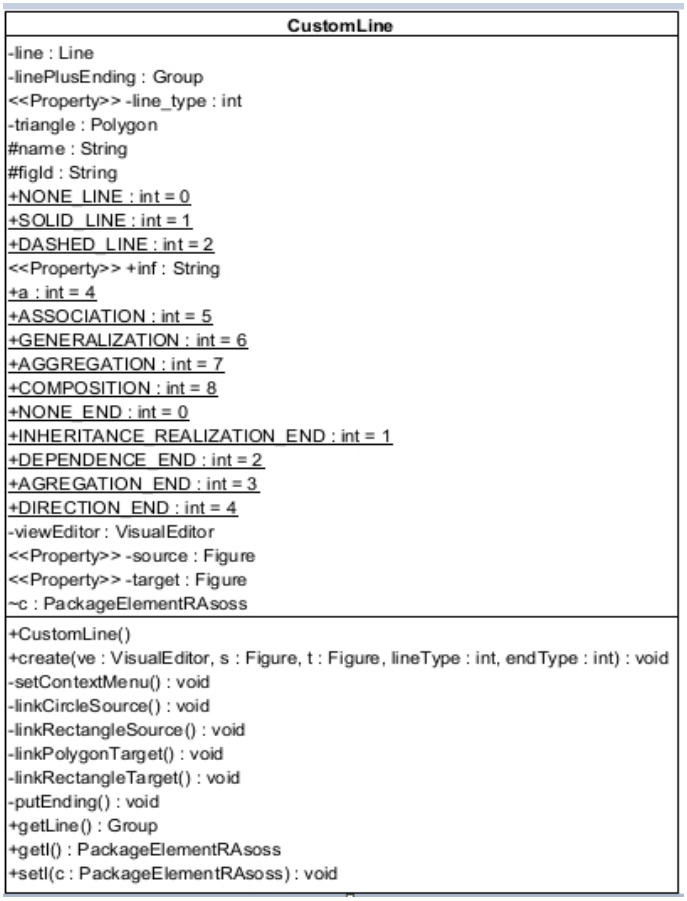

Figura 27. Clase CustomLine

En la figura 28 se muestra la jerarquía de clases *CustomLine*, *NavigationFlowR* y *DataFlowR*, es decir, las clases relacionada con la representación gráfica de los elementos mencionados de IFML.

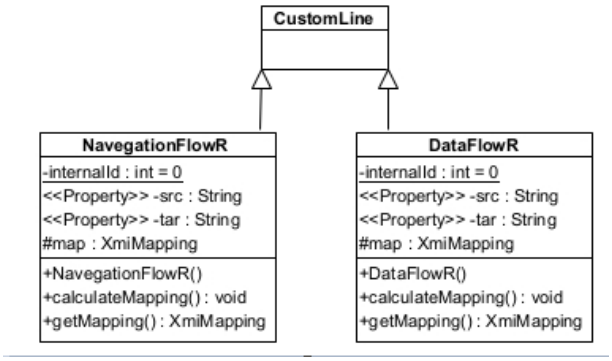

Figura 28. Representación de herencia para DataFlowR y NavigationFlowR

3.7.3 Construcción

Los elementos seleccionados de la estructura de ITO's_iBRAIN en la *Figura 29* son lo necesarios para crear las relaciones descritas en el punto anterior.

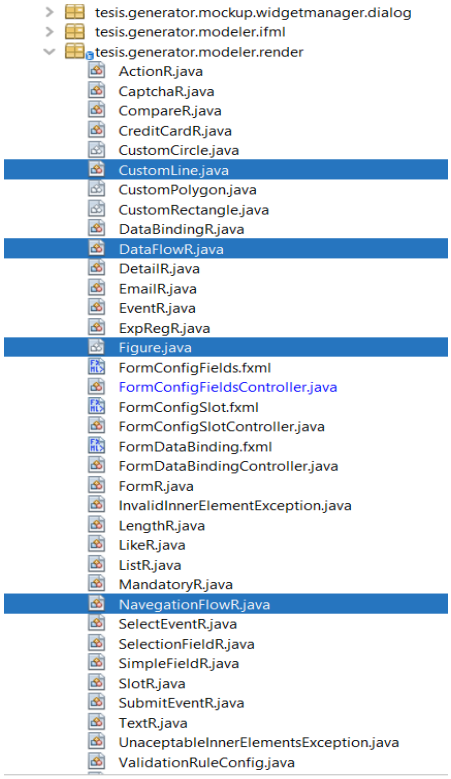

Figura 29. Elementos para DataFlow y NavigationFlow

En el *Listado 17* se muestra un fragmento de código de la clase *Figure*, relacionado con la representación gráfica de líneas entre elementos siempre y cuando el destino sea válido. Cada elemento IFML, heredero de Figure, define qué tipos de línea trabaja (línea 4) y qué destinos admite dependiendo del origen (línea 7), por lo que el método simplemente se hereda sin modificaciones.

Listado 17. Fragmento de código de la clase Figure para verificar si existe un tipo de línea

```
1  protected void renderRelationship() {
2          CustomLine cl;
3          Figure src;
4          if (viewEditor.isReadyForRelationship()) {
5              src = viewEditor.getSelectedSource();
6
7              if (this.isValidTarget(src)) {
8                  if (src.getCurrentLineType() !=
CustomLine.NONE_LINE) {
```

```
9                       cl = new CustomLine();
10                      cl.create(viewEditor, src, this,
11                              src.getCurrentLineType(),
12                              src.getCurrendLineEnd());
13                      this.lineType = CustomLine.NONE_LINE;
14                  }
15              }
16              viewEditor.clearRelationships();
17          }
18  }
```

En el *Listado 18* se muestran fragmentos del código de la clase *EventR*; en las líneas 4 y 5 se indica el tipo de línea que se necesita, en este caso es para un elemento *NavigationFlow* (línea continua, cabeza de flecha en el destino).

Entre las líneas 10 y 21 se verifican las líneas existentes y se realiza el mapeo con XMI de acuerdo al tipo particular.

Listado 18. Fragmento de código de la clase EventR para instanciar un NavegationFlow

```
1   mnuNavigationFlow.setOnAction(new EventHandler<ActionEvent>() {
2               @Override
3               public void handle(ActionEvent event) {
4                   lineType = CustomLine.SOLID_LINE;
5                   lineEndType = CustomLine.DIRECTION_END;
6
7               }
8           });
9
10  if (!arrRelationships.isEmpty()) {
11              for (CustomLine e : arrRelationships) {
12                  if (e.getLine_type() == 1) {
13                      NavegationFlowR nf = new NavegationFlowR();
14  nf.setSrc(arrRelationships.get(r).getSource().getFigId());
15  nf.setTar(arrRelationships.get(r).getTarget().getFigId());
16                      nf.calculateMapping();
17                      nav.add((NavigationFlow) nf.getMapping());
18                  }
19              }
20              r++;
21          }
```

En el *Listado 19* se muestra fragmentos de código de la clase *FormR* para instanciar un *DataFlow*, con un comportamiento semejante al caso anterior.

Listado 19. Fragmento de código de la clase FormR para instanciar un DataFlow

```
1          mnuDataFlow.setOnAction(new EventHandler<ActionEvent>() {
2              @Override
3              public void handle(ActionEvent event) {
4                  lineType = CustomLine.DASHED_LINE;
5                  lineEndType = CustomLine.DIRECTION_END;
6
7              }
8          });
9   if (!arrRelationships.isEmpty()) {
10             for (CustomLine e : arrRelationships) {
11                 if (e.getLine_type() == 2) {
12                     DataFlowR nf = new DataFlowR();
13 nf.setSrc(arrRelationships.get(0).getSource().getFigId());
14 nf.setTar(arrRelationships.get(0).getTarget().getFigId());
15                     nf.calculateMapping();
16                     arrDataFlow.add((DataFlow) nf.getMapping());
17
18                 }
19
20             }
21         }
```

En el *Listado 20* se muestran fragmentos de código de la clase *CustomLine*, en las líneas 1 - 6 se definen constantes para facilitar la lectura del código; en la línea 8 se encuentra un fragmento del método que permite mostrar una línea en el editor gráfico. Dependiendo del tipo de figura destino, se hacen algunos ajustes para la cabeza de flecha, si existe, pues debido a que es permitido mover las figuras en el editor, las líneas realmente se conectan a los centros de las figuras y no a las orillas.

Listado 20 Fragmento de código de la clase CustomLine

```
1      public static final int SOLID_LINE = 1;
2      public static final int DASHED_LINE = 2;
3      public static final int ASSOCIATION = 5;
4      public static final int GENERALIZATION = 6;
5      public static final int AGGREGATION = 7;
6      public static final int COMPOSITION = 8;
```

```
7
8  public void create(VisualEditor ve, Figure s, Figure t,
9              int lineType, int endType) {
10
11         if (lineType == SOLID_LINE) {
12             this.putEnding();
13             line.setStrokeWidth(2);
14             line.setStrokeLineJoin(StrokeLineJoin.BEVEL);
15
16             if (source instanceof CustomCircle) {
17                 if (source instanceof SubmitEventR || source
   instanceof SelectEventR
18                         || source instanceof EventR) {
19                     this.linkCircleSource();
20                 }
21             }
22             if (target instanceof CustomRectangle) {
23                 this.linkRectangleTarget();
24             }
25             this.viewEditor.addLine(linePlusEnding);
26             this.source.addRelationship(this);
27             this.target.addRelationship(this);
28         }
29         if (lineType == DASHED_LINE) {
30             this.putEnding();
31             line.getStrokeDashArray().setAll(1.0, 4.0);
32             if (source instanceof CustomRectangle) {
33                 if (source instanceof FormR || source instanceof
   ListR) {
34                     this.linkRectangleSource();
35                 }
36             }
37             if (target instanceof CustomRectangle) {
38                 this.linkRectangleTarget();
39             }
40             this.viewEditor.addLine(linePlusEnding);
41             this.source.addRelationship(this);
42             this.target.addRelationship(this);
43         }
```

3.7.4 Despliegue

En la *Figura 30* se muestra la representación gráfica de los elementos *DataFlow* y *NavigationFlow* en el ambiente de modelado de ITO's_iBRAIN, el tipo de relación se selecciona por medio del menú contextual de la figura seleccionada como origen.

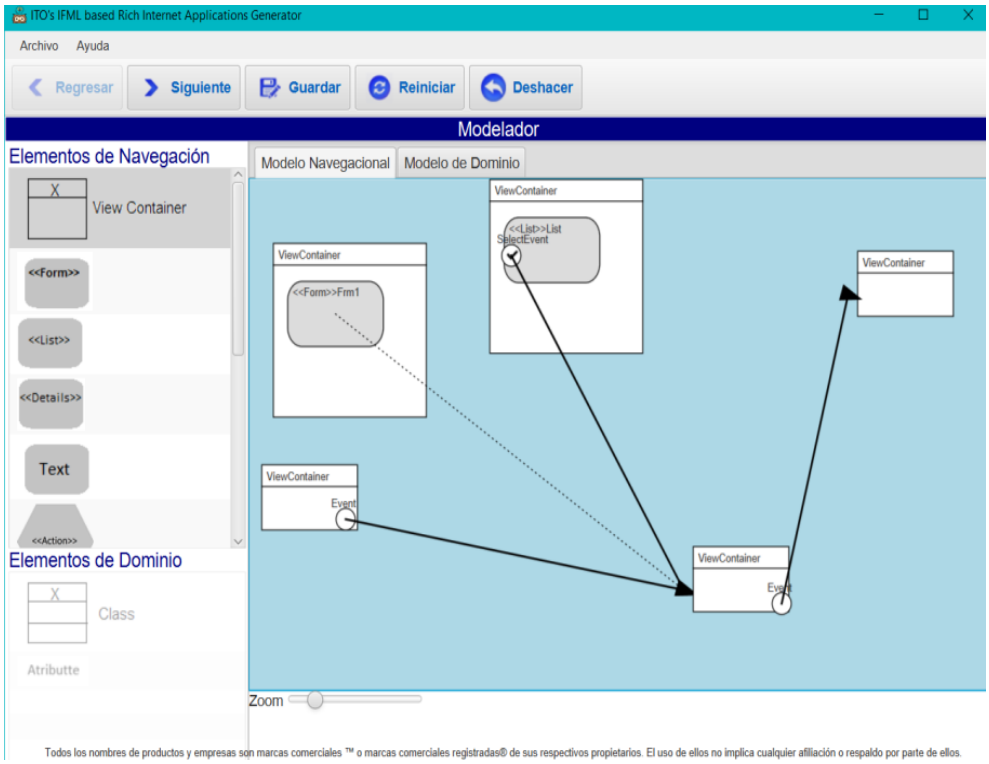

Figura 30. Representación gráfica de los elementos DataFlow y NavigationFlow

3.8 Incremento VII Área de modelado para el diagrama de dominio en ITO´s_iBRAIN

En incrementos anteriores se describió el desarrollo del modelo navegacional de IFML, en este incremento se describe el desarrollo del modelo de dominio; ya que este modelo es más pequeño sólo abarca solo un incremento.

3.8.1 Análisis de requerimientos

El modelo de domino tiene como elemento central las clases que representan conceptos del negocio; cada clase tiene un menú contextual donde se muestran los diferentes tipos de relaciones que permiten: asociación, composición, agregación y herencia. Cada clase acepta atributos, mismos que también manejan un menú contextual que tiene las opciones para eliminar, cambiar nombre y ajustar, en este último se muestra una ventana modal para elegir el tipo de dato y la visibilidad del atributo (por omisión son cadenas con visibilidad privada). Dadas las características de generación de ITO's_iBRAIN, se decidió no modelar métodos en las clases.

3.8.2 Modelado

En la *Figura 31* se muestra el diagrama de clases correspondiente a los elementos del modelo de dominio. `ClassR` representa las clases y `OwnedAttributeR` los atributos; ambas heredan de `CustomRectangle`; las relaciones son clases que heredan de `CustomLine`; como IFML no incluye una terminología particular para ellas, se decidió mantener el nombre de su equivalente XMI.

3.8.3 Construcción

Los elementos seleccionados en la *Figura 32* son los correspondientes para la representación gráfica de los elementos del modelo de domino.

En el incremento IV se describió el funcionamiento de la clase `CustomRectangle`; en el *Listado 12* del incremento IV se muestran fragmentos de código de dicha clase que también aplican en este punto, por tal motivo en este incremento no se detalla tal comportamiento. En este incremento sólo se explican fragmentos de código de las clases `ClassR` y `CustomLine`.

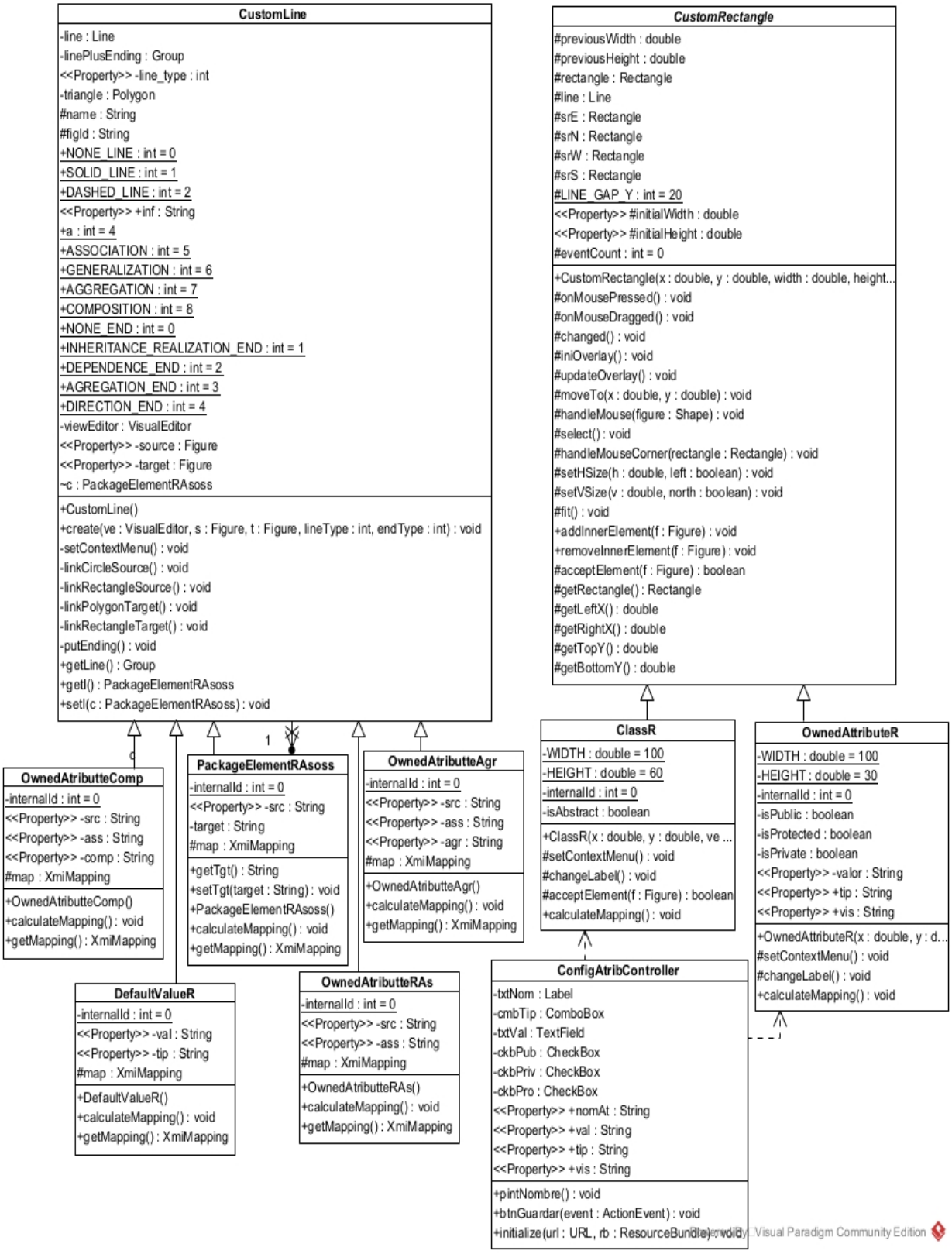

Figura 31. Diagrama de clases para la representación gráfica de los elementos del modelo de dominio

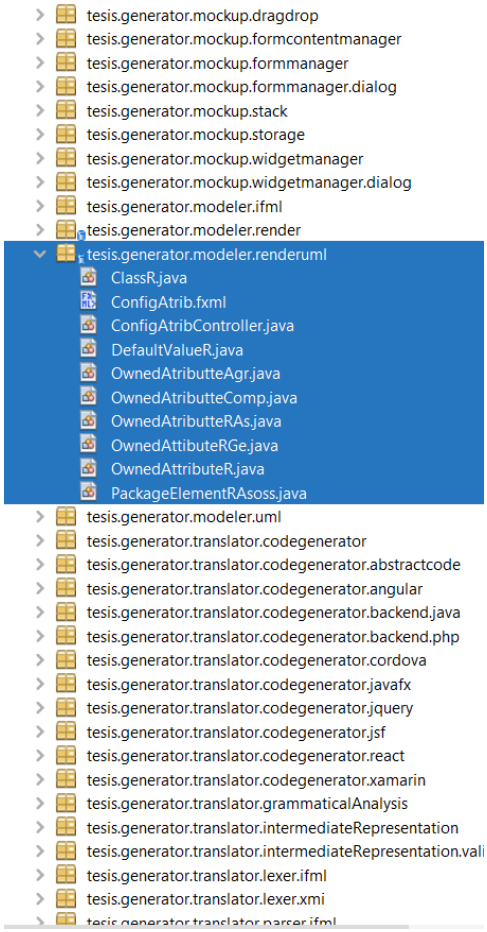

Figura 32. Elementos para la representación gráfica del modelo de dominio

En el *Listado 21* se muestra un fragmento de código de `ClassR`, en la línea 1 está el constructor; en las líneas 13 – 20 se encuentra el comportamiento del menú contextual con las opciones particulares para `ClassR`; en la línea 23 se encuentra el método para cambiar el nombre de la clase tomando en cuenta que el estilo de la letra es cursiva si la clase es abstracta (condición que también se modifica mediante el menú contextual); en la línea 26 se encuentra el método para saber qué elementos anidados acepta `ClassR` y en la línea 29 está el método para relacionar la clase del modelo de dominio con su equivalente XMI.

Listado 21. Fragmento de código de la clase ClassR

```
1    public ClassR(double x, double y, VisualEditor ve) {
2
3        super(x, y, WIDTH, HEIGHT, Color.WHITE, ve,
4                true, true, "Class_" + internalId, false);
5        this.isDomein = true;
6        this.figId = "clss_" + internalId;
```

```
7          this.name = "Class_" + internalId;
8
9          this.expectedInnerPosition = HORIZONTAL_POSITION;
10         internalId++;
11     }
12 @Override
13     protected void setContextMenu() {
14         MenuItem mnuAbstc = new MenuItem("Abstracto"),
15                  mnuAgreg = new MenuItem("Agregacion"),
16                  mnuComp = new MenuItem("Composicion"),
17                  mnuGenz = new MenuItem("Generalizion"),
18                  mnuAso = new MenuItem("Asociacion");
19         super.setContextMenu();
20         contextMenu.getItems().addAll(mnuAbstc, mnuAgreg, mnuComp,
   mnuGenz, mnuAso);
21     }
22 @Override
23     protected void changeLabel() {}
24
25 @Override
26     protected boolean acceptElement(Figure f) {}
27
28 @Override
29     public void calculateMapping() {}
```

En el *Listado 22* se muestran fragmentos de código del método `create` de la clase *CustomLine*, con los cambios referentes al soporte de los distintos tipos de relaciones que admite UML y que se reflejan tanto en el tipo de línea (punteada, continua) y en la cabeza de flecha en la figura destino (cabeza de flecha abierta o cerrada, rombo relleno o sin rellenar); en la línea 1 inicia la configuración de la línea cuando se trata de generalización o herencia.

El mismo proceso es para las líneas de tipo asociación, composición y agregación, lo único que cambia es la figura al final de la línea.

Listado 22 fragmento de código del método create de la clase CustomLine

```
1          if (lineType == GENERALIZATION) {
2              this.putEnding();
3              line.setStrokeWidth(2);
4              line.setStrokeLineJoin(StrokeLineJoin.BEVEL);
5
6              if (source instanceof CustomRectangle) {
```

```
7                   this.linkRectangleSource();
8               }
9
10              if (target instanceof CustomRectangle) {
11                  this.linkRectangleTarget();
12              }
13
14              this.viewEditor.addLine(linePlusEnding);
15
16              this.source.addRelationship(this);
19              linePlusEnding.toBack();
20          }
```

3.8.4 Despliegue

En esta sección se muestran las pruebas realizadas para incremento; en la *Figura 33* se muestran las representaciones gráficas de los elementos del modelo de dominio, se observan 8 clases, relacionadas mediante asociación, herencia, agregación y composición; además, se observa que la clase 9 tiene un atributo con un tipo de dato específico y la clase 1 está marcada como abstracta.

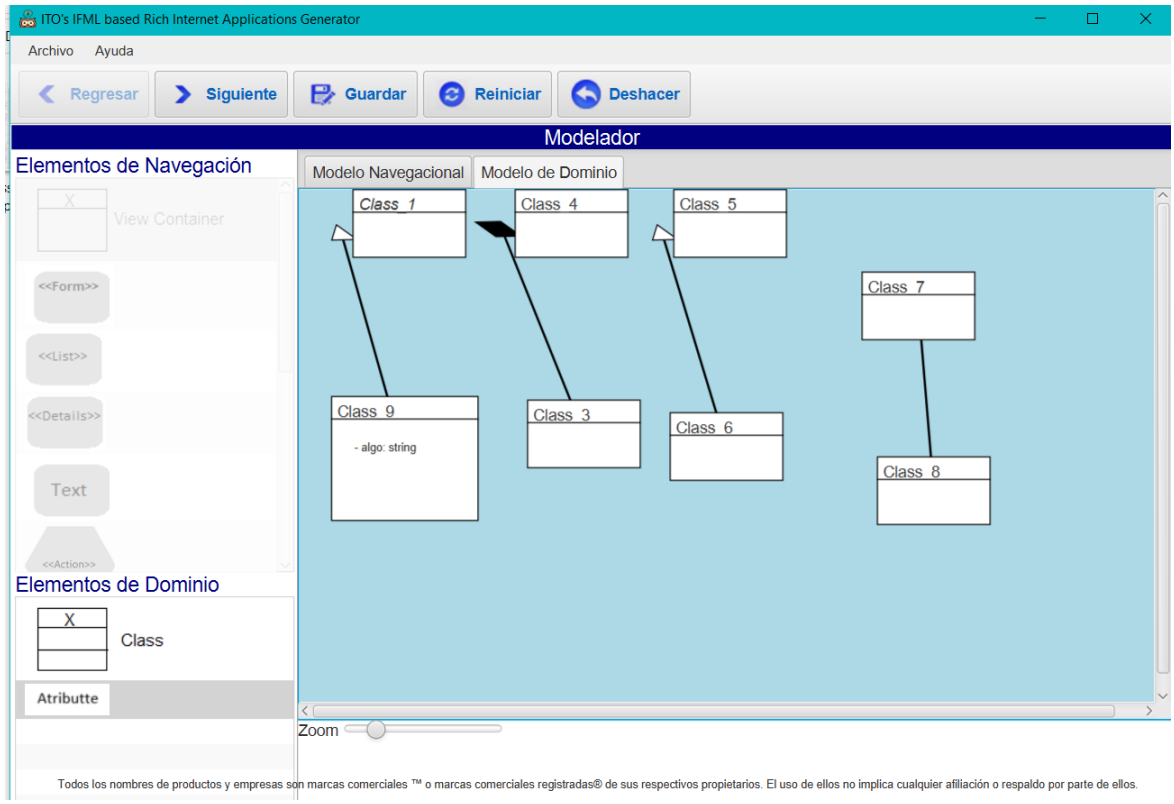

Figura 33. Representación gráfica de los elementos del modelo de dominio

En la *Figura 34* se muestra la ventana modal para seleccionar el tipo de dato del atributo y su visibilidad.

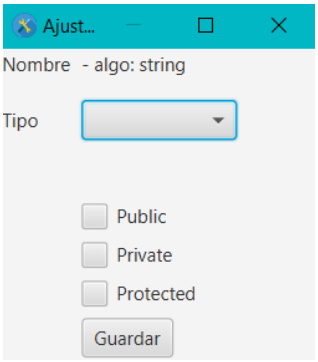

Figura 34. Ventana modal para configurar atributos

3.9 Incremento VIII Generar archivos XMI dinámicamente para el diagrama navegacional (IFML) y dominio (UML)

Este incremento se enfoca en generar los archivos XMI del modelo creado en el editor. En el incremento III se generó un archivo XMI con elementos estáticos. El archivo XMI que se genera en este incremento incluye tanto del modelo de dominio como el navegacional, los dos en un solo archivo.

3.9.1 Análisis de requerimientos

En necesario generar un solo archivo XMI con los elementos en el área de modelado de la herramienta ITO's_iBRAIN, los dos modelos, navegacional y de dominio.

El archivo XMI generado es legible por una herramienta de modelado para UML, por ejemplo *MagicDraw*, al menos en lo que respecta al modelo de dominio. Nuevamente se hace énfasis en que las herramientas de modelado para IFML reportadas en la literatura no tienen la capacidad de importar archivos XMI ni del modelo navegacional ni del modelo de dominio.

3.9.2 Modelado

En la *Figura 35* se muestra el diagrama de paquetes de la herramienta ITO's_iBRAIN, los paquetes marcados de color gris son los correspondientes para el ambiente de modelado, los paquetes `xmimap` y `rendering` contienen las clases relacionadas con la creación de archivos XMI y que se definieron en el Incremento I.

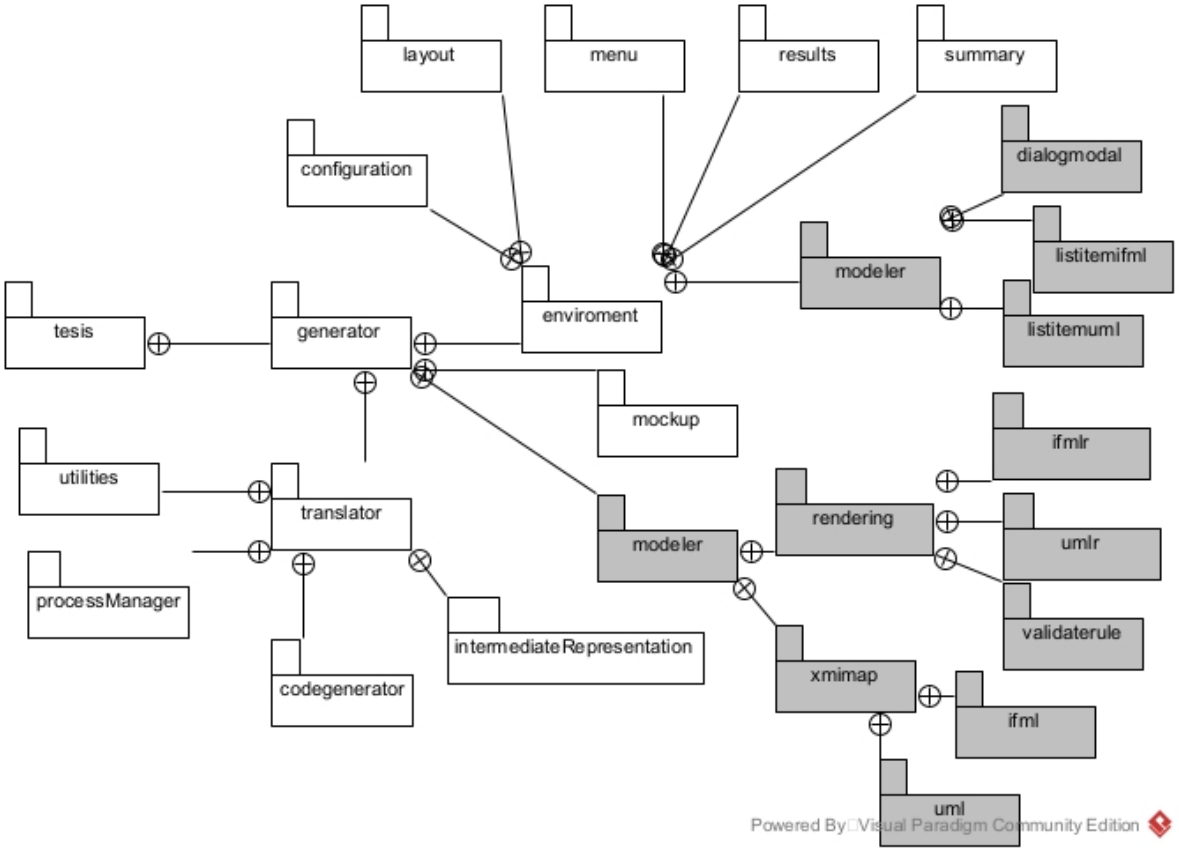

Figura 35. Diagrama de paquetes de la herramienta ITO's_iBRAIN

3.9.3 Construcción

En el *Listado 23* se muestran fragmentos de código del método `save` de la clase `ModelerSceneController`. En las líneas 3 – 12 se encuentran las variables que se necesitan, ya que existen elementos que por omisión existen en el archivo XMI; las líneas 14 – 22 definen las listas de los elementos que siempre existen. En la línea 26 se verifica que las listas de figuras de los modelos de dominio y navegacional que mantiene el editor tengan información para almacenar como XMI, de lo contrario se marca error; posteriormente, en la línea 36 se realiza un ciclo para recorrer los elementos del modelo

navegacional, solicitando a cada uno su método `calculateMapping`, este método lo implementan todos los descendientes de la clase `Figure` para transformar sus propios atributos en los equivalentes de las clases XMI correspondientes. En la línea 45 se encuentra el ciclo correspondiente al modelo de dominio.

En el *Listado 7* del incremento III se explica el fragmento de código para generar el archivo XMI, dicho fragmento no sufrió afectaciones en el presente incremento.

Listado 23. Fragmento de código del método save en la clase ModelerSceneController

```
1   @Override
2       public void save(ActionEvent ae) {
3           XMI xm;
        …….
12          Model mod;
13
14          ArrayList<Ifml_Model> lIfml = new ArrayList<>();
        ……
22          ArrayList<PackagedElement> LPelement = new ArrayList<>();
23          JAXBContext jaxbContext = null;
24          Marshaller jaxbMarshaller = null;
25
26          if (this.arrNavFigures.isEmpty()
27                  && this.arrDomFigures.isEmpty()) {
28              showError("mod.error.emptyDiagrams");
29          } else {
30              xm = new XMI();
31              ifmlmodel = new Ifml_Model();
32              ifmlmodel.setXmi_id("_1I");
33              lIfml.add(ifmlmodel);
34              xm.setIfmlModel(lIfml);
35
36   for (Figure vc : this.arrNavFigures) {
37                  vc.calculateMapping();
38                  if (vc instanceof ViewContainerR) {
39                      lvc.add((ViewContainer) vc.getMapping());
40                  } else {
41                      la.add((Action) vc.getMapping());
42                  }
43  }
44
45              for (Figure vc : this.arrDomFigures) {
46
47                  vc.calculateMapping();
```

```
48                  if (vc instanceof ClassR) {
49                      LPelement.add((PackagedElement)
vc.getMapping());
50
51                  }
52                  if (((PackagedElement)
vc.getMapping()).getPackagedElement() != null) {
53                      LPelement.add(((PackagedElement)
vc.getMapping()).getPackagedElement().get(0));
54                  }
55                  r++;
56              }
```

En el *Listado 24* se muestra, a modo de ejemplo, un fragmento de código del método `calculateMapping` de la clase `ListR`; en las líneas 3 -5 se definen las listas que necesita el elemento `Lista` de XMI; cabe aclarar que, aunque se han mantenido los términos en inglés para los elementos de IFML y UML, al ser *List* una interface propia de Java, fue necesario usar un nombre distinto para la clase XMI equivalente, optándose entonces por el término en español; en las líneas 8 y 9 se colocan los atributos de identificador y nombre de la figura en sus equivalentes XMI; en la línea 14 se verifica si *List* tiene elementos anidados (*DataBinding*, elemento de IFML que permite indicar qué datos del modelo de dominio se mostrarán en el listado representado por el elemento *List* de IFML y *SelectEvent*, elemento de IFML que permite modelar el comportamiento de la aplicación al seleccionar un elemento de la lista), en cuyo caso se recorren los elementos para obtener el mapeo a XMI de cada uno de ellos e incorporarlo al mapeo de *List*. Después, en las líneas 26 – 33 se verifica si existen elementos en la lista de líneas, en cuyo caso de hace el mapeo correspondiente como `DataFlow`.

Listado 24. Fragmento de código de la clase ListR

```
1   @Override
2       public void calculateMapping() {
3           List<SelectionEventt> arrInnerSelect = new ArrayList<>();
4           List<DataFlow> arrDataFlow = new ArrayList<>();
5           List<DataBinding> arrDataBin = new ArrayList<>();
6
7           this.mapping = new Lista();
8           mapping.setId(figId);
9           mapping.setName(name);
10
```

```
11          ((Lista) mapping).setSelectionEvent(new ArrayList<>());
12          ((Lista) mapping).setDataFlow(arrDataFlow);
13          ((Lista) mapping).setDataBinding(arrDataBin);
14          if (!this.innerElements.isEmpty()) {
15              for (Figure ie : this.innerElements) {
16                  ie.calculateMapping();
17                  if (ie instanceof SelectEventR) {
18                      ((Lista)
mapping).getSelectionEvent().add((SelectionEventt)
ie.getMapping());
19                  }
20                  if (ie instanceof DataBindingR) {
21                      ((Lista)
mapping).getDataBinding().add((DataBinding) ie.getMapping());
22                  }
23
24              }
25          }
26          if (!arrRelationships.isEmpty()) {
27              for (CustomLine e : arrRelationships) {
28                  if (e.getLine_type() == 2) {
29                      DataFlowR nf = new DataFlowR();
30
nf.setSrc(arrRelationships.get(0).getSource().getFigId());
31
nf.setTar(arrRelationships.get(0).getTarget().getFigId());
32                      nf.calculateMapping();
33                      arrDataFlow.add((DataFlow) nf.getMapping());
34                  }
35              }
36          }
37
38          if (!arrDataFlow.isEmpty()) {
39              ((Lista) mapping).setDataFlow(arrDataFlow);
40          }
41
42          if (!arrInnerSelect.isEmpty()) {
43              ((Lista) mapping).setSelectionEvent(arrInnerSelect);
44          }
45          if (!arrDataBin.isEmpty()) {
46              ((Lista) mapping).setDataBinding(arrDataBin);
47          }
48      }
49  }
```

3.9.4 Despliegue

En el incremento III se realizó un archivo XMI con elementos estáticos, en este incremento se muestra en la *Figura 36* los mismos elementos del incremento III pero generados directamente en el módulo de modelado.

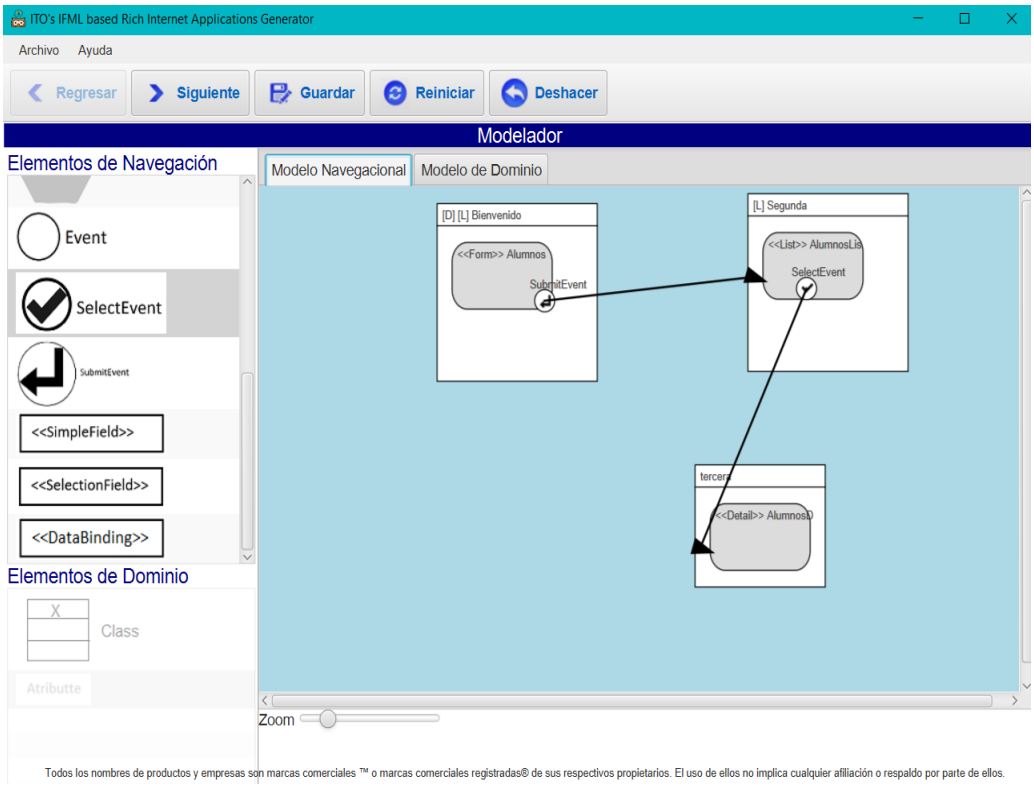

Figura 36. Ambiente de modelado con elementos del modelo navegacional

En la *Figura 37* se muestran los elementos del modelo de domino, que para este caso sólo se agregaron 3 clases con herencia y en la clase Persona se agregó un atributo. Los elementos gráficos se crearon mediante el módulo de modelado.

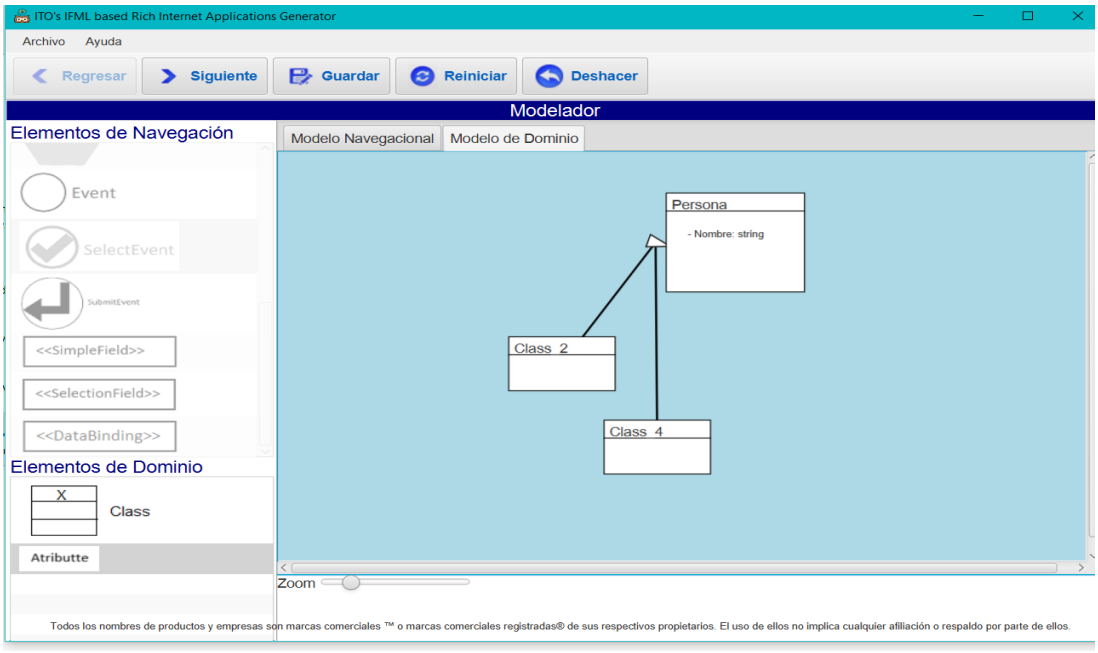

Figura 37. Ambiente de modelado con elementos del modelo de dominio

En la *Figura 38* se muestra el archivo XMI que corresponde a los elementos del modelo navegacional y de dominio, los dos modelos en un solo archivo.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmi:XMI xmlns:ifml="https://www.omg.org/spec/IFML/20140301" xmlns:mofext="http://www.omg.org/spec/MOF/20131001"
         xmlns:uml="http://www.omg.org/spec/UML/20131001"
         xmlns:xmi="http://www.omg.org/spec/XMI/20131001">
    <uml:Model xmi:type="uml:Model">
        <packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_1" name="Persona">
            <ownedAttribute type="string_id" xmi:type="uml:Property" visibility="private" xmi:id="atr_2" name="Nombre"/>
        </packagedElement>
        <packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_2" name="Class_2">
            <generalization general="clss_1" xmi:type="uml:Generalization" xmi:id="gnz_1" name="generalization_1"/>
        </packagedElement>
        <packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_4" name="Class_4">
            <generalization general="clss_1" xmi:type="uml:Generalization" xmi:id="gnz_2" name="generalization_2"/>
        </packagedElement>
        <packagedElement xmi:type="uml:DataType" visibility="public" xmi:id="string_id" name="string"/>
        <packagedElement xmi:type="uml:DataType" visibility="public" xmi:id="integer_id" name="integer"/>
        <packagedElement xmi:type="uml:DataType" visibility="public" xmi:id="double_id" name="double"/>
        <packagedElement xmi:type="uml:DataType" visibility="public" xmi:id="char_id" name="char"/>
        <packagedElement xmi:type="uml:DataType" visibility="public" xmi:id="float_id" name="float"/>
        <packagedElement xmi:type="uml:DataType" visibility="public" xmi:id="boolean_id" name="boolean"/>
    </uml:Model>
    <ifml:Model xmi:id="_1I">
        <ViewPoint name="areaAdministrador" xmi:id="_2"/>
        <Context xmi:id="_3" xmi:idref="_2">
            <UserRole name="Administrador" xmi:id="_5" xmi:idref="_4"/>
        </Context>
        <InteractionFlowModel name="ModelAdm" xmi:id="_4">
            <ViewContainer IsDefault="false" IsLandmark="true" IsXOR="false" xmi:id="vc_1" name="Segunda">
                <List xmi:id="lst_1" name="AlumnosLis">
                    <SelectionEvent xmi:id="Selc_1" name="Select">
                        <NavigationFlow src="Selc_1" target="dtl_1" xmi:id="nvf_1" name="NavegationFlow_1"/>
                    </SelectionEvent>
                </List>
            </ViewContainer>
            <ViewContainer IsDefault="true" IsLandmark="true" IsXOR="false" xmi:id="vc_2" name="Bienvenido">
                <Form xmi:id="frm_1" name="Alumnos">
                    <SubmitEvent xmi:id="sub_1" name="Submit">
                        <NavigationFlow src="sub_1" target="lst_1" xmi:id="nvf_2" name="NavegationFlow_2"/>
                    </SubmitEvent>
                </Form>
            </ViewContainer>
            <ViewContainer IsDefault="false" IsLandmark="false" IsXOR="false" xmi:id="vc_3" name="tercera">
                <Detail xmi:id="dtl_1" name="AlumnosD"/>
            </ViewContainer>
        </InteractionFlowModel>
    </ifml:Model>
</xmi:XMI>
```

Figura 38. Archivo XMI con los elementos del modelo de navegacional y de dominio

En la *Figura 39* muestra la correcta importación del archivo XMI de la figura anterior con los elementos del modelo de dominio. Los mismos elementos que se muestran en la *Figura 37* se muestran en la *Figura 39.*

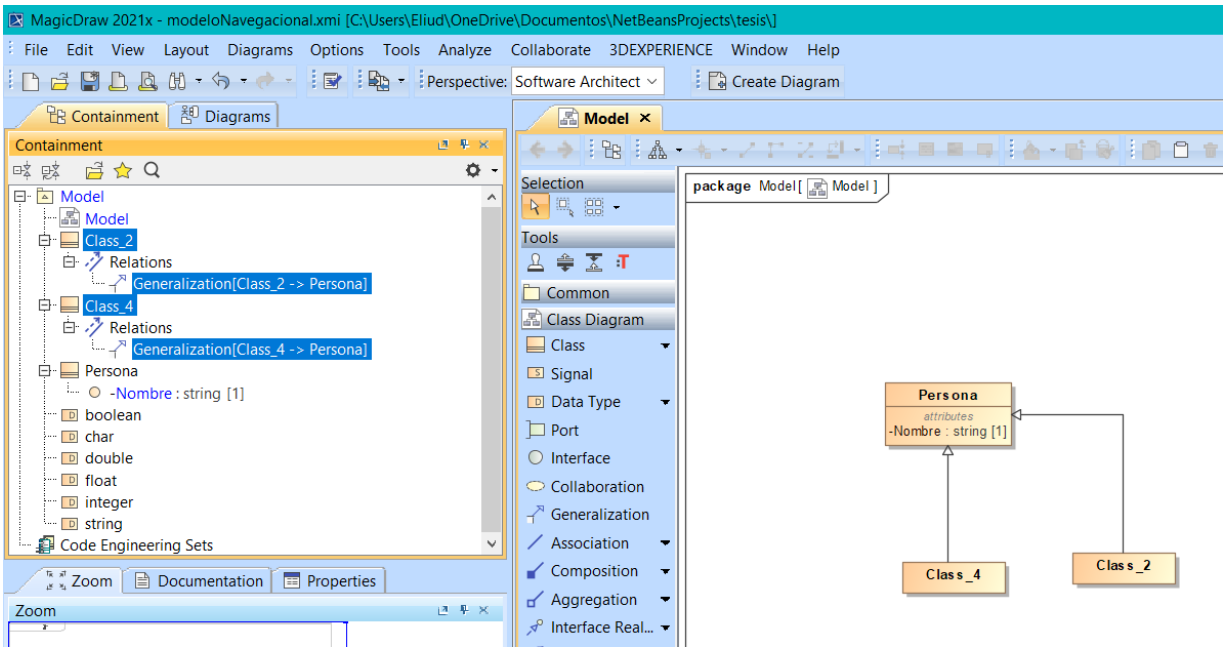

Figura 39. Importación de elementos del diagrama de dominio a herramienta MagicDraw

Capítulo 4 Resultados

En este capítulo se muestran los resultados obtenidos para el módulo de modelado integrado a la herramienta ITO's_iBRAIN, mediante un caso de estudio que incluye los elementos gráficos, las restricciones en los elementos, configuraciones de los elementos, obtención del archivo XMI que incluye ambos modelos y la generación de código funcional.

4.1 Caso de estudio

El caso de estudio que se presenta en este documento de tesis es el mismo que se utilizó en la tesis que dio origen a ITO's_iBRAIN, ya que cuenta con todas las condiciones esperadas: distintos elementos de IFML con características que obligan a su configuración mediante las ventanas modales construidas en el generador, distintas clases de dominio con atributos que necesitan distintos tipos de datos y validaciones, entre otras condiciones que se describen en el siguiente punto.

4.2 Planteamiento caso de estudio *FastRent*

"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" [10].

4.2.1 Modelo navegacional del caso de estudio

Con base en el planteamiento del caso de estudio se reunieron los requisitos necesarios y en la *Figura 40* se muestra el modelo navegacional del caso de estudio, el modelo cuenta con 11 contenedores (*ViewContainer*), 6 formularios (*Form*), 2 textos fijos (*Text*), 1 lista (*List*), 1 informe detallado (*Details*), 5 eventos generales (*Event*, que en este caso implican

navegación entre pantallas), 2 eventos de envío (*SubmitEvent*, sólo posible en formularios), 1 evento de selección (*SelectEvent*, sólo posible en listas), 6 flujos de navegación (*NavigationFlow*), 6 campos sencillos (*SimpleField*) y 1 campo de selección (*SelectionField*). El modelo equivalente, generado en WebRatio, se encuentra en la tesis original [10].

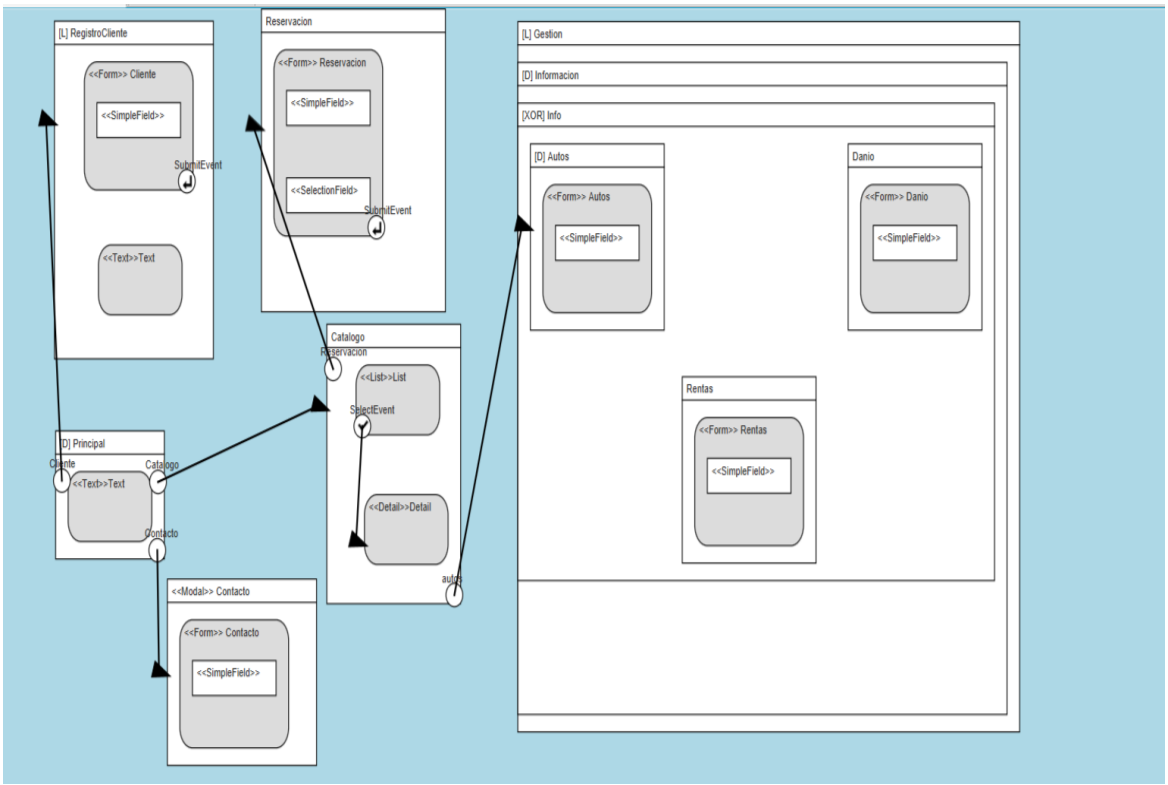

Figura 40. Modelo navegacional del caso de estudio

En el modelo navegacional la página principal es la que está marcada con el estereotipo *Default* ([D]) y tiene el nombre de "*Principal*", contiene un elemento *Text* y 3 elementos *Event* que se conectan con sus respectivos elementos *ViewContainer* mediante elementos *NavigationFlow*.

Al elemento *ViewContainer* que se llama *Reservación* solo accede mediante el elemento *Catálogo* a través de un evento general; mientras que en el caso del contenedor *Gestión* es posible llegar por el menú (al estar marcado como *Landmark*, [L]) y por un evento en *Catálogo* que se interpreta a código como una liga cuando se trata de aplicaciones enriquecidas de Internet (en el caso de las aplicaciones de escritorio y aplicaciones para dispositivos móviles se utilizan botones estilizados).

Los campos modelados dentro de los formularios y que no son visibles directamente en el diagrama sino en las ventanas modales a través de los menús contextuales son:

Formulario Cliente

- Clave
- Nombre
- Apellido paterno
- Apellido materno
- Correo
- *Password*
- Licencia
- Celular
- RFC
- Domicilio

Formulario Reservación

- Folio
- Fecha Inicio
- Fecha fin
- Hora inicio
- Adelanto
- Automóvil
- Forma de pago
- Hora fin

Formulario Contacto

- Nombre
- Correo electrónico
- Comentario

Formulario Autos

- Id auto
- Placa
- Modelo
- Marca
- Capacidad
- Color
- Costo
- Foto
- Descripción

Formulario Rentas

- Folio
- Fecha inicio
- Fecha inicio real
- Fecha fin
- Fecha fin real
- Hora inicio
- Hora inicio real
- Forma de pago
- Adelanto
- Estado

Formulario *Danio* (Daño)

- Id
- Descripción
- Costo

4.2.2 Configuración de los elementos del modelo navegacional

Para generar el elemento *NavigationFlow* se da clic derecho sobre elemento *Event* para mostrar su menú contextual como se muestra en la *Figura 41*.

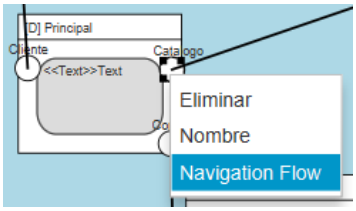

Figura 41. Menú contextual del elemento Event

Para asignar el estereotipo a los elementos *ViewContainer* se abre el menú contextual con clic derecho sobre el elemento como se muestra en la *Figura 42.*

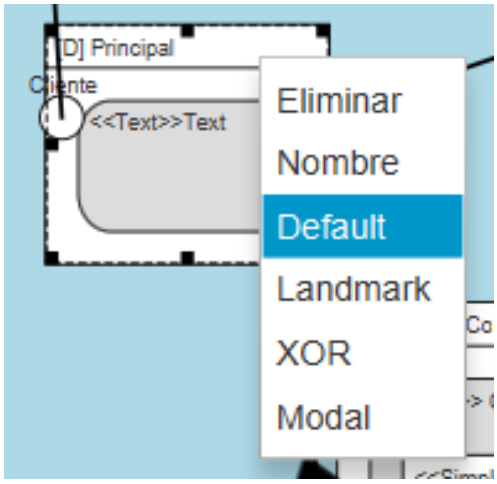

Figura 42. Menú contextual de elemento ViewContainer

Para configurar los elementos *Form* se da clic derecho sobre éste para mostrar el menú contextual como se muestra en la *Figura 43*. Posteriormente se muestra una ventana modal para realizar las respectivas configuraciones de los elementos *SimpleField.*

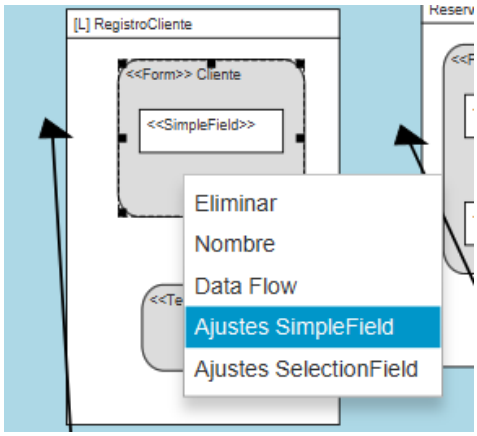

Figura 43. Menú contextual de elemento Form

En la *Figura 44* se muestra la ventana modal para realizar las configuraciones necesarias a los elementos *SimpleField* con las opciones agregar, actualizar o eliminar.

Para actualizar un elemento se da doble clic en un elemento de la tabla, los datos seleccionados se muestran en los campos de Nombre y Tipo; si el campo requiere validaciones, al marcar las cajas del lado derecho de la ventana y actualizar el elemento, éstas se reflejan en la tabla inferior.

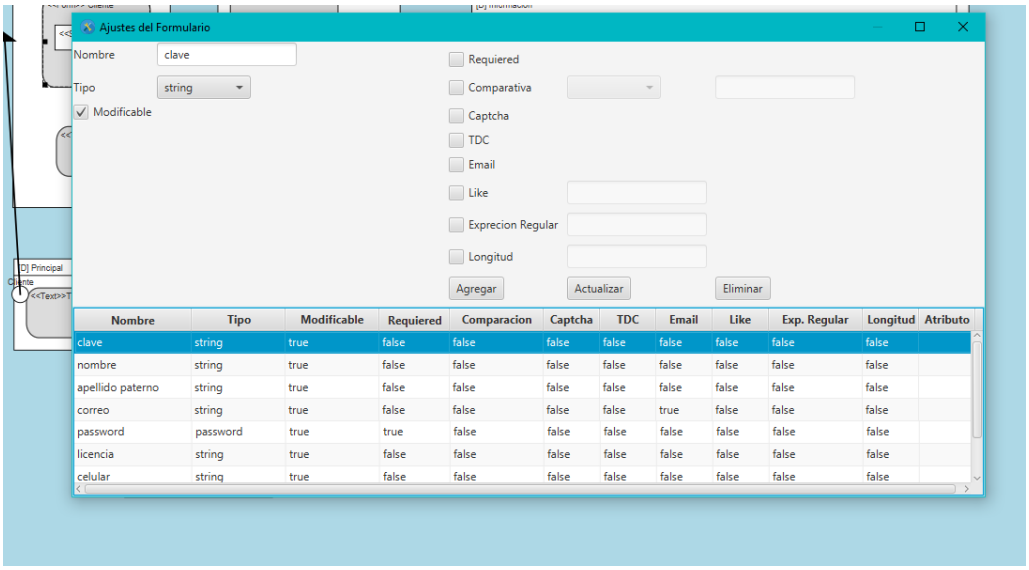

Figura 44. Ventana modal para el formulario de clientes

Al terminar la actualización del elemento, se muestra una ventana emergente (*Figura 45*) que indica al usuario que la operación fue exitosa.

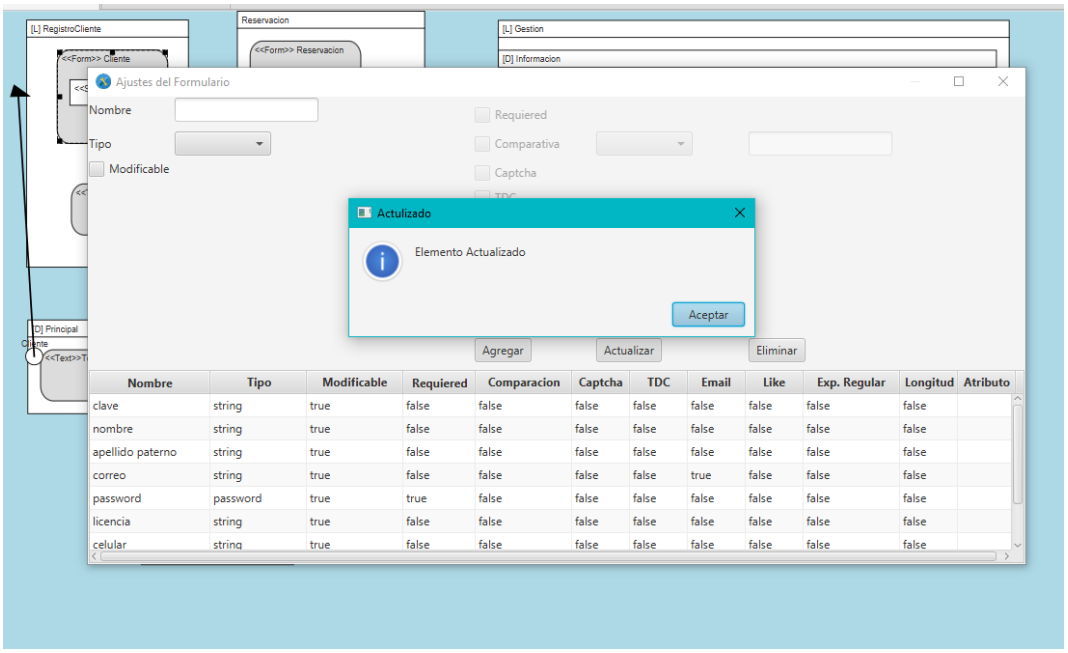

Figura 45. Ventana emergente de elemento actualizado

Una de las reglas de validación que se tiene en el modelador es que si no se ha seleccionado previamente en la tabla algún elemento entonces no es posible actualizar y la herramienta muestra el error de la *Figura 46.*

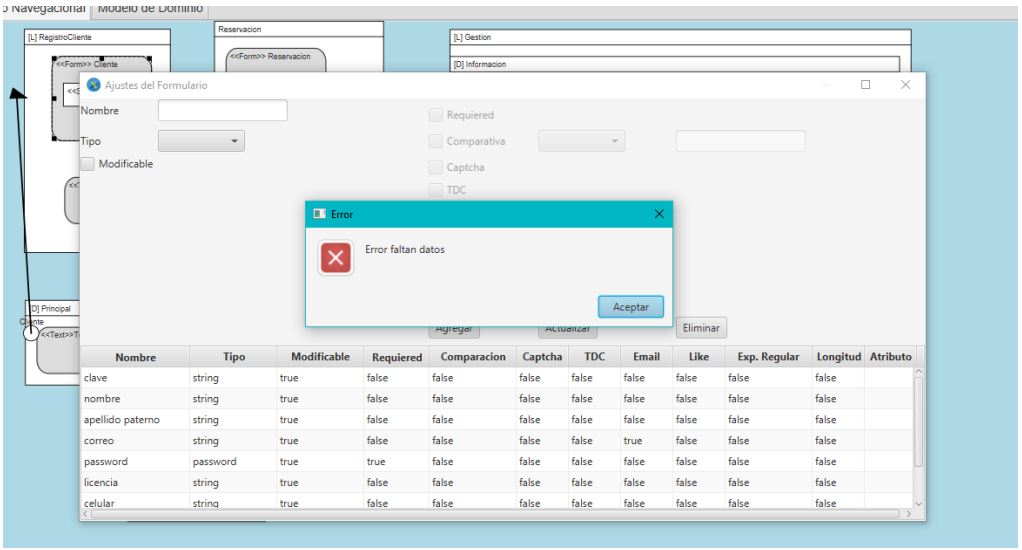

Figura 46. Alerta de error

Al oprimir el botón Agregar, se coloca un nuevo elemento en la tabla, con valores por omisión y se muestra una ventana emergente como en la *Figura 47*.

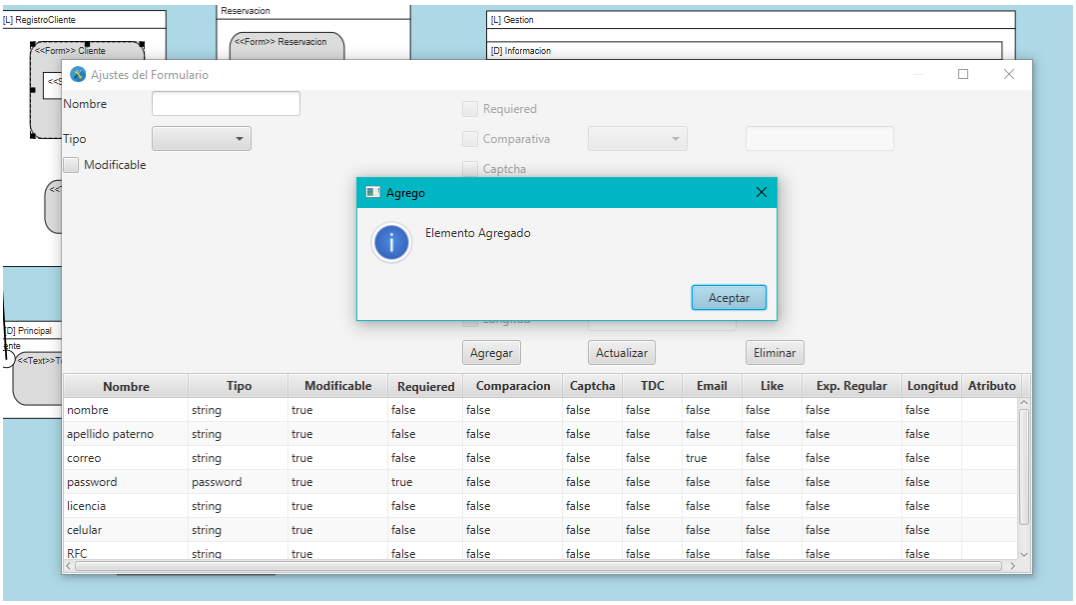

Figura 47. Ventana emergente de elemento agregado

De igual manera, para eliminar un elemento, se selecciona en la tabla y se oprime el botón Eliminar, posteriormente muestra una ventana emergente como en la *Figura 48*.

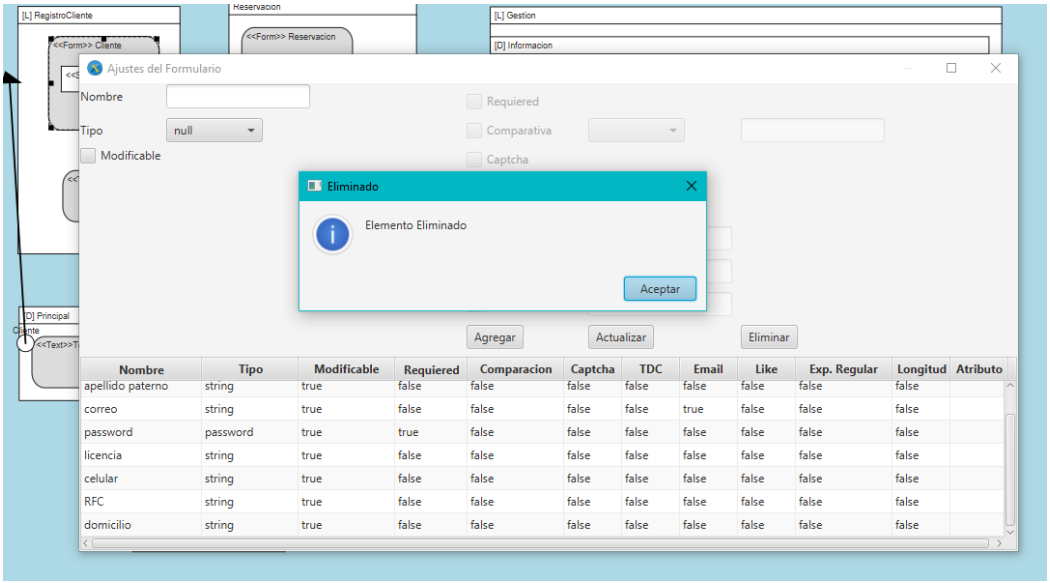

Figura 48. Ventana emergente de elemento eliminado

El procedimiento de configuraciones es el mismo para los elementos *SelectionField* en los formularios.

Los formularios configurados para el caso de estudio fueron los de *Contacto, Cliente, Reservación, Autos, Danio y Rentas.*

4.2.3 Restricciones en los elementos del modelo navegacional.

Con excepción de los elementos *ViewContainer* y *Action*, todos los elementos de IFML requieren colocarse dentro de otro y existen restricciones para tal relación. En la *Figura 49* se muestra el error al tratar de agregar un formulario al editor sin haber seleccionado previamente el contenedor donde quedará colocado.

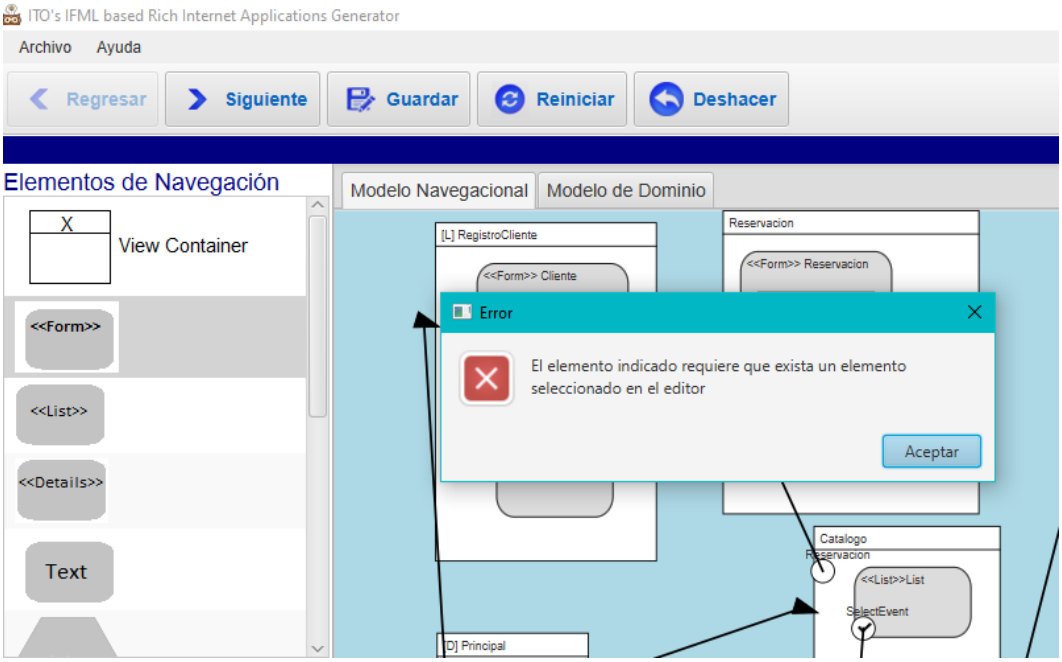

Figura 49. Alerta de error para los elementos que no son ViewContainer o Action

Los *SubmitEvent* solo se colocan en elementos *Form*, los *SelectEvent* por su parte sólo son válidos en elementos *List*. Por ejemplo, si un *SubmitEvent* se intenta colocar en un *ViewContainer* o *List* aparece el error de la *Figura 50*.

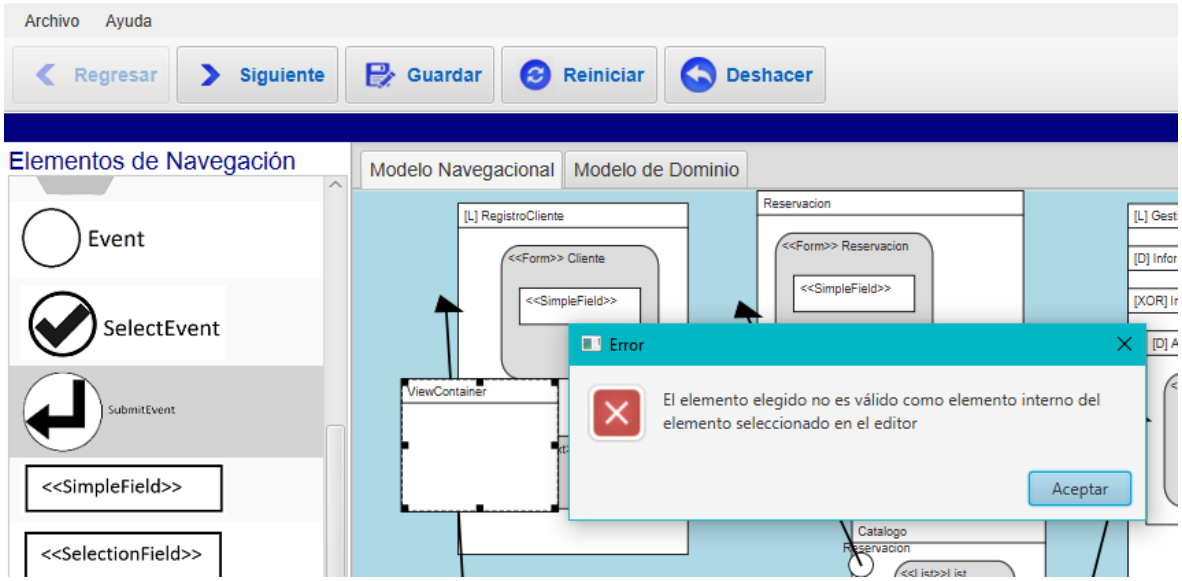

Figura 50. Alerta de error para los elementos SubmitEvent, SelectEvent y Event

4.2.4 Modelo de dominio del caso de estudio

En la *Figura 51* se muestra el modelo de dominio del caso de estudio, contiene cuatro clases: `Cliente` con 11 atributos, `Automóvil` con 9 atributos, `Renta` con 12 atributos y `Danio` con 3 atributos. Las clases se unen con una relación de tipo asociación entre *Cliente – Renta, Danio – Renta* y *Automóvil- Renta*.

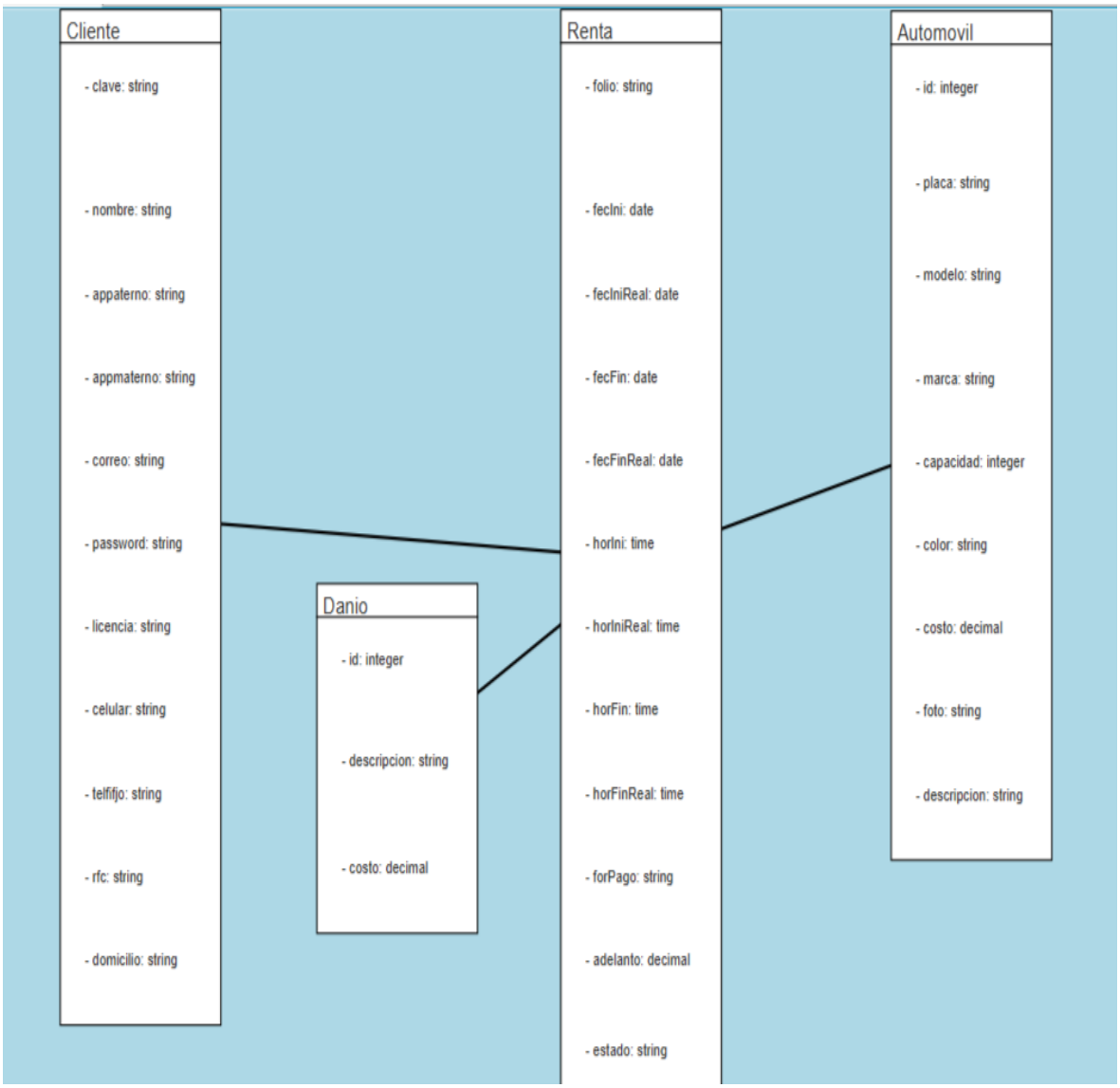

Figura 51. Modelo de domino del caso de estudio

4.2.5 Configuración de los elementos del modelo de dominio

Al dar clic derecho sobre la clase que se desea, se muestra un menú contextual como en la *Figura 52*. En este menú es posible cambiar el nombre de la clase, marcar si es abstracta, eliminar la clase o elegir el tipo de relación que se desea utilizar (al dar clic en una segunda clase se muestra la línea que representa la relación indicada).

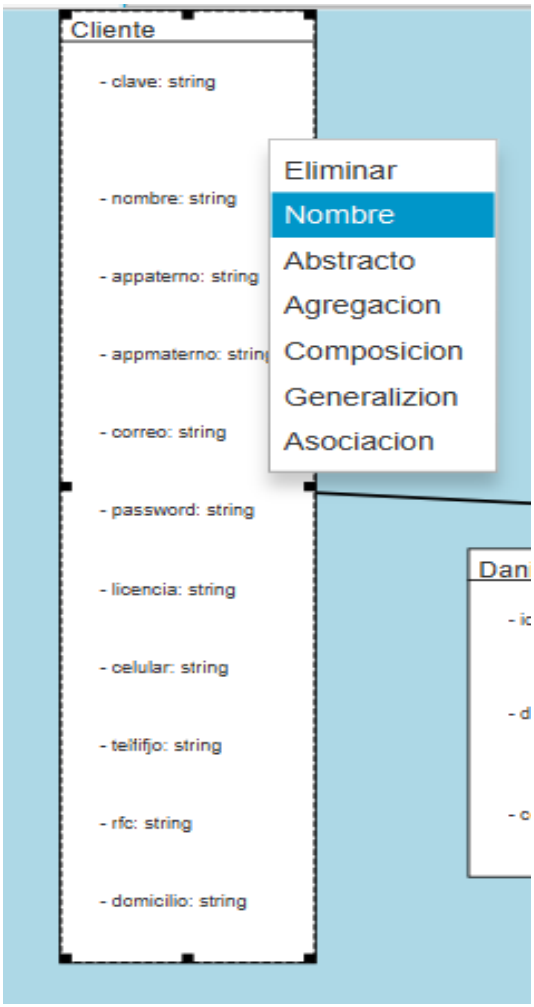

Figura 52. Menú contextual para las clases

Para configurar el nombre de un atributo, se selecciona el atributo y con clic en el botón derecho se abre un menú contextual, se elige la opción Nombre y se muestra una pequeña ventana modal como en la *Figura 53.*

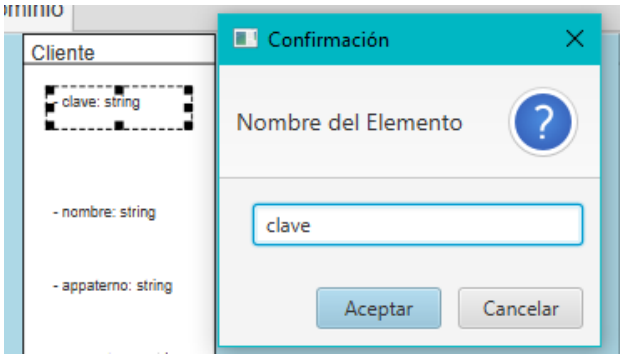

Figura 53. Ventana modal para cambiar nombre del atributo

Para configurar el tipo de dato y visibilidad del atributo, en el menú contextual se selecciona la opción Ajustes y muestra otra pequeña venta modal como en la *Figura 54*.

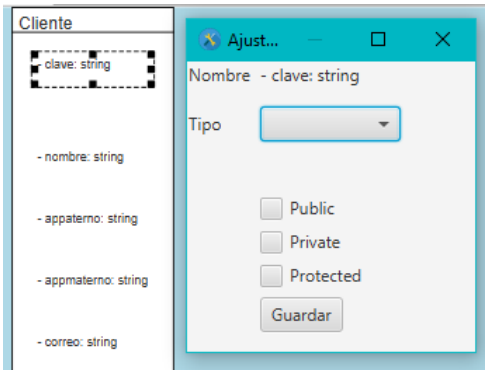

Figura 54. Ventana modal para configurar atributos de una clase en el diagrama de dominio

4.2.6 Restricciones en el modelo de dominio

Las restricciones del modelo de domino son las siguientes:

No es posible colocar una clase dentro de otra; por ejemplo, si se intenta agregar una clase dentro de otra clase previamente seleccionada, se muestra un error como el de la *Figura 55*.

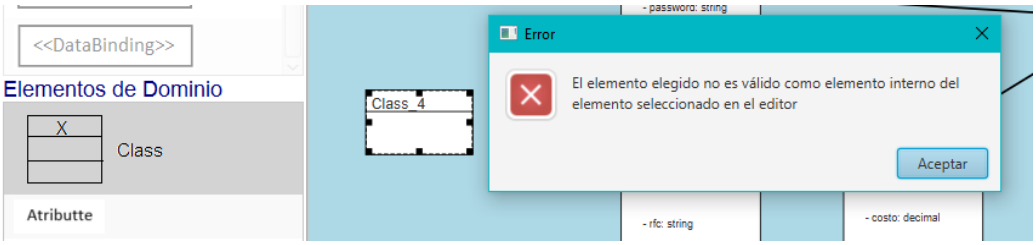

Figura 55. Error al intentar incluir una clase dentro de otra

Tampoco es posible incluir un atributo dentro de otro, es obligatorio que exista una clase seleccionada; el error correspondiente se muestra en la *Figura 56*.

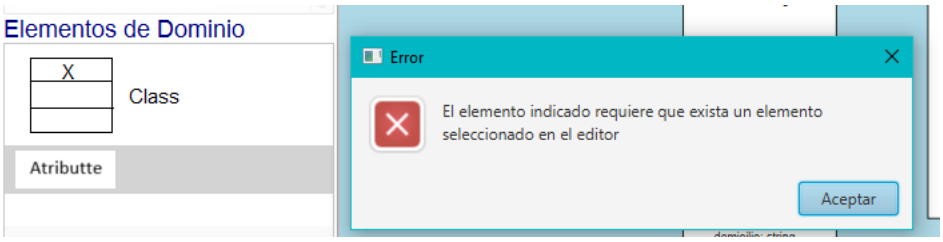

Figura 56. Error al intentar agregar un atributo sin indicar la clase

4.2.7 Archivo XMI resultante

En el *Listado 25* se muestra un fragmento del archivo XMI resultante del modelado descrito en los puntos anteriores, en el mismo archivo están presentes las etiquetas correspondientes para el modelo de dominio y para el modelo navegacional.

En las líneas 6 – 26 se encuentran las etiquetas correspondientes al modelo de domino y de las líneas 27 – 46 se encuentran las etiquetas para el modelo navegacional.

Listado 25. Fragmento del archivo XMI del caso de estudio

1	**<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**
2	<xmi:XMI xmlns:ifml="https://www.omg.org/spec/IFML/20140301"
3	xmlns:mofext="http://www.omg.org/spec/MOF/20131001"
4	xmlns:uml="http://www.omg.org/spec/UML/20131001"
5	xmlns:xmi="http://www.omg.org/spec/XMI/20131001">
6	<uml:Model xmi:type="uml:Model">
7	<packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_0" name="Renta">
8	……
9	<ownedAttribute association="packasso_3" type="clss_2" xmi:type="uml:Property" visibility="private" xmi:id="asso_3"/>
10	</packagedElement>
11	<packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_1" name="Danio">
12	…… .
13	<ownedAttribute type="integer_id" xmi:type="uml:Property" visibility="private" xmi:id="atr_33" name="id"/>
14	</packagedElement>
15	<packagedElement xmi:type="uml:Association" xmi:id="packasso_2" name="packasso_2"/>
16	<packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_2" name="Cliente">
17	<ownedAttribute type="string_id" xmi:type="uml:Property" visibility="private" xmi:id="atr_13" name="clave"/>

18	……
19	</packagedElement>
20	<packagedElement xmi:type="uml:Association" xmi:id="packasso_1" name="packasso_1"/>
21	<packagedElement isAbstract="false" xmi:type="uml:Class" xmi:id="clss_3" name="Automovil">
22	<ownedAttribute type="integer_id" xmi:type="uml:Property" visibility="private" xmi:id="atr_24" name="id"/>
23	……
24	</packagedElement>
25	<packagedElement xmi:type="uml:Association" xmi:id="packasso_3" name="packasso_3"/>
26	</uml:Model>
27	<ifml:Model xmi:id="_1I">
28	<ViewPoint name="areaAdministrador" xmi:id="_2"/>
29	<Context xmi:id="_3" xmi:idref="_2">
30	<UserRole name="Administrador" xmi:id="_5" xmi:idref="_4"/>
31	</Context>
32	<InteractionFlowModel name="ModelAdm" xmi:id="_4">
33	<ViewContainer IsDefault="false" IsLandmark="false" IsXOR="false" xmi:id="vc_1" name="Contacto">
34	<Form xmi:id="frm_3" name="Contacto">
35	<SimpleField modifiable="true" type="string" xmi:id="simf_5" name="nombre"/>
36	<SimpleField modifiable="true" type="string" xmi:id="simf_20" name="correo Electronico">
37	<ValidationRule xmi:id="vr4" name="ValidationRule">
38	<EmailVR errorMessage="Error" xmi:id="eml2" name="EmailVR"/>
39	</ValidationRule>
40	</SimpleField>
41	<SimpleField modifiable="true" type="text" xmi:id="simf_21" name="Comnetarios"/>
42	</Form>
43	</ViewContainer>
44	……
45	</InteractionFlowModel>
46	</ifml:Model>
47	</xmi:XMI>

4.3 Funcionamiento de ITO's_iBRAIN con el módulo de modelado

Después de terminar los dos modelos y almacenarlos como archivo XMI, se da clic en el botón Siguiente en el módulo de modelado, *Figura 57*, con lo que se llega a las pantallas originales de la herramienta.

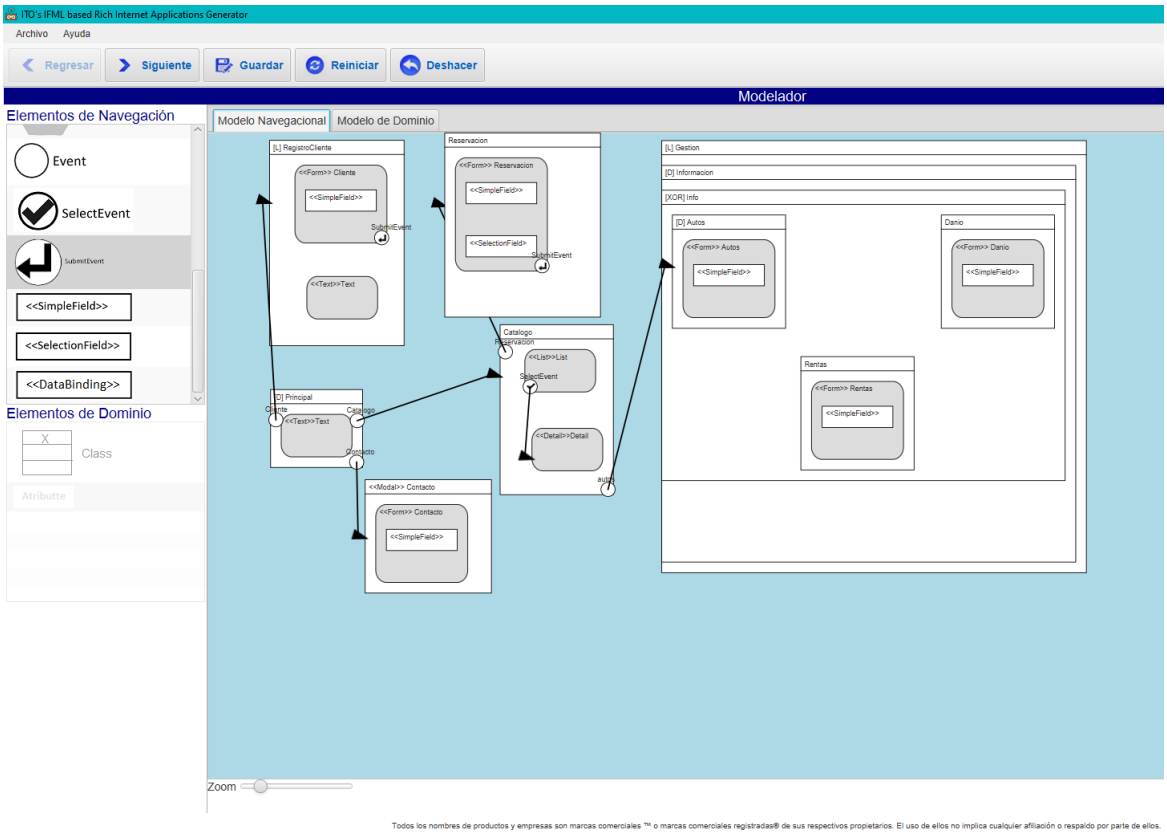

Figura 57 Módulo de modela de la herramienta ITOs_iBRAIN

En la *Figura 58* se muestran las opciones de configuración para generar código, se observa que en esta versión ITO's_iBRAIN ya no se permite ingresar los archivos XMI del modelo navegacional y XML del modelo de dominio, porque ya se tiene el módulo de modelado. Solo solicita ingresar la ruta de salida y seleccionar la ruta de los estilos a utilizar si así lo requieren las tecnologías elegidas.

Para este caso de estudio se seleccionó la plataforma Web, el diseño con la distribución A (cabecera, menú, contenido principal mismo que corresponde al modelo navegacional, distribuidos de forma vertical), la tecnología para *front-end JQuery* y tecnología para *back-end PHP*.

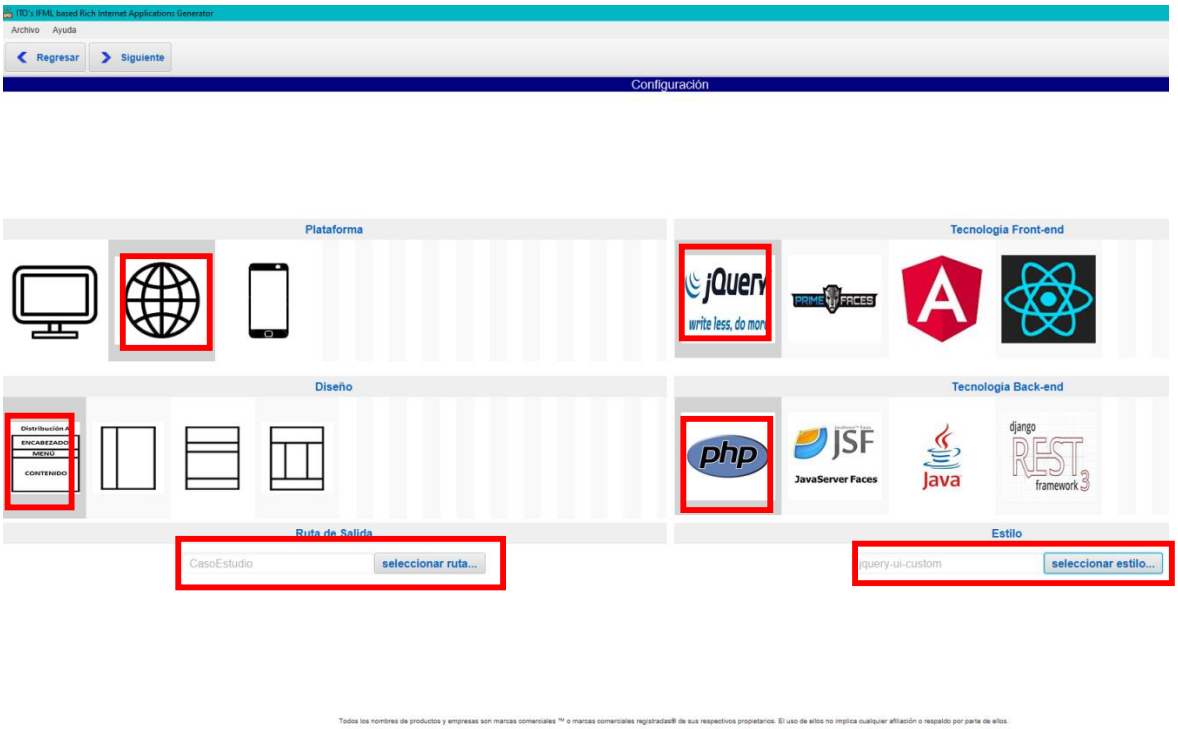

Figura 58. Configuración para generar código del caso de estudio

En la *Figura 59* se muestra la pantalla para el maquetado de los formularios, no se muestra la configuración de esta parte ya que no pertenece a este proyecto de tesis; por lo tanto, la herramienta informa que se utilizará la distribución predeterminada para los campos de cada formulario (vertical, etiqueta del lado izquierdo del campo, un campo por línea).

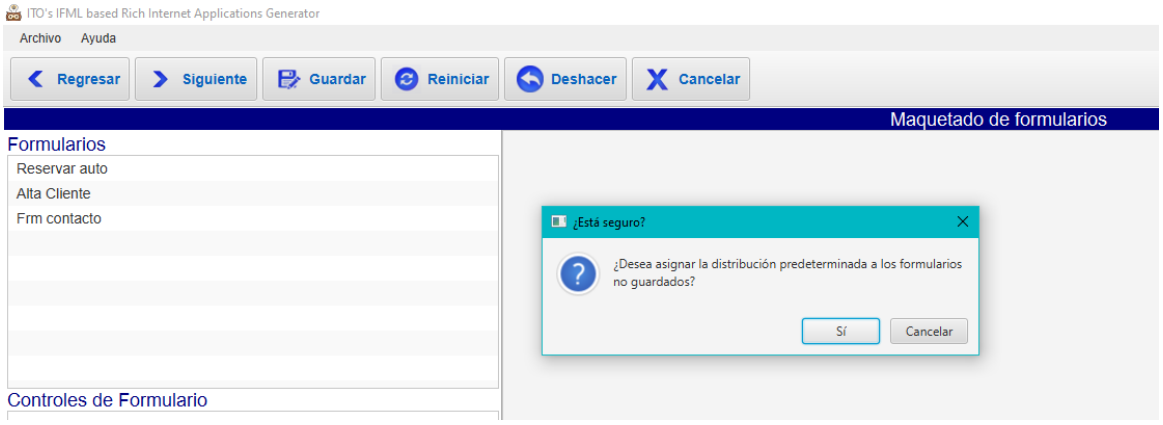

Figura 59. Maquetado de formularios

En la Figura 60 se muestra el resumen de configuración de la herramienta ITO´s_iBRAIN.

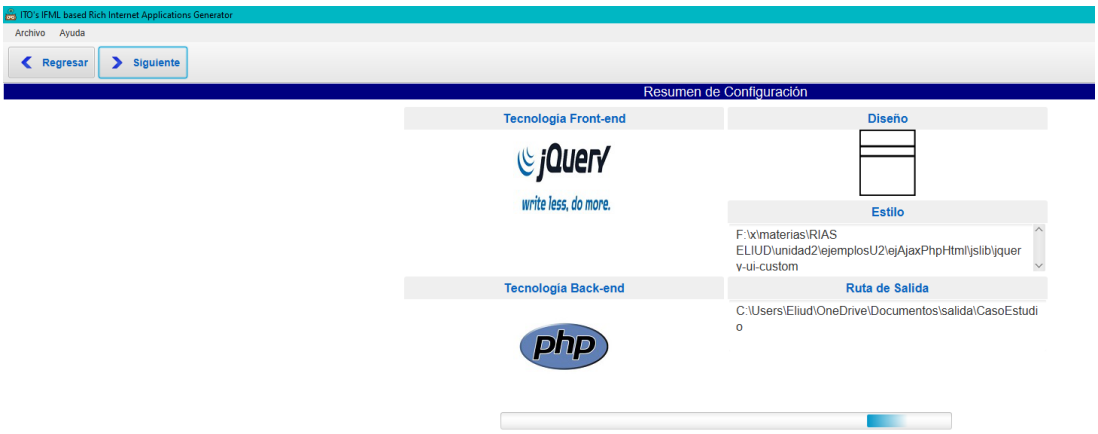

Figura 60. Resumen de configuración de ITO's_iBRAIN

En la *Figura 61* se muestra un listado con todos los archivos generados por la herramienta a partir de los modelos y configuración realizados previamente.

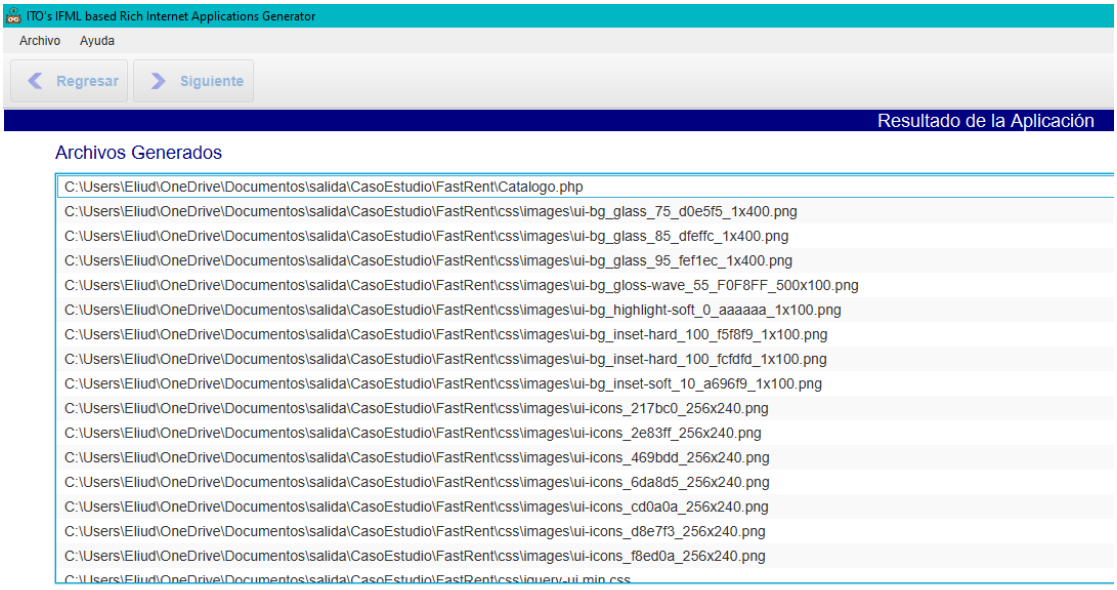

Figura 61. Resumen de Archivos generados

En la *Figura 62* se muestra la carpeta con los archivos generados; posteriormente, dicha carpeta se copia y pega en el servidor de pruebas *XAMPP* puesto que en este caso se eligió como tecnología de *back-end* a PHP.

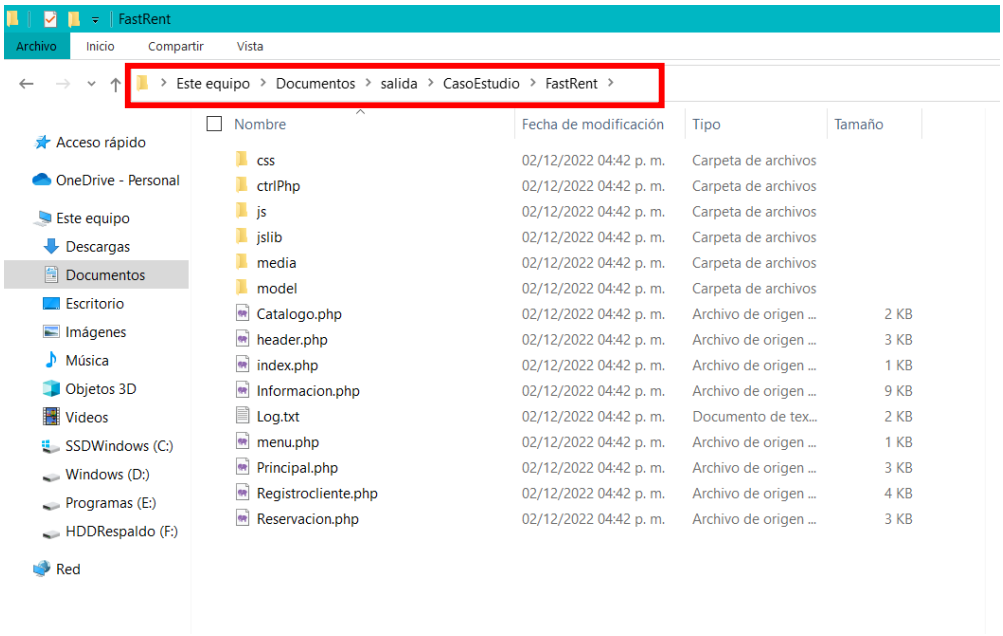

Figura 62. Archivos de la aplicación generada en ITO´s_iBRAIN

En la *Figura 63* se muestra en el *browser* la pantalla principal de la aplicación *FastRent* modelada y generada en ITO's_iBRAIN; debajo del área "*header*" aparece el menú de navegación, con ligas a las páginas equivalentes a los contenedores modelados que incluían la opción [L]. El botón de la parte inferior es la "traducción" del evento y flujo de navegación asociados al contenedor Principal.

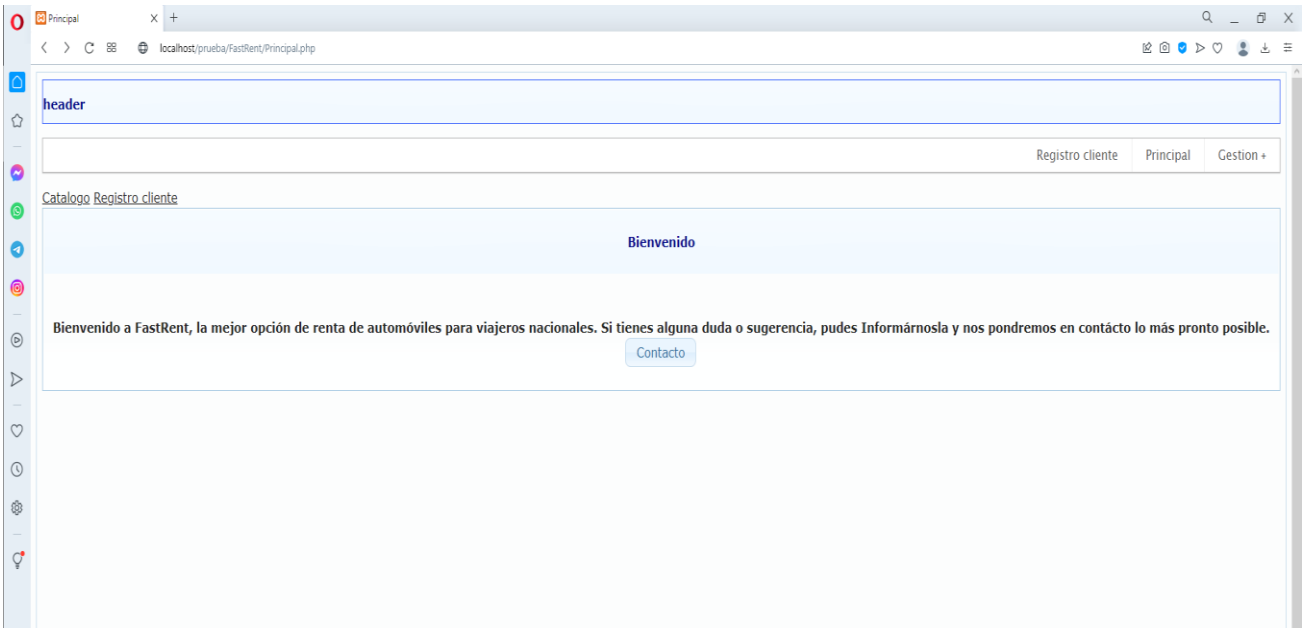

Figura 63. Pantalla principal de la aplicación FastRent

Capítulo 5 Conclusiones y recomendaciones

En el presente capítulo se describen las conclusiones a las que se llegó al término del proyecto de tesis, los objetivos y los trabajos a futuro esperados para el módulo de modelado.

5.1 Conclusiones

Los años que lleva IFML en el mercado aún no se han reflejado en una amplia cantidad de herramientas que soporten su modelado, por lo que las definiciones formales del lenguaje y las tres herramientas publicadas hasta el momento son la única guía disponible para generar un nuevo modelador, esto llevó a una investigación detallada que por momentos parecía recursiva ya que se trataba de modelar un modelador.

En contraparte, la definición formal de XMI para UML, fue más sencilla de revisar ya que al estar el lenguaje sólidamente posicionado en el mercado, existe gran cantidad de herramientas, documentos, validadores de archivos y libros que permiten no sólo crear un modelador sino verificar su correcta funcionalidad con un esfuerzo bastante menor comparado con IFML, si bien hay que acotar el hecho de que no se consideraron todos los diagramas descritos con UML sino solamente el diagrama de clases que hace las veces de diagrama de dominio en IFML.

Por su parte la herramienta ITO's_iBRAIN ha recibido contribuciones tanto de dos tesis de maestría como de varias residencias profesionales de licenciatura, lo que la obliga a contar con una arquitectura sólida a la que es posible incorporar nuevos módulos en tanto se respeten los lineamientos particulares de dicha arquitectura.

La principal contribución del módulo de modelado es que logra que la herramienta ITO's_iBRAIN sea totalmente autónoma, ya que desaparece la dependencia de WebRatio para realizar los modelos; esto permitirá en el mediano plazo que ITO's_iBRAIN sea un apoyo para que los desarrolladores (estudiantes o miembros de pequeñas empresas de desarrollo) apliquen Ingeniería de Software y buenas prácticas sin necesitar una inversión económica alta y sin considerar una pérdida de tiempo el modelado, dado que obtienen un

código funcional para verificar los requerimientos del usuario y que sirve de base para el resto del desarrollo.

El nuevo módulo también se modeló usando herramientas que permitieron generar artefactos de ingeniería de software, como los diagramas de clases y de paquetes, que dejan constancia de las decisiones de diseño que se tomaron a lo largo del desarrollo y que guiaron la construcción a lo largo de los diversos incrementos. En ese sentido, el uso de los patrones de diseño facilitó el desarrollo iterativo al permitir la incorporación de nuevos elementos sin implicar cambios a lo construido previamente.

5.2 Recomendaciones

Actualmente el modelador cumple con su función que es realizar los modelos navegacional y de dominio así como almacenarlos en un archivo XMI; sin embargo, es importante el optimizar el modelador, empezando con la selección de clases y atributos en el elemento *DataBinding*, incluir la configuración de multiplicidad en los atributos de las clases presentes en el modelo de dominio así como en las asociaciones que se representan en ese mismo modelo; por la parte del modelo navegacional, se requiere asegurar que el elemento *SelectionField* de tipo múltiple se relacione específicamente con atributos con multiplicidad mayor a uno, una vez que tal cambio se incluya en el modelo de dominio; también es importante implementar la posibilidad de que el elemento *SelectionField* permita elegir clases como fuente de opciones en lugar de sólo trabajar con valores fijos en *Slots* y, finalmente, corregir la presentación gráfica de los elementos de conexión, ya que actualmente al generar un elementos de conexión se sitúa en un solo punto afectando la visualización del modelado.

Productos académicos

Eliud Federico De los Santos Vera, Beatriz Alejandra Olivares Zepahua, Ulises Juárez Martínez, José Luis Sánchez Cervantes, Celia Romero Torres, Eduardo Lezama Sánchez.

"Propuesta de Arquitectura para el Módulo de Modelado IFML para la Herramienta ITO's_iBRAIN"

5th Conference on Computer Science and Computer Engineering en el marco del Congreso Internacional de Investigación Tijuana que se llevó a cabo del 25 al 28 de abril del 2022, en la ciudad de Tijuana, B. C. México.

Referencias

[1] M. Gómez, J. Cervantes, and P. González, *Fundamentos de Ingeniería de Software*, vol. 51, no. 2. 2019.

[2] S. Estevéz *et al.*, "Análisis Comparativo de Herramientas de Modelado IFML," pp. 141–146.

[3] V. Esterkin and C. Pons, "Evaluación de calidad en el desarrollo de software dirigido por modelos," *Ingeniare*, vol. 25, no. 3, pp. 449–463, 2017, doi: 10.4067/S0718-33052017000300449.

[4] M. Brambilla and P. Fraternali, "Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML," *Interact. Flow Model. Lang. Model. UI Eng. Web Mob. Apps with IFML*, pp. 1–408, 2014, doi: 10.1016/C2013-0-15398-6.

[5] OMG, "INTRODUCTION TO OMG'S UNIFIED MODELING LANGUAGE." https://www.uml.org/what-is-uml.htm (accessed Aug. 18, 2021).

[6] V. Paradigm, "How to Draw Class Diagram?" .

[7] "UML Class and Object Diagrams Overview." https://www.uml-diagrams.org/class-diagrams-overview.html (accessed Aug. 18, 2021).

[8] OMG, "XML." https://www.omg.org/technology/readingroom/XML.htm (accessed Aug. 18, 2021).

[9] OMG, "XMI." https://www.omg.org/spec/XMI/2.5.1/About-XMI/ (accessed Aug. 18, 2021).

[10] S. Estevéz, "DESARROLLO DE UN GENERADOR DE APLICACIONES ENRIQUECIDAS DE INTERNET MODELADAS BAJO EL PATRÓN ARQUITECTÓNICO MVC USANDO UML E IFML," 2019.

[11] Y. Rosales, "Desarrollo del módulo de maquetado para formularios del Generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML."

[12] IFMLEdit.org, "IFMLEdit.org." .

[13] M. Hamdani, W.-H. Butt, M.-W. Anwar, and F. Azam, "A systematic literature review on Interaction Flow Modeling Language (IFML)," *ACM Int. Conf. Proceeding Ser.*, pp. 134–138, 2018, doi: 10.1145/3180374.3181333.

[14] C. Bernaschina, S. Comai, and P. Fraternali, "Online Model Editing, Simulation and Code Generation for Web and Mobile Applications," *Proc. - 2017 IEEE/ACM 9th Int. Work. Model. Softw. Eng. MiSE 2017*, pp. 33–39, 2017, doi: 10.1109/MiSE.2017.1.

[15] F. Ciccozzi *et al.*, "How do we teach modelling and model-driven engineering? A survey," *21st ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion Proceedings, Model. 2018*, pp. 122–129, 2018, doi: 10.1145/3270112.3270129.

[16] P. Pourali and J.-M. Atlee, "An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools," *Proc. - 21st ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Model. 2018*, pp. 224–234, 2018, doi: 10.1145/3239372.3239400.

[17] P. Pourali, “Tooling advances inspired to address observed challenges of developing UML-like models when using modelling tools,” *21st ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion Proceedings, Model. 2018*, pp. 168–173, 2018, doi: 10.1145/3270112.3275340.

[18] R. Queiroz, A.-B. Marques, A. Lopes, E. Oliveira, and T. Conte, “Evaluating usability of IFML models: How usability is perceived and propagated,” *ACM Int. Conf. Proceeding Ser.*, 2018, doi: 10.1145/3274192.3274213.

[19] M. Hamdani, W. H. Butt, M. W. Anwar, I. Ahsan, F. Azam, and M. A. Ahmed, “A Novel Framework to Automatically Generate IFML Models from Plain Text Requirements,” *IEEE Access*, vol. 7, pp. 183489–183513, 2019, doi: 10.1109/ACCESS.2019.2959813.

[20] Oracle, “JavaFX.” https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#BABJBBHJ.

[21] T. Parr, “ANTRL.” https://www.antlr.org/index.html.

[22] Oracle, “JaXB.” https://javaee.github.io/jaxb-v2/.

[23] Oracle, “Group.” https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Group.html.

[24] A. Pérez, “metodologia incremental.” https://www.obsbusiness.school/blog/caracteristicas-y-fases-del-modelo-incremental.

[25] VisualParadigm, “VisualParadigm.” https://www.visual-paradigm.com/.

[26] MagicDraw, “MagicDraw.” https://www.magicdraw.com/main.php.

[27] NIST, “the NIST validador.” http://validator.omg.org/se-interop/tools/validator.

[28] OMG, “MetaModeloIFML.” https://www.omg.org/spec/IFML/20140301/IFML-Metamodel.xmi.