SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Instituto Tecnológico de Orizaba

División de Estudios de Posgrado e Investigación

Maestría en Sistemas Computacionales

TESIS

Desarrollo del módulo de maquetado para formularios del Generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML

PRESENTADO POR:

I.S.C. Yasmin Rosales Cruz M13010225

PARA OBTENER EL GRADO DE:

Maestro en Sistemas Computacionales

DIRECTOR:

M.C.E. Beatriz Alejandra Olivares Zepahua

Orizaba, Veracruz, México. **Diciembre 2019**

Autorización de impresión

Revisión de trabajo escrito

Agradecimientos

Gracias a Dios por siempre estar en los momentos más difíciles conmigo, gracias por bendecirme y darme fuerzas cada vez que las necesité.

Estos dos años y medio estuvieron llenos de nuevos retos pero siempre ha habido una constante en mi vida, esa constante que me da fuerzas y ánimos para siempre luchar por ser la mejor, mis padres. Gracias a ellos por darme tanto amor y siempre decirme que sea la mejor, gracias por enseñarme a luchar por mis sueños y a nunca rendirme, que si bien a veces las cosas no salen bien, siempre estamos los tres juntos luchando contra todos y apoyándonos siempre. Los amo tanto.

Gracias a mis maestros, por guiarme en este camino, darme buenos consejos y tomarse el tiempo fuera del horario de clases para explicarme cosas que a veces no quedaban del todo claras.

Agradezco a mis compañeros, que aunque a veces la carga académica fue pesada, de vez en cuando pasamos buenos ratos y tenemos gratos recuerdos.

Gracias a mis amigos y familiares por estar ahí y ser parte de esto, por echarme porras y creer en mí. Gracias porque cuando necesite siempre estuvieron ahí para ayudarme.

Gracias al TECNM, al Instituto Tecnológico de Orizaba por ser mi alma máter y a la División de Estudios y Posgrados por brindarme las instalaciones para el desarrollo de este proyecto.

Agradezco a CONACYT por brindarme la beca para desarrollar este proyecto y financiar mi Maestría en Sistemas Computacionales.

Y finalmente quiero agradecer a la maestra que guió mis pasos y me convenció de hacer la Maestría, Maestra Betty gracias por darme la oportunidad de contribuir en el desarrollo de este gran proyecto, gracias por ayudarme siempre, a calmar mis nervios y a ver las cosas más claras, porque hubo veces que sentí que era demasiado y usted estuvo ahí para decir las palabras adecuadas. Le estoy muy agradecida que sea mi Director de Tesis, ¡es la mejor!

Índice

Lista de figuras……..……..…..……IX

Lista de tablas…….……XI

Resumen……..……..……..……XII

Abstract…….....XIII

Introducción……...XIV

Capítulo 1 Antecedentes 1

1.1 Marco Teórico 1

1.1.1 Aplicación enriquecida de internet (RIA) 1

1.1.2 Aplicación Web 1

1.1.3 Front end 2

1.1.4 ITOs IFML based Rich Internet Applications Generator (ITOs_IBRAIN) 2

1.1.5 HTML5 2

1.1.6 IFML 2

1.1.7 Interfaz gráfica de usuario 3

1.1.8 Java Server Faces 3

1.1.9 jQuery 3

1.1.10 Maqueta (*Mockup*) 4

1.1.11 Wireframe 4

1.1.12 MVC 4

1.1.13 PHP 5

1.1.14 PrimeFaces 5

1.1.15 Traductor 6

1.2 Planteamiento del problema 7

1.3 Objetivo general y específicos 8
1.3.1 Objetivo general 8
1.3.2 Objetivos específicos 8
1.4 Justificación 9
Capítulo 2 Estado de la práctica 10
2.1 Trabajos relacionados 10
2.2 Análisis comparativo 19
2.3 Propuesta de solución 28
2.3.1 Descripción de la solución 28
2.3.2 Tecnologías a usar en la solución propuesta 30
Capítulo 3 Aplicación de la metodología 32
3.1 Obtención de requerimientos 32
3.1.1 Análisis de las tecnologías de diagramación de *mockups* 32
3.1.2 *Backlog* del módulo de maquetado 37
3.2 Determinación de casos de uso 41
3.3 Aplicación de la metodología 43
3.3.1 *Sprint* 0: Arquitectura del módulo de maquetado 43
3.3.2 *Sprint* 1: Descripción del comportamiento esperado del módulo 45
3.3.3 *Sprint* 2: Desarrollo de los componentes de la arquitectura 46
3.3.3.1 Drag and Drop 47
3.3.3.2 Enviroment 49
3.3.3.3 Form Content Manager 51
3.3.3.4 Form manager 52
3.3.3.5 Widget manager 53
3.3.3.6 Storage 54

3.3.4 *Sprint* 3: Integración del Módulo de maquetado con el Generador de Aplicaciones Enriquecidas 55

3.3.5 *Sprint* 4: Obtención de la RI del Generador de Aplicaciones Enriquecidas 57

3.3.6 *Sprint* 5: Afectación de la RI del Generador de Aplicaciones Enriquecidas 58

3.3.7 *Sprint* 6: Modificación de la generación de código para establecer las distribuciones maquetadas 60

3.3.7.1 Generación de Java ServerFaces con PrimeFaces 61

3.3.7.2 PHP con jQuery 63

3.4 Pruebas de validación de la integración del Módulo al Generador 66

Capítulo 4 Resultados 71

4.1 Caso de estudio FastRent 71

4.1.1 Planteamiento caso de estudio de estudio *FastRent* 71

4.1.2 Modelado en IFML del caso de estudio FastRent en WebRatio Web Platform 71

4.1.2.1 Modelo de Dominio del caso de estudio *FastRent* 72

4.1.2.2 Modelo Navegacional del caso de estudio FastRent 72

4.1.3 Generador de Aplicaciones Enriquecidas en funcionamiento en el caso FastRent 77

4.1.4 Aplicación generada en la combinación de Java ServerFaces y PrimeFaces 82

4.1.4.1 Formularios responsivos en JSF con PF 85

4.1.5 Aplicación generada con la combinación de PHP y jQuery 86

4.1.5.1 Formularios Responsivos en PHP con jQuery 89

4.2 Casos de estudio realizados en las estancias 90

4.2.1 Sistema de Información de la Armada de México (SIAM) 91

Capítulo 5 Conclusiones y recomendaciones 93

5.1 Conclusiones 93

5.2 Recomendaciones 94

Referencias 96

Índice de figuras

Fig. 1.1 Esquema general de un traductor 7
Fig. 2.1 Representación gráfica de la solución 30
Fig. 3.1 Wireframe de la GUI del módulo de maquetado de formularios 38
Fig. 3.2 Diagrama de casos de uso 41
Fig. 3.3 Arquitectura del módulo de maquetado 44
Fig. 3.4 Diagrama de actividades del módulo de maquetado 46
Fig. 3.5 Diagrama de clases del componente drag and drop 48
Fig. 3.6 Representación visual del ToolBar 50
Fig. 3.7 Diagrama de clases del componente environment 50
Fig. 3.8 Diagrama de clases del componente formcontentmanager 52
Fig. 3.9 Diagrama de clases del componente formmanager 53
Fig. 3.10 Diagrama de clases del componente widgetmanager 54
Fig. 3.11 Diagrama de clases del componente storage 55
Fig. 3.12 Diagrama de paquetes general de la integración 56
Fig. 3.13 Modelo en IFML de un formulario asociado a un Action 66
Fig. 3.14 Área de maquetado con controles distribuidos del formulario Persona 67
Fig. 3.15 Distribución del formulario Persona obtenido del Generador 67
Fig. 3.16 Modelo en IFML de un formulario con validaciones asociado a un Action 68
Fig. 3.17 Validaciones de los campos del formulario Validaciones 68
Fig. 3.18 Área de maquetado con controles distribuidos del formulario Validaciones 69
Fig. 3.19 Distribución del formulario Validaciones obtenido del Generador 70
Fig. 4.1 Modelo de dominio de la aplicación FastRent realizado en WebRatio Web Plataform 72
Fig. 4.2 Modelo navegacional en IFML del caso de estudio FastRent 73
Fig. 4.3 Caso de estudio FastRent: Lista de controles para el formulario Reserva auto 74
Fig. 4.4 Caso de estudio FastRent: Slots asigndos a los campos Extras y Forma de pago de Reservar auto 75
Fig. 4.5 Caso de estudio FastRent: Lista de controles para el formulario Alta Cliente 76
Fig. 4.6 Caso de estudio FastRent: Lista de las validaciones asignas a los controles del formulario Alta Cliente 76

Fig. 4.7 Caso de estudio FastRent: Lista de controles para el formulario Frm contacto 77
Fig. 4.8 Caso de estudio FastRent: Lista de las validaciones asignas a los controles del formulario Frm contacto 77
Fig. 4.9 Caso de estudio FastRent: Pantalla de Configuración del Módulo de maquetado 78
Fig. 4.10 Caso de estudio FastRent: Maquetado del formulario Reservar auto 79
Fig. 4.11 Caso de estudio FastRent: Maquetado del formulario Alta Cliente 80
Fig. 4.12 Caso de estudio FastRent: Maquetado del formulario Frm contacto..... 80
Fig. 4.13 Caso de estudio FastRent: Pantalla de Resumen del Módulo de maquetado 81
Fig. 4.14 Caso de estudio FastRent: Pantalla de Resultado del Módulo de maquetado 82
Fig. 4.15 Caso de estudio FastRent en JSF + PF: Aplicación generada..... 82
Fig. 4.16 Caso de estudio FastRent en JSF + PF: Página Principal de la Aplicación ... 83
Fig. 4.17 Caso de estudio FastRent en JSF + PF: Formulario Contacto..... 84
Fig. 4.18 Caso de estudio FastRent en JSF + PF: Formulario Alta cliente 84
Fig. 4.19 Caso de estudio FastRent en JSF + PF: Formulario Reservar auto 85
Fig. 4.20 Caso de estudio FastRent en JSF + PF: Formulario responsivo Alta cliente . 86
Fig. 4.21 Caso de estudio FastRent en PHP + jQuery: Aplicación generada..... 87
Fig. 4.22 Caso de estudio FastRent en PHP + jQuery: Página Principal de la Aplicación 87
Fig. 4.23 Caso de estudio FastRent en PHP + jQuery: Formulario Contacto..... 88
Fig. 4.24 Caso de estudio FastRent en PHP + jQuery: Formulario Alta cliente..... 88
Fig. 4.25 Caso de estudio FastRent en PHP + jQuery: Formulario Reservar auto..... 89
Fig. 4.26 Caso de estudio FastRent en PHP + jQuery: Formulario Alta Cliente Responsivo 90

Índice de tablas

Tabla 2.1 Análisis comparativo de los artículos relacionados 19
Tabla 2.2 Justificación de la elección de tecnologías..... 30
Tabla 3.1 Tabla comparativa de herramientas que permiten el maquetado..... 34
Tabla 3.2 Descripción de los casos de uso 42
Tabla 3.3 Descripción de componentes de la arquitectura..... 44
Tabla 3.4 Atributos y métodos agregados en las clases IR_Form e IR_Widget..... 59
Tabla 3.5 Descripción de las clases abstractas modificadas para la generación de código..... 60

Resumen

La tesis “Desarrollo de un Generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML” da como resultado prototipos de aplicaciones enriquecidas de Internet (RIAs) en los lenguajes *Java Server Faces* con *PrimeFaces* y PHP con jQuery. IFML (*Interaction Flow Modeling Language*, Lenguaje de Modelado de Flujo de Interacción) no contempla consideraciones de diseño gráfico, por lo que los formularios obtenidos manejan una distribución vertical; dicha distribución resulta poco práctica en sistemas de información donde la captura de datos es extensa, ya que da la impresión de que se desperdicia mucho espacio en la pantalla.

La presente investigación pretende complementar el generador original mediante un módulo de maquetado que incluye aspectos de diseño gráfico como son: la distribución de controles dentro un formulario, la edición de las etiquetas asociadas a formularios y la elección de controles específicos para los casos de selecciones simples y múltiples.

El módulo se integra al proceso del generador justo después de crear la representación intermedia (RI), ya que a partir de la información de RI se presenta una interfaz gráfica que permite al usuario maquetar los formularios modelados para organizar la distribución de los controles de tal forma que estén mejor ubicados en el espacio disponible; esto permitirá que el área designada para los formularios tenga una mejor organización visual dentro de la aplicación resultante.

Para el desarrollo del módulo se siguió una adaptación de la metodología Scrum; también se identificaron e incorporaron algunas de las mejores prácticas relacionadas con diseño gráfico y diseño de sistemas mediante un análisis comparativo de herramientas que permitieran el maquetado de aplicaciones de tipo Web, Móviles y Escritorio, y así identificar la forma en que un usuario interactúa con dichas herramientas y tomar los mejores aspectos para incorporarlos y adaptarlos al módulo a desarrollar.

Abstract

From the development of a previous thesis that generated the project: Internet Rich Application Generator (RIAs) Modeled Under the Architectural pattern Model, View, Controller, (MVC) using UML (Unified Modeling Language) and IFML (Interaction Flow Modeling Language), which results are skeletons of RIAs in languages like Java Server Faces with PrimeFaces and PHP with jQuery; It is required to include the Layout Module for the resulting application forms that correspond to what the user has modeled in IFML and UML.

The forms that are obtained from the RIA's generator today are with vertically layout; in large information systems, where the capture of forms is extensive, the resulting code of the generator with respect to the forms is not feasible, all the components of the form are generating them one below the other, giving the impression that it is wasted a lot of space on the screen and there is no way to organize the components to avoid that happening in the generator process because IFML does not have relevant labels to describe the organization of the forms. For that reason the module to be integrated will allow the layout of these forms to improve navigation, the capture of these information systems and in general the graphic aspect of these.

This module will be integrated into the generator process, just after generating the Intermediate Code, since from that step information will be taken to display a graphical interface that will allow the user to layout the forms he has modeled and thus organize the distribution of controls to your liking, so that they are better placed in the application; This will allow the designated space for the forms to have a better visual organization for the users who are going to capture the data.

For the development of the module, an adaptation of the Scrum methodology was followed; Some of the best practices related to graphic design and system design were also identified and incorporated. This through a comparative analysis of tools that allow the layout of Web, Mobile and Desktop applications, and thus identify the way in which a user interacts with these tools and take the best aspects to incorporate them and adapt them to the module to be developed.

Introducción

El diseño visual correctamente empleado es muy importante cuando se desarrollan aplicaciones Web, ya que ayuda a facilitar la interactividad entre la aplicación y el usuario final. A diferencia del diseño de *mockups* (Maquetado) donde las ideas se desarrollan de cero y se empiezan a diseñar con los componentes que se consideran pertinentes, el maquetado que se propone en el desarrollo de este tema de tesis parte de un modelado con diagramas IFML y UML que determina los componentes que conformarán la vista, el modelo y los controladores de la aplicación; tomando en cuenta esto, el módulo a desarrollar será integrado en un generador de aplicaciones RIAs que dentro de este proceso permitirá el despliegue de una interfaz de maquetado donde el usuario será capaz de diseñar y organizar la distribución de los controles de los formularios con los que cuente su aplicación y así maximizar el uso del espacio disponible para facilitar la captura de datos de los sistemas de información, afectando sólo los archivos de salida del generador correspondientes a la vista.

Con la intención de brindar al lector un mejor entendimiento del proyecto de tesis, el presente documento está conformado por tres capítulos.

El capítulo uno llamado Antecedentes, está conformado por el marco teórico, el planteamiento del problema, los objetivos generales y específicos y la justificación del proyecto de tesis. El capítulo dos se titula Estado de la práctica, el cual está conformado por quince resúmenes de artículos científicos que están relacionados con el proyecto que conforma la tesis, así como un análisis comparativo de los artículos, donde se describen las problemáticas, tecnologías usadas, los resultados obtenidos y el estado del trabajo realizado. Y por último el capítulo tres del proyecto de tesis lleva por título Propuesta de solución, el cual contiene una breve explicación del trabajo a desarrollar junto con la descripción de la solución que se le va a dar a la problemática, un listado de nueve distintas tecnologías de información, la descripción de la solución propuesta así como su justificación, el cronograma de actividades para el desarrollo de la tesis y los entregables correspondientes al tercer y cuarto semestre.

Capítulo 1 **Antecedentes**

En este capítulo se presenta el marco teórico del proyecto, donde se describen los conceptos básicos relacionados con el proyecto de tesis.

1.1 Marco Teórico

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

1.1.1 Aplicación enriquecida de internet (RIA)

Macromedia define Aplicación Enriquecida de Internet como la combinación de las mejores características en las interfaces de usuario para aplicaciones de escritorio junto con las mejores características de las aplicaciones Web, las cuales cuentan con una comunicación multimedia interactiva (incorporación de audio y video interactivo bidireccional) para obtener aplicaciones que proporcionan una experiencia de usuario intuitiva, responsiva y efectiva [1].

1.1.2 Aplicación Web

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

La distinción que existe entre un sitio Web y una aplicación Web recae en la capacidad que tiene un usuario para interactuar con en el servidor, si no existe una lógica de negocio en un servidor, el sistema no es una aplicación Web. Para la mayoría de las aplicaciones Web, excepto las más sencillas, el usuario debe introducir más información que la simple solicitud de navegación, esto normalmente se realiza por medio de campos de entrada de algún tipo en formularios [2].

1.1.3 Front end

Es la parte de una aplicación Web donde los usuarios interactúan directamente y se refiere a todas aquellas tecnologías que son visibles para los clientes y que se ejecutan del lado del visualizador Web, centralizándose más que nada en tres lenguajes: 1) HTML, 2) CSS y 3) JavaScript. Los objetivos a los que se enfoca el *Front-End* son: que el usuario tenga una buena experiencia, inmersión y usabilidad [3].

1.1.4 ITOs IFML based Rich Internet Applications Generator (ITOs_IBRAIN)

Es un generador de Aplicaciones Enriquecidas de Internet desarrollado por Estévez [4] que obtiene esqueletos de RIAs en *Java Server Faces*, estándar de Oracle para el desarrollo de interfaces de usuario del lado del servidor, enriquecidas con *PrimeFaces* o PHP, ampliamente utilizado en el mercado, junto con jQuery. Dichas aplicaciones resultantes del uso del generador cuentan con características estipuladas por el usuario en cuanto a distribución de su contenido, temas y navegabilidad de acuerdo a los modelos IFML y UML diseñados.

1.1.5 HTML5

HTML (*HyperText Markup Language*, Lenguaje de Marcado de Hipertexto) usado para el desarrollo de aplicacions Web. Está constituido por un conjunto de etiquetas que sirven para dar estructura a una página Web. HTML5 es la útima versión de HTML, el cual cuenta con nuevos elementos, atributos y comportamientos; provee tres características: 1) escritura, 2) estilo y 3) funcionalidad. HTML5 es considerado el producto de la combinación de HTML, CSS y *JavaScript* (JS), las cuales son altamente dependientes y actúan como solo una unidad organizada bajo la especificación de HTML5 [5].

1.1.6 IFML

IFML es un estándar diseñado para expresar el contenido, la interacción con el usuario y el comportamiento del *Front-End* de las aplicaciones [6] que pertenecen a los siguientes dominios:

- Aplicaciones Web tradicionales basadas en HTML + HTTP
- RIAs soportadas por el estándar HTML5

- Aplicaciones móviles
- Aplicaciones cliente-servidor
- Aplicaciones de escritorio
- Interfaces usuario máquina integradas para el control de aplicaciones
- Aplicaciones de multicanales y contextualizadas

IFML se centra en describir la estructura y comportamiento de la aplicación tal y como lo percibe el usuario final.

1.1.7 Interfaz gráfica de usuario

También conocida como GUI por sus siglas en inglés (*Graphical User Interface*), se caracteriza por mostrar imágenes y objetivos gráficos como botones, iconos, entre otros, los cuales son intuitivos, sencillos de usar y fáciles de aprender [7].

1.1.8 Java Server Faces

Java Server Faces (JSF) es un marco de trabajo Web de Java que establece el estándar para construir interfaces de usuario del lado del servidor, introducido por *Sun Microsystems* para la simplificación del desarrollo de aplicaciones Web, apegado a la arquitectura Modelo-Vista-Controlador empleando archivos XML (*eXtensible Markup Language*, Lenguaje de Marcado Extensible) para la construcción de la vista y clases escritas con Java para la lógica de la aplicación [8].

JSF tiene dos funciones principales. La primera es generar una interfaz de usuario, generalmente una respuesta HTML que se sirve a un visualizador Web y se ve como una página Web y la segunda función de JSF es responder a eventos generados por el usuario en la página invocando escuchas del lado del servidor [9].

1.1.9 jQuery

Es una biblioteca de JavaScript, desarrollada para simplificar la programación del lado del cliente que interactúa directamente con HTML. Esta biblioteca es de código abierto y posee licencia dual (MIT y GNU). La sintaxis de jQuery fue desarrollada para hacer más simple la manipulación de documentos HTML, manejo de eventos, creación de

animaciones, y el desarrollo de aplicaciones AJAX (*Asynchronous JavaScript And XML*, JavaScript Asíncrono y XML) mucho más simple con una Interfaz de Programación de Aplicaciones (API) que funciona en múltiples visualizadores Web [10].

1.1.10 Maqueta (*Mockup*)

También llamados bosquejos o maquetados son usados para representar prototipos de interfaces de usuario de un sistema subjetivo [11].

Esta representación proporciona información a mediana fidelidad, capaz de introducir elementos gráficos y visuales, convirtiéndose así, en un modelo a escala de la aplicación a desarrollar, el cual se utiliza para obtener la representación gráfica que tendrá y siendo capaz de mostrar los fundamentos de su funcionalidad, incluyendo detalles visuales como colores, tamaños, tipografías, iconografía, entre otros [12].

La herramienta más utilizada para la realización de *Mockups* es *Sketch*[13].

1.1.11 Wireframe

Un *Wireframe* es equivalente al esqueleto o estructura de un sitio Web o aplicación. Contiene mayor detalles que un simple bosquejo realizado a mano alzada con papel y lápiz. Se realiza por página (tantas conformen la aplicación) e incluye los formularios y sus componentes. Se usa para indicar la estructura, el contenido por omisión y elementos que permitan visualizar funcionalidad tales como ligas y botones pero sin considerar la navegación. La herramienta más utilizada para la realización de *Wireframes* es *Balsamiq* [14]. Cabe mencionar que la palabra *Wireframe* no es muy usada en el diseño de interfaces de aplicaciones Web, la que es más común es *Mockup* aunque su significado sea diferente; es por eso que en el documento de la tesis se usa la palabra Mockup para referirse al proceso de describir la estructura que tendrán de los formularios de las aplicaciones Web.

1.1.12 MVC

Conocido como Modelo Vista Controlador, es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Está compuesto por

tres capas, que son el modelo, la vista y el controlador; por un lado define componentes para la representación de la información y por otro la interacción del usuario [15].

El modelo (*Model)* es la lógica del negocio, la cual representa la información en la que opera la aplicación.

La vista (*View*) o interfaz de usuario, está formada por la información que visualiza el usuario y los mecanismos de interacción.

El controlador (*Controller*) responde a las acciones del usuario e invoca cambios en el modelo o genera la vista apropiada, dependiendo de las peticiones del usuario [16].

1.1.13 PHP

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

Se utiliza principalmente de tres formas:

1. Como *script* del lado del servidor: En la creación de contenido Web dinámico.

2. Como *script* de línea de comando: Para la ejecución de *scripts* de línea de comando con la finalidad de realizar la administración del sistema, como lo son realizar copias de seguridad o análisis de registros, entre otras tareas.

3. Para generar aplicaciones de GUI del lado del cliente: Mediante el uso de PHP-GTK le es posible escribir aplicaciones multiplataforma GUI [17].

1.1.14 PrimeFaces

Es una biblioteca de código abierto y uso gratuito, la cual cuenta con una colección de componentes enriquecidos para la creación de interfaces de usuario y está integrada

para usarse con JavaServer Faces y así proporcionar una mayor funcionalidad a aplicaciones del lado del cliente [18].

Entre sus características principales se encuentran:

- Cuenta con un abundante conjunto de componentes (*HtmlEditor*, diálogos, autocompletado y gráficos, por mencionar algunos).
- Incorpora AJAX basado en el estándar de las APIs JSF AJAX.
- Su configuración es nula y no necesita de dependencias externas, el único requisito es incorporar la biblioteca de *PrimeFaces* a una aplicación de JSF estándar.
- Cuenta con un conjunto de herramientas de interfaz de usuario para el desarrollo de aplicaciones de dispositivos móviles.
- Dispone de numerosos temas incorporados por medio del marco de trabajo *Skinning*.
- Comunidad de usuarios grande y activa.
- Incluye soporte para herramientas de diseño de temas visuales.
- Posee una amplia documentación [18].

1.1.15 Traductor

Se define como un programa con la capacidad de traducir o convertir desde un texto o programa escrito en un lenguaje fuente hasta un texto o programa equivalente escrito en un lenguaje destino. Los traductores engloban tanto a los compiladores (en el que el lenguaje destino suele ser código máquina) como a los intérpretes (en los que el lenguaje destino está construido por las acciones atómicas que puede ejecutar el intérprete) [19]. A continuación en la Fig. 1.1 se muestra el esquema básico que compone a un traductor.

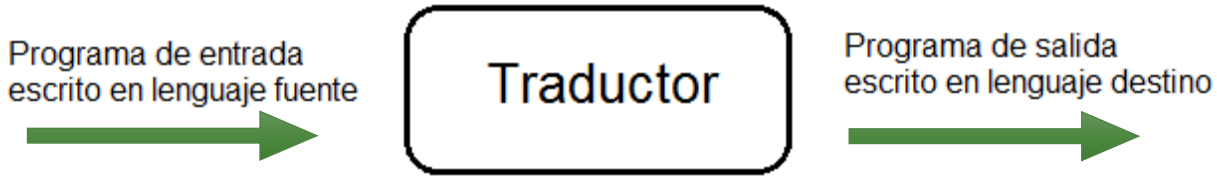

Fig. 1.1 Esquema general de un traductor

1.2 Planteamiento del problema

En el desarrollo de aplicaciones Web se le dedica mucho tiempo al diseño gráfico, para hacerlas llamativas y que cumplan con cierto estilo para facilitar la navegación. La acción de maquetado facilita a los analistas para obtener los requerimientos funcionales, a los diseñadores para saber en dónde distribuir los controles y componentes que formarán la aplicación y a los desarrolladores para tener una idea de lo que se requiere codificar y cómo debe funcionar dicha aplicación.

El generador de RIAs desarrollado por Estévez que lleva por nombre ***ITOs IFML based Rich Internet Applications Generator*** **(ITOs_IBRAIN)**, permite la obtención de esqueletos de las aplicaciones modeladas en IFML y UML, en el código generado se visualiza la aplicación con distribuciones básicas. Con respecto a los formularios que pertenecen a la aplicación, sólo cuentan con una distribución vertical, los controles se colocan uno debajo del otro, esto hace que dé la impresión que se está desperdiciando espacio importante en la pantalla. En los sistemas de información donde la captura de datos es muy extensa esto sería poco práctico porque no es natural presentarle al usuario final formularios tan extensos si es posible tener una mejor distribución, así que el desarrollador aún tendría que invertir tiempo en la codificación para mejorar la distribución de los controles en cada formulario de tal manera que tenga una mejor organización visual para un usuario final. Esto representa un problema ya que, uno de los propósitos de ITOs_IBRAIN es disminuir el tiempo de codificación del desarrollador para que su enfoque sea más fuerte en el negocio.

Así que por lo anterior, en este trabajo se propone desarrollar un módulo que permita el maquetado de formularios para Aplicaciones Enriquecidas de Internet a través de la

integración con la aplicación ITOs_IBRAIN, teniendo como resultado Aplicaciones Web que cuenten con formularios un diseño gráfico específico y organizado, el cual lo decidirá el usuario a través de una interfaz gráfica, que le permitirá arrastrar y soltar los controles de los formularios que haya diagramado en IFML.

1.3 Objetivo general y específicos

A continuación se presentan el objetivo general y los objetivos específicos los cuales atiende el presente proyecto de tesis.

1.3.1 Objetivo general

Desarrollar un módulo de maquetado que permita la selección específica y distribución de controles dentro de formularios, que tome como entrada una Representación Intermedia (RI) y afecte la creación de Código Final del Generador de Aplicaciones de Internet desarrollado en una tesis previa, tomando en consideración las características del lenguaje elegido y la posibilidad de ser responsivo.

1.3.2 Objetivos específicos

- Analizar la arquitectura de ITOs_IBRAIN para identificar los requerimientos técnicos del módulo.
- Analizar las diversas técnicas de creación de interfaces gráficas de usuario, como maquetado y desarrollo de prototipos, para elegir las que mejor se adapten a ITOs_IBRAIN.
- Analizar las tecnologías de graficación disponibles para elegir las que mejor se adapten a ITOs_IBRAIN.
- Identificar las diversas implementaciones posibles de los controles visuales (*widgets*) al menos en PrimeFaces y jQuery para incluirlas en el módulo.
- Implementar el módulo de maquetado al generador ITOs_IBRAIN.
- Realizar los casos de estudio (uno por cada combinación disponible en el Generador de Aplicaciones, al menos JSF- PF y PHP - jQuery).

1.4 Justificación

ITOs_IBRAIN tiene como finalidad crear código fuente ejecutable en PHP con jQuery o JSF con PF, el cual genera la vista, el modelo y el controlador de una aplicación Web modelada, permitiéndole al desarrollador que pueda dedicarle más tiempo al desarrollo del modelado de la aplicación en IFML y UML y menos a la codificación de una Aplicación Enriquecida de Internet.

Así que el propósito de este módulo a generar es obtener como resultado la integración de una GUI con el generador ITOs_IBRAIN, que permita realizar el maquetado de formularios afectando la generación de archivos con respecto a la vista de la aplicación ya sea en PHP con jQuery o con *Java Server Faces* con *PrimeFaces* y así tener una aplicación más organizada y estructurada visualmente, según sean los requerimientos del cliente. Facilitando así al desarrollador este proceso, a veces tan repetitivo y tedioso para que se enfoque directamente en la funcionalidad (controlador) de la aplicación que haya decidido generar.

Capítulo 2 Estado de la práctica

En este capítulo se dan a conocer algunos trabajos relacionados con el proyecto de tesis.

2.1 Trabajos relacionados

En [20] se estudió el problema que se tiene en la ingeniería de requerimientos cuando se necesita obtener los requerimientos funcionales de un sistema, ya que existen defectos típicos tales como la ambigüedad, la inconsistencia, que estén incompletos, o la falta de detalle; siendo la mayoría problemas de comunicación. Lo que se propuso fue implementar *"mockups"* (maquetas) junto con los casos de uso para mejorar la fase de obtención de requerimientos funcionales facilitando la comprensión y reducir así los defectos originales de esta fase. Para ponerlo a prueba se hicieron casos de estudio donde se aplicó lo anteriormente mencionado. Y como resultado se obtuvo que al agregar *"mockups"* en esa fase se incrementó significativamente la comprensión y la eficiencia de la obtención de requerimientos en un 69% y 89%, respectivamente. Aún así el tiempo en esfuerzo para la obtención no fue tan significativo como cuando no se usaron los *"mockups".*

Brambilla et al. propusieron en [21] un modelo de integración entre aplicaciones web y aplicaciones móviles por medio del estándar de IFML (*Interaction Flow Modeling Language*, Lenguaje de Modelado en un Flujo de Interacción) dando así soluciones a problemáticas que existen en el desarrollo de estas dos aplicaciones. Ya que en primera instancia el modelado de casos de uso, presentaciones, flujos de interacción y características son distintas para ambas tecnologías. Esto hace que se necesiten distintos diseños, métodos y herramientas, los cuales requieren el doble de tiempo de desarrollo, mantenimiento y costos. Para mitigar lo anterior los autores propusieron un novedoso enfoque que admite el diseño de aplicaciones web y móviles a partir de un diagrama en IFML. Una vez desarrollado el diagrama IFML, el modelo propuesto se

refina y se detalla para las dos plataformas (web y móvil) asegurando la trazabilidad de la plataforma a partir de los requisitos a la aplicación en ejecución para proporcionar una buena experiencia al usuario.

Raneburger et al. [22] presentaron un proceso que se apoya de herramientas para facilitar la exploración y la evaluación de alternativas de diseño e interacción a través de la generación automática de GUI (*Graphical User Interface,* Interfaz Gráfica de Usuario) por medio de UCP (*Unified Communication Platform*, Plataforma de Comunicación Unificada), para lograr una aplicación en ejecución más rápida y con menos esfuerzo, en comparación con el proceso tradicional, permitiéndole así al diseñador encontrar una alternativa más adecuada. El problema es que al desarrollar una GUI se necesita un nivel de abstracción alto, así como determinar el comportamiento externo de interacción con el usuario y en la mayoría de los casos el diseño de interacción se deja sin especificar o en las manos del diseñador. La solución propuesta es enfocarse en dar un soporte para la mejora del diseño de la interacción para varios dispositivos ayudando al mismo tiempo a mejorar el desarrollo de la lógica de la aplicación. El proceso propuesto es iterativo y permite facilitar el mejoramiento del diseño de la interacción de alto nivel para la GUI en cada una de las iteraciones desarrolladas, ya que permite la flexibilidad a cambios. En los casos de estudio se enfocaron más en aplicarlo a aplicaciones relativamente grandes. Aún así cabe resaltar que este proceso propuesto es iterativo mientras que el desarrollo de software en general es usualmente iterativo e incremental.

En [23] Hamdani et al. presentaron una SRL (*Systematic Literature Review,* Revisión Sistemática de Literatura) para examinar las aplicaciones IFML. Esta SRL seleccionó y analizó 22 estudios de investigación publicados entre el 2014 y el 2017 donde se usó IFML. El análisis de los 22 estudios permitió identificar cuatro áreas principales donde IFML se aplica con frecuencia, las cuales son: aplicaciones móviles, aplicaciones Web, aplicaciones de escritorio, entre otras. Uno de los análisis que se realizaron, mostró la capacidad de integración que tiene IFML con otros lenguajes de modelado los cuales fueron: a) UML, b) ODM (*Decision Management*, Gestión de decisiones), c) SOA ML

(Arquitectura Orientada a Servicios con UML), d) Mob ML y Web ML. También se analizaron las herramientas y marcos de trabajo que usa, clasificándolos de la siguiente manera: a) herramientas de modelado, b) herramientas de transformación de modelo y c) herramientas de validación de modelo.

Al final se concluyó que IFML es capaz de ser una herramienta de soporte, ya que es un estándar que maneja varios dominios, pero Hamdani et al. dijeron que las herramientas con las que cuenta IFML aún no son lo suficientemente maduras para ser utilizadas en aplicaciones complejas y grandes. Por lo cual es necesario seguir investigando las herramientas que ofrece IFML e identificar las limitaciones con las que cuenta. Aún así IFML incorpora criterios de mejora para el diseño e implementación del *Front-End* (Capa de presentación).

Oran et al. [24] hicieron hincapié en que la comunicación para la obtención de los requisitos es esencial en todos los proyectos de software, ya que es necesario tener un buen entendimiento durante todo el ciclo del proceso. Los analistas deben tener la capacidad de entender los requerimientos del cliente y al mismo tiempo de comunicarlos al resto del equipo de forma clara y eficaz. Pero autores citados en el artículo afirman que hasta profesionales en el área tienen dificultades con respecto a la validación y comprensión en la especificación de requisitos. Lo que estos autores propusieron fue usar casos de uso o historias de usuario para la obtención de requisitos, evaluando y comparando el grado de corrección de las especificaciones en los "*mockups*" (maqueta) creados, cuestionando si el uso de estas dos técnicas afecta la comunicación de requisitos para la construcción de "*mockups*" de forma diferente. Los resultados finales demostraron que al usar los casos de uso generó más defectos en la parte de la construcción de los "*mockups"* y menos defectos en la parte de especificación. Sin embargo, estos resultados no son suficientes para determinar cuál de las dos especificaciones es mejor o peor, ya que no hay una diferencia significativa.

En [25] Lasecki et al. propusieron un sistema llamado "*Aparition*" (Aparición) que usa herramientas y técnicas para la creación de prototipos de sistemas interactivos en el

tiempo que lleva describir simplemente la idea visual y verbalmente. Este proceso empieza con los usuarios que dibujan la interfaz y la describen en voz alta en un lenguaje natural mientras que un grupo de trabajadores hacen bosquejos, refinan detalles de la interfaz, agregan animaciones controlando el desarrollo del prototipo. Los bocetos se realizan en un promedio de ocho segundos, esta velocidad permite crear prototipos momentos después de pensar la idea. Para usar "*Aparition*" es necesario un grupo de personas que realicen micro tareas para coordinar un conjunto complejo de objetivos en tiempo real. Los prototipos creados con "*Aparition*" tienen más del 90% de precisión con lo que desea el usuario, y el sistema es capaz de crear elementos en ocho segundos y prototipos de comportamientos en tres segundos, todo esto, con base en lo que el usuario vaya describiendo. Este sistema permite a los diseñadores crear prototipos de manera rápida, como para iterar en una sola sesión, mejorando así la retroalimentación que reciben.

Los autores del artículo [26] realizaron un estudio acerca de la sustitución del papel y lápiz a la hora de realizar bocetos en las primeras etapas de tareas de "*sensemaking*", con un prototipo de aplicación usando una Tablet y un bolígrafo en conjunto, la aplicación se llama *SketchViz,* la cual centra fácilmente la restructuración de la información para los usuarios. Para ponerlo a prueba se realizó un estudio comparativo con 25 participantes que realizaron una determinada tarea usando lápiz y papel o la aplicación *SketchViz*. Durante la realización de las tareas los usuarios comentaron que les gustó la forma en la que la aplicación permitió cambiar de tamaño y la posición de objetos dibujados en el lienzo (el cual es distinto al de herramientas actuales) y que permite una mejor posibilidad de cambiar entre los detalles y errores en comparación con el que se tiene con el papel y lápiz. Como resultado final se obtuvo que la aplicación ofrece una experiencia a los participantes que les resulta muy familiar como cuando usan papel y lápiz excepto por algunos problemas técnicos que se presentaron en el prototipo.

La problemática que se presentó en [27] se centró en las fases tempranas del desarrollo de aplicaciones; ya que en este punto, no es posible tener interactividad con la aplicación haciendo que los usuarios no tengan una experiencia realista. Por otro lado, algunos diseñadores implementan los prototipos. El artículo puso en comparación prototipos realizados en Visual Basic o HTML, estos presentan una alta interactividad, por otra parte, si no se presentan imágenes visualmente atractivas se dice que carecen de diseño de interfaz. Aunque los diseñadores prefieren realizar prototipos con baja interactividad en la etapa inicial también les gustaría presentar prototipos con alta interactividad, para verificar el flujo de la información y que los usuarios prueben la interfaz en la vida real. Es por esto que los autores del artículo presentaron una herramienta de creación de prototipos que consiste en una pluma *nCode* llamada Neo, una plantilla de papel y una aplicación móvil desarrollada en Android. El proceso empieza cuando el diseñador dibuja las interfaces de usuario y las transiciones de pantalla de interacciones en las plantillas, cada trazo realizado se transmite a la aplicación móvil conectada. Después de realizar la conversión de los trazos, las interacciones se muestran en la aplicación móvil y el diseñador interactúa con la aplicación como con cualquier aplicación normal. Después de realizar el estudio y poner a prueba la aplicación con diseñadores, la conclusión a la que se llegó fue que los diseñadores no están a gusto haciendo líneas en los bocetos para mostrar la interactividad; los diseñadores acordaron que es útil en la generación de la aplicación pero que no sería útil en un entorno de trabajo y menos en la revisión o reutilización.

En [28] se describió la problemática de desarrollo que tienen las aplicaciones web en la actualidad, ya que su desarrollo es una tarea grande y compleja que involucra muchas fases en las cuales intervienen muchos trabajadores en conjunción con los usuarios finales. En su mayoría las aplicaciones Web fracasan por problemas de usabilidad ya que no logran cumplir sus objetivos o los deseos del cliente final; este fracaso se deriva principalmente de la falta de comunicación entre los equipos de desarrollo, usuarios finales, especialistas y diseñadores, entre otros. Para mitigar lo anterior, los autores de este artículo proponen incorporar un canal de comunicación para automatizar el

proceso entre los prototipos de interfaz y la arquitectura de software, para reducir la brecha entre el usuario final y los desarrolladores de software. Este enfoque está dirigido a idear una forma de automatizar la traducción de prototipos de usuario hacia una descripción formalizada como los diagramas UML o ADL (*Architecture Definition Language,* Lenguaje de descripción de arquitectura) usando XML (*eXtensible Markup Language*, Lenguaje de Marcado Extensible). Al final se concluyó que esta automatización que proponen puede ser un hito importante para el desarrollo de aplicaciones web con usabilidad alta durante las fases del desarrollo de software continuo e iterativo.

En la actualidad existen metodologías basadas en modelos que son compatibles con la generación automática de código, esto ayuda a mitigar costos de desarrollo de software pero en el caso de los sistemas interactivos crece el desafío de generación automática, ya que un alto nivel de automatización requiere el uso de modelos detallados; lo que es contrario al proceso de desarrollo iterativo. Por otro lado, la arquitectura de software implica una distinción entre el modelo utilizado en la lógica de negocio y la vista. Por lo anterior, en [29] se presentó MODUS (*Model-based User Interfaces Prototyping*, Modelo Basado en Prototipos de Interfaces de Usuario); una herramienta que aborda el enfoque de la generación semiautomática de interfaces de usuario para aplicaciones web. Concretamente es un *plug-in* que se instala en Eclipse y desde ahí se usa para la generación de código ejecutable. Como entrada se necesita un diagrama de clases, el dominio de la aplicación y las especificaciones de la interfaz para entregar como resultado código ejecutable. A través del *plug-in* fue posible probar la viabilidad de generación de la interfaz de usuario completa para una aplicación y así demostrar que se generó realmente lo que un usuario final necesitaba.

En [30] se presentó la problemática que se tiene cuando se requiere realizar el desarrollo de una aplicación Web y Móvil. En el caso del desarrollo para móviles se requiere hasta el desarrollo en varias plataformas (Sistemas operativos distintos). Para esto existen herramientas de plataformas cruzadas, que permiten la creación y

distribución de aplicaciones móviles para múltiples plataformas, aunque esto sólo es por medio de codificación absoluta en HTML (*Hyper Text Markup Language*, Lenguaje de Marcas de Hipertexto) y JS (JavaScript). Por otra parte, mencionaron que MDD (*Model Driven Development*, Desarrollo Dirigido por Modelos) muestra un enfoque alternativo para este paradigma, ya que difiere de las plataformas cruzadas promoviendo los prototipos iniciales, mitigando la divergencia entre las especificaciones y la implementación típica de los cambios rápidos de requerimientos. Por todo lo anterior los autores propusieron una herramienta basada en MDD que se enfoca en la generación automática de aplicaciones para plataformas cruzadas, tanto Web como Móvil con especificaciones IFML. Se realiza un mapeo de IFML a PCN (es una variante de las redes Petri) especificando si es Web o Móvil y se obtiene un prototipo de una aplicación ejecutable que está lista para entregar. Todo esto se realiza a través de la herramienta Web IFMLEditor.org

Practicar la realización de bosquejos es de suma importancia para los diseñadores, no solo para la externalización de ideas de su mente, sino también como impulso para la generación de nuevas ideas o el surgimiento de nuevo conocimiento con una específica propiedad visual. Por lo anterior, los autores de [31] presentaron un marco de trabajo conceptual para abordar el bosquejado de interacciones inactivas. Este marco de trabajo se organiza en un mapa bidimensional; la primera dimensión se basa en la interactividad incorporada por las representaciones y la segunda se basa en la expresividad de términos de experiencia del usuario (UX, *User experience*). Este marco incluye bocetos en papel, maquetas y bocetos de hardware vinculados por atributos de interactividad. Este marco presentó distintas formas de bosquejo en el diseño de interacción inactiva; centrándose en la inclusión de diferentes niveles de bosquejo ligado por los atributos de interactividad deseada.

En la actualidad los dispositivos habilitados para el uso de internet son tan diversos en formas y tamaños que representan desafíos en el desarrollo muy grandes. Aún así, todos estos dispositivos son capaces de mostrar páginas interactivas desarrolladas en

HTML5 y JS; por otra parte, los problemas de *Back-End* son mitigados por las APIs (*Application Programming Interfaces*, Interfaz de Programación de Aplicaciones). Y por último para la generación de código para *Back-End* se tiene OpenAPI. Estas herramientas son amigables para desarrolladores, más no para diseñadores o usuarios finales. Por lo anterior, en [32] se plantearon automatizar el proceso de creación de prototipos de interfaces, cuyo diseño sea adaptado a las demandas específicas de un usuario final. La solución al problema se basa en una herramienta para generar interfaces Web, con la ayuda de documentación API. Para esto se conceptualizaron dos pasos, el primero transforma documentación de OpenAPI a un modelo intermedio de IFML y el segundo paso genera HTML5 y JS dando como resultado prototipos de interfaces.

Moreira y Paiva en [33] presentaron una nueva metodología llamada PBGT (*Pattern-Based GUI Testing,* Patrones Basados en Pruebas de Interfaz de Usuario), la cual ayuda a automatizar el proceso de prueba de las GUI. Esta metodología toma muestras de entrada utilizando patrones de prueba de UI, que proporcionan una estrategia reutilizable que es configurable. Teniendo como objetivo principal proporcionar un nuevo paradigma que promueve la reutilización de estrategias de prueba para la GUI. Esta necesidad surgió con base en el tiempo que los desarrolladores pasan escribiendo código para desarrollar GUIs y la complejidad que se tiene en la actualidad para hacerlo, ya que los niveles de lógica son muy altos y complejos por lo que es necesario realizar pruebas de software. Y una de las desventajas de esto es que el proceso de realizar pruebas es difícil, lleva mucho tiempo y es costoso. Por lo anterior, los autores de este artículo propusieron esta nueva metodología, usando los patrones de prueba de UI para modelar aplicaciones Web reales, encontrando fallas reales en el software, pero que aún pretenden seguir aumentado el número de patrones de prueba para hacer frente a las más recientes capacidades de diseño de las GUIs.

En [34] los autores explicaron la problemática que se da cuando se requiere desarrollar un sistema grande. En estos casos, si se quiere mitigar errores, el sistema se debe diseñar y modelar adecuadamente; para esto se usa UML pero carece de medios para describir el *Front-End* de una aplicación, por lo cual surgió WebML el cual usa notaciones visuales y metodologías para el diseño de formularios complejos. Más tarde WebML evolucionó a IFML para enfocarse más en el *Front-End* de una aplicación. Por otra parte el proceso de pruebas se vuelve más complicado, ya que el sistema se debe probar para verificar cada requerimiento y su funcionalidad de forma correcta. Probar el *Front-End* es aún más complicado, ya que es un proceso no automatizado, el cual necesita trabajo manual. Las herramientas que implementan IFML son capaces de generar la interfaz gráfica de una aplicación y casos de prueba para probar la UI. Por lo anterior, los autores propusieron una solución para la generación de casos de prueba, basándose en los modelos IFML de una aplicación, esto se realiza a través de una transformación de ciertos constructores que ofrece IFML convirtiéndolos en código JS y así obtener los casos de prueba ejecutables.

Los autores mencionaron que en esta etapa solo algunos constructores están soportados, pero se pretende cubrir todos los constructores que ofrece IFML.

2.2 Análisis comparativo

La Tabla 2.1 muestra de manera abstracta la información relevante de cada artículo mencionado en la sección anterior, con el fin de realizar un análisis comparativo.

Tabla 2.1 Análisis comparativo de los artículos relacionados

	Problema	**Tecnologías**	**Resultado**	**Estado**
[20]	En la ingeniería de requerimientos es común tener problemas típicos de ambigüedad, inconsistencia, no ser tan específico o tener información incompleta. Siendo la mayoría problemas de comunicación.	*Mockups* Casos de uso	Informe sobre los resultados de una serie de experimentos controlados diseñados para evaluar si los participantes, de diferentes perfiles, se benefician al usar *mockups* en la fase de obtención de requisitos funcionales usando casos de uso. Y el informe mostró un incremento significativamente de la comprensión y eficiencia de la obtención de requerimientos favorablemente.	Mejoras a futuro
[21]	El modelado de aplicaciones móviles y web es totalmente distinto y en muchos casos los	IFML	Un modelo de integración entre aplicaciones web y aplicaciones móviles por medio del estándar	Mejoras a futuro

	Problema	**Tecnologías**	**Resultado**	**Estado**
	usuarios finales necesitan el desarrollo de sus aplicaciones para ambas plataformas. Esto hace que los desarrolladores tengan que hacer para cada una de estas plataformas distintos diseños, métodos y herramientas, los cuales requieren el doble de tiempo de desarrollo, mantenimiento y costos.		IFML.	
[22]	Existen varios inconvenientes cuando se desarrolla GUI, ya que se necesita un nivel alto de abstracción así como determinar el comportamiento externo de interacción con el usuario y en la mayoría de los casos se deja sin especificar.	BPMN (*Business Process Model and Notation*, Notación de modelado del proceso de negocios) UCP	Presenta un proceso apoyado de herramientas que facilita la exploración y evaluación de alternativas de diseño e interacción a través de la generación automática de la interfaz gráfica de usuario usando UCP.	Finalizado

	Problema	**Tecnologías**	**Resultado**	**Estado**
[23]	Dada la falta de información que hay de IFML y que es relativamente un estándar nuevo, se decidió realizar una SRL analizando 22 estudios de investigación para conocer más sobre este estándar.	No se menciona ninguna tecnología ocupada.	De los 22 estudios analizados la investigación dividió en cuatro categorías de aplicaciones: 1) móvil, 2) web, 3) Escritorio, 4) otras. También arrojó un estudio de las herramientas que conforman IFML y respondieron cinco preguntas en concreto.	Mejoras a futuro
[24]	Existen casos donde personas profesionales en la recolección de requerimientos tienen problemas con respecto a la validación y comprensión de los requerimientos.	Casos de uso Historias de usuario basadas en *Behavior Driven Development* (BDD) *Mockups* (maquetas)	Se realizó un estudio experimental, el cual demostró que entre las dos especificaciones usadas (casos de uso e historias de usuario) no hay mucha diferencia entre el número de defectos que se tienen al usarlas para obtener requisitos y crear "*mockups*". Ya que ambas obtuvieron un número similar de defectos cuando se pusieron a prueba en el caso de estudio.	Finalizado

	Problema	**Tecnologías**	**Resultado**	**Estado**
[25]	El esfuerzo requerido para producir un prototipo interactivo es un claro factor limitante en la práctica del diseño efectivo. Los prototipos se basan en un mecanismo continuo de prueba y error. Aun la construcción del prototipo interactivo más rápido tiene una lenta visualización a través de un boceto.	Bosquejos "*Aparition*"	"*Aparition"* es un sistema que cuenta con herramientas, el cual permite gestionar a un grupo de trabajo grande, este grupo ayuda colectivamente a los usuarios a crear prototipos de interfaz, bocetos y descripciones verbales en cuestión de segundos.	Finalizado
[26]	En una era llena de tecnología algunos usuarios prefieren usar papel y lápiz para realizar bocetos en las etapas de tareas de "*sensemaking*". Hacer eso representa un desperdicio de la era tecnológica, ya que existen herramientas tecnológicas que pretenden brindar la misma experiencia con bajos costos a errores.	*SketchViz*	Se realizó un estudio comparativo con 25 personas que usaron la aplicación o lápiz y papel para resolver diversas tareas. Este estudio arrojó como resultado que la aplicación ofrece una experiencia muy familiar a los usuarios con respecto al lápiz y papel.	Mejoras a futuro

	Problema	**Tecnologías**	**Resultado**	**Estado**
[27]	El uso de bocetos para el desarrollo de aplicaciones muestra una gran fidelidad visual pero dejan de lado la fidelidad interactiva, ya que no son capaces de mostrar el funcionamiento de la aplicación bosquejada. Aunque por otra parte si no se presentaran imágenes llamativas, el diseño carecería de diseño de interfaz. Por lo tanto, se requiere tener un equilibrio de ambas partes.	*nCode* (pluma)	Presentan una aplicación de creación de prototipos que consiste en una pluma llamada *nCode*, una plantilla de papel y una aplicación móvil capaz de convertir el bosquejo realizado en una aplicación móvil con cierto nivel de funcionalidad y navegación como una aplicación móvil cualquiera.	Mejoras a futuro
[28]	El desarrollo de aplicaciones web es una tarea grande y compleja, ya que involucra muchas fases y muchos trabajadores. Eso hace que exista una amplia brecha entre el equipo de desarrollo y los usuarios finales por problemas	UML ADL XML *Mockup*	Se obtuvo como resultado un canal de comunicación entre el usuario final y los desarrolladores que mitigue la brecha que existe entre ellos a través de la realización de una traducción entre los prototipos a una descripción formalizada como	Finalizado

	Problema	**Tecnologías**	**Resultado**	**Estado**
	de comunicación haciendo que en la mayoría las aplicaciones web fracasen por no cumplir los objetivos de usabilidad.		UML o ADL usando XML.	
[29]	Existen metodologías capaces de producir generación automática de código ayudando a mitigar el costo de desarrollo de software. Pero existen varios problemas si la aplicación a desarrollar tiene un nivel de interacción alto, ya que requiere el uso de modelos detallados y la arquitectura de software implica hacer una distinción entre la lógica de negocio y la vista.	Eclipse MODUS	Se presenta un *plug-in* compatible con Eclipse que permite la generación semiautomática de interfaces de usuario para aplicaciones Web.	Mejoras a futuro
[30]	El costo de desarrollar aplicaciones Web y Móviles son altos, ya que se tratan de tecnologías distintas y aunque	IFML PCN MDD Mockup	Se obtuvo una herramienta para la especificación y generación rápida y automática de prototipos Webs y Móviles usando	Finalizado

	Problema	Tecnologías	Resultado	Estado
	en la actualidad se cuenta con herramientas cruzadas que permiten la ejecución de aplicaciones móviles en varias plataformas se realiza solo con codificación en HTML y JS.		especificaciones de IFML. Esta herramienta soporta la semántica de IFML y realiza un mapeo a PCN, siendo una variante de las redes Petri para la simulación y la verificación de modelos. Y obtiene un prototipo de la aplicación Móvil o Web ejecutable.	
[31]	La realización de bosquejo es muy importante para los diseñadores. En la mayoría de los casos solo se realiza a lápiz y papel; siendo este proceso muy básico o tradicional. Ya que este es un proceso importante los autores decidieron crear un marco de trabajo conceptual.	No se menciona alguna tecnología.	Como resultado se obtuvo un marco de trabajo conceptual para abordar el bosquejado de interacciones inactivas. El cual se organiza en dos dimensiones, la primera se basa en la interactividad incorporada por las representaciones y la segunda en la expresividad de términos por la experiencia del usuario.	Mejoras a futuro

	Problema	**Tecnologías**	**Resultado**	**Estado**
[32]	En la actualidad existen gran variedad de dispositivos que son capaces de mostrar páginas web. Aún así existen varios problemas de Back-End como la escalabilidad, rentabilidad y mantenibilidad, todos estos problemas son sustentados por las APIs.	IFML OpenAPI HTML5 JS	Se obtuvo una herramienta que genera interfaces Web, con la ayuda de documentación API. Primero se transforma la documentación de OpenAPI a un modelo intermedio de IFML y después se genera código HTML5 y JS dando como resultado el prototipo de interfaces.	Finalizado
[33]	Los desarrolladores pasan mucho tiempo escribiendo código para desarrollar GUIs, por dos razones: la aceptación (que sean amigables) y facilitar a los usuarios finales. En la actualidad los niveles de lógica de las GUI son muy altos y más complejos por lo que se vuelve de suma importancia realizar pruebas de software. Aún así	Patrón de prueba para Interfaces Gráficas	Una nueva metodología llamada PBGT que ayuda a automatizar el proceso de prueba de las GUI.	Finalizado

	Problema	**Tecnologías**	**Resultado**	**Estado**
	las pruebas son difíciles, llevan mucho tiempo y son costosas.			
[34]	Al desarrollar sistemas grandes y complejos se recurre a herramientas de modelado para mitigar errores. Comúnmente se usa UML, ya que cuenta con varios diagramas que permiten modelar sistemas pero carece de medios para describir *Front-End*. Para mitigar esto surgió WebML, el cual años después se convirtió en IFML. Y finalmente la etapa de pruebas se vuelve compleja para el *Front-End,* ya que probar GUI es un proceso no automatizado, el cual necesita trabajo manual.	IFML	Una solución para la generación de casos de prueba, basándose en el modelado de aplicaciones usando IFML.	Mejoras a futuro

Después de analizar los artículos comparados anteriormente se concluye que la realización de maquetado o bosquejado de interfaces es importante, ya que brinda a los diseñadores una herramienta que permite facilitar la comprensión y obtención de requisitos funcionales de una aplicación. El permitirle al diseñador hacer un maquetado de formularios de sus aplicaciones sin escribir ni una línea de código facilita el proceso de creación de aplicaciones, ya que disminuye el tiempo de codificación de interfaces gráficas simplificando esta tarea. Cabe mencionar que en los artículos citados anteriormente [21], [30], [32] y [34]; donde se usa la tecnología de IFML, no se considera el maquetado para diseñar la interfaz de las aplicaciones Web y es por esto que el desarrollo del módulo de maquetado para formularios es importante, ya que al integrarlo al generador de aplicaciones enriquecidas de internet permitirá obtener aplicaciones Web ejecutables en PHP y JSF con una mejor distribución y diseño visual que hará más atractivas a la vista las aplicaciones resultantes.

2.3 Propuesta de solución

A continuación se presenta una descripción general de la propuesta de solución para el proyecto de tesis.

2.3.1 Descripción de la solución

La solución a la problemática explicada anteriormente, es la creación y adaptación de un módulo que permita al generador de Aplicaciones Enriquecidas de Internet el maquetado de formularios a través de una interfaz gráfica de usuario para mejorar el diseño visual de los formularios resultantes d la generación que se hayan modelado en IFML y UML; con esto se hará la modificación de los métodos que intervengan en la generación de código, específicamente en la capa de la vista para JSF con PF o PHP con jQuery según lo que el usuario haya descrito con anterioridad.

Para lograr lo anterior, la propuesta de solución se representa gráficamente en la Fig. 2.1, lo que está en recuadros negros son procesos pertinentes a ITOs_IBRAIN y lo que está en recuadros verdes son procesos propios del módulo de maquetado.

A continuación se explicará brevemente en qué consiste cada uno de los puntos que conforman la solución.

- El proceso de entrada consiste en elegir los modelos IFML y UML, seleccionar el lenguaje de salida, los temas y la plantilla que se usará en la aplicación Web.
- El siguiente paso es el reconocimiento de la gramática de los modelos ingresados para la transformación del lenguaje, en esta parte el generador realiza una representación intermedia de los modelos, justamente después de que termine este proceso se integrará un paso extra que permitirá obtener la información pertinente a los formularios que existan en la aplicación modelada.
- Después de que ya se conocen los formularios y los controles hay en cada página de la aplicación, se pasa a lo que es la interfaz de maquetado, este es el primer proceso adicional, que pertenece al módulo que se integrará al generador. Se basa en permitirle al usuario realizar el maquetado de la aplicación Web que requiere, a través de herramientas disponibles en la interfaz.
- Luego de haber realizado la distribución de los controles de los formularios de la aplicación, se pasa al siguiente proceso adicional perteneciente al módulo, el cual es la afectación del código de salida de la generación, ya que hasta este punto los formularios de la aplicación que se obtienen del generador cuentan con una distribución de forma vertical, los controles se colocan uno debajo del otro; así que se modificarán los métodos de la generación de código que afecten a la vista y distribución de controles.
- Y como último paso se tiene el proceso final que conforma al generador, el cual es la salida. Esta salida tiene como resultado darle al usuario archivos de código que siguen la arquitectura MVC, los cuales contienen toda la aplicación que corresponden a los modelos realizados en IFML y UML. De los documentos resultantes obtenidos por ITOs_IBRAIN, el desarrollo de este módulo se encargará de modificar y afectar únicamente lo relacionado con la vista de la aplicación Web en JSF con PF o PHP con jQuery.

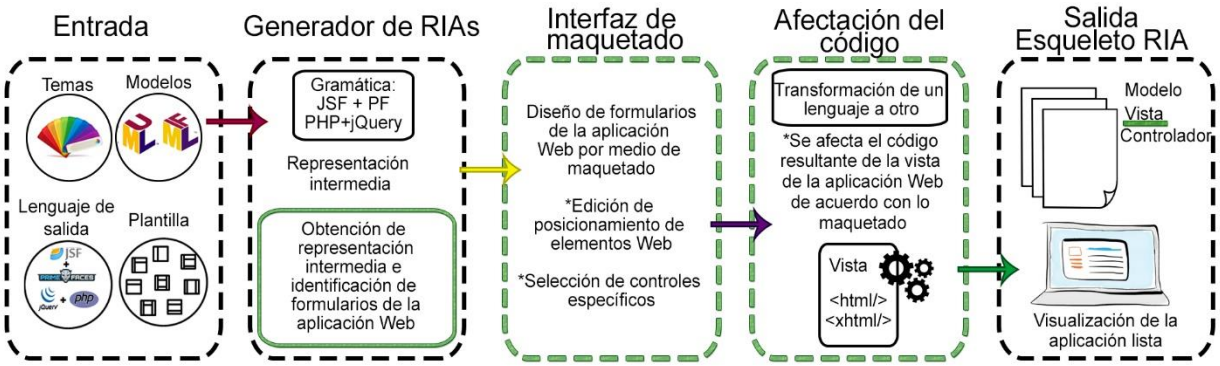

Fig. 2.1 Representación gráfica de la solución

2.3.2 Tecnologías a usar en la solución propuesta

Para desarrollar el proyecto de tesis se eligieron las tecnologías descritas en la Tabla 2.2 donde se explica la justificación de su elección.

Tabla 2.2 Justificación de la elección de tecnologías

Propuesta	**Justificación**
JavaFX	JavaFX es una biblioteca de desarrollo que cuenta con los componentes necesarios para el desarrollo en Java de interfaces gráficas de usuario; ya que está orientada a brindar una mejor experiencia al usuario creando aplicaciones enriquecidas. Otra de las características por la cual se eligió es que cuenta con la capacidad de integrarse a aplicaciones ya existentes desarrolladas con Java y acoplarse sin ningún problema. Esta característica es ideal ya que ITOs_IBRAIN está desarrollado en Java.
NetBeans	Se eligió como ambiente de desarrollo NetBeans ya que no necesita instalaciones extras de *plugins* para desarrollar aplicaciones en JavaFX. Y porque el equipo que se usará para desarrollar el módulo de maquetado cumple con los requisitos que exige este IDE para funcionar correctamente.
Scrum	Se eligió como metodología ágil Scrum ya que se acopla a trabajar con mínimo tres integrantes para designar los roles, es capaz de adaptarse para el desarrollo de tesis para alumnos de ingeniería tal como lo indica Cortez-Herrera en [35]. Además fomenta la

	comunicación entre el equipo y el dueño del producto, esto ayuda a minimizar malos entendidos de requerimientos; por la parte de los *Sprints* se tienen mejoras e incrementos continuos que no duran más de dos semanas, lo que la hace la metodología ágil ideal para el desarrollo de tesis por el tiempo que se tiene para su desarrollo. Aún así se deben hacer adecuaciones a la metodología ya que solo existen dos roles en el desarrollo del proyecto de tesis, el primero es el *Scrum Master* y el segundo es el equipo de trabajo. Para suplir el rol de dueño de proyecto (cliente) se realizó un análisis del generador y otro de aplicaciones de maquetado existentes en el mercado.

Capítulo 3 Aplicación de la metodología

En el presente capítulo explica el proceso que se llevó a cabo para desarrollar el módulo de maquetado de formularios para el generador de Aplicaciones Enriquecidas de Internet bajo el patrón arquitectónico MVC usando UML e IFML.

Este proyecto se realizó utilizando una adaptación de la metodología Scrum por dos aspectos importantes, el primero debido a que no existe un usuario final real que determine las funcionalidades del módulo de maquetado y el segundo que el proyecto de tesis se realizó de forma individual y no con un equipo de desarrollo.

3.1 Obtención de requerimientos

De primera instancia, al identificar la necesidad de incorporar un módulo de maquetado al generador ITOs_IBRAIN, se sabía que se debía dar soporte a funcionalidades como: a) “Arrastrar y soltar” elementos de una sección a otra, b) asegurar el crecimiento futuro de la aplicación (inclusión de nuevos lenguajes de salida), c) manejar un comportamiento de tipo asistente (*wizard*) que dirige al usuario por una serie de pantallas y d) obtener los datos a partir de la Representación Intermedia (elemento propio de ITOs_IBRAIN).

Sin embargo, no se contaba con el detalle completo de las funcionalidades propias de un módulo de maquetado, por lo cual, para determinar esos requerimientos funcionales y no funcionales, se realizaron ciertas actividades previas para complementarlos.

3.1.1 Análisis de las tecnologías de diagramación de *mockups*

El primer *Sprint* corresponde al análisis de herramientas que permiten el maquetado de aplicaciones. Este análisis complementó los requerimientos funcionales y no funcionales a aplicar al módulo de maquetado de formularios.

En la Tabla 3.1 se muestran las seis herramientas analizadas, como criterio de comparación se tomaron en cuenta características como manejo de interacción con el usuario, la forma en que posicionan los elementos visuales y las acciones que soportan entre otros elementos.

Tabla 3.1 Tabla comparativa de herramientas que permiten el maquetado

Herramienta	Balsamiq Mockup [14]	Sketch : Felipe [13]	UXPin [36]	JFrame Designer [37]	Visual Paradigm: UX [38]	Visual studio: Web [39]
Interacción	Permite arrastrar y soltar los elementos	Los controles se dibujan y aparecen en el área de modelado	Permite arrastrar y soltar los elementos	Permite arrastrar y soltar los elementos	Permite seleccionar el elemento y soltarlos en el área de modelado	Permite arrastrar y soltar los elementos
Posicionamiento de UI	La barra de componentes cuenta con una clasificación y está ubicada en la parte superior de la pantalla. Del lado izquierdo presenta una lista de maquetados. Del lado derecho una sección de propiedades y en el centro el espacio para el modelado. No permite modificar el posicionamiento de las secciones.	A la izquierda tiene una lista de componentes clasificados según su tipo. Del lado derecho un espacio para modificar las propiedades de los componentes y en el centro el espacio para el modelado. No permite modificar el posicionamiento de las secciones.	Cuenta con tres secciones, a la izquierda la lista de componentes básicos para el modelado de móviles sin clasificación, en la derecha la sección de propiedades de componentes y en el centro el espacio para el modelado. No permite modificar el posicionamiento de las secciones.	Contiene una sección para el modelado justo en el centro de la pantalla, la lista de componentes está ubicado a la derecha con una clasificación basada en el tipo y justo debajo de esa lista la sección de propiedades. Sí permite modificar el posicionamiento de las secciones.	La barra de componentes está a la izquierda de la pantalla sin ninguna clasificación, algunas propiedades de los componentes se modifican dando clic derecho y el espacio restante se usa para el modelado de las pantallas. No permite modificar el posicionamiento de las secciones.	Contiene dos paneles, el primero contiene la lista de los componentes que se pueden usar y está clasificada por el tipo y el segundo panel contiene las propiedades a modificar de los componentes. Al igual que las anteriores herramientas la sección de modelado está situado en el

Herramienta	Balsamiq Mockup [14]	Sketch : Felipe [13]	UXPin [36]	JFrame Designer [37]	Visual Paradigm: UX [38]	Visual studio: Web [39]
						centro. Sí permite modificar el posicionamiento de las secciones.
Distribución de maquetado	Cuenta con una cuadrícula trasera de guía pero no hay restricciones de distribución	Como solo modela para aplicaciones móviles los componentes se acomodan de forma vertical, siguiendo esa estructura	Posiciona los componentes del modelo en forma vertical ya que solo modela para aplicaciones móviles	Trabaja bajo el esquema de *grids* (cuadrículas) y el uso de paneles para indicar el posicionamiento de los controles	No cuenta con restricciones, permite el posicionamiento de los controles en cualquier lugar y no obedece a ninguna distribución particular	Permite una distribución vertical por omisión y tiene ciertas restricciones como que no permite colocar botones junto a tablas. Aún así permite colocar DIVs para modificar la distribución, pero debe ser indicado por el usuario
Salida	PNG, PDF	Exporta en CSS y SVG	PNG, PDF y HTML	Proyecto de NetBeans	Genera una animación de funcionalidad con	Proyecto de Microsoft

Herramienta	Balsamiq Mockup [14]	Sketch : Felipe [13]	UXPin [36]	JFrame Designer [37]	Visual Paradigm: UX [38]	Visual studio: Web [39]
					base en las *storyboard*	
Acciones: copiar, pegar, cortar, deshacer, rehacer	Si realiza todas estas acciones	Si realiza todas estas acciones	Si realiza todas estas acciones	Si realiza todas estas acciones	Si realiza todas estas acciones	Si realiza todas estas acciones
Para que dispositivos modela	Móvil y Web	Móvil (iOS)	Móvil (Android y iOS)	Aplicaciones de escritorio	Móvil (Android y iOS) y Web	Web
Dónde se ejecutan	Nube Escritorio (Windows y Mac) Google drive Con un tiempo de prueba de 30 días	Mac	Nube	Es una aplicación de escritorio que necesita la MVJ instalada previamente (Windows, Mac, Linux)	Escritorio en su versión estándar, profesional y empresa	Escritorio (Windows y Mac)
Qué tipo de código generan	Exporta JSON	CSS para obtener el diseño	HTML	Java	Ninguno	ASP.NET, Node.js, Python y JavaScript

3.1.2 *Backlog* del módulo de maquetado

Después de comparar los aspectos importantes con los que cuentan las diferentes herramientas analizadas, se destacan los siguientes requerimientos a tomar en cuenta para aplicarse en el módulo de maquetado de formularios:

1. **Interacción**: Arrastrar del Panel de controles disponibles los elementos (*widgets*) correspondientes a cada formulario y soltarlos en el área de maquetado, acción que realizan la mayoría de las herramientas analizadas.
2. **Posicionamiento**: La pantalla contará con tres secciones importantes como la mayoría de las herramientas analizadas, en la Fig. 3.1 se muestran estas secciones:
    a. **Verde**: Barra de menú de acciones ubicado en la parte superior. Esta sección contiene seis acciones, la primera permite regresar a la pantalla anterior (pantalla donde se realiza la configuración de la aplicación), la segunda avanza a la siguiente pantalla (pantalla donde se visualiza un resumen de la aplicación a desarrollar), la tercera guarda la distribución de los controles del formulario sobre el que se está trabajando en un almacenamiento temporal, la cuarta reinicia la distribución del formulario sobre el que se está trabajando y de los controles maquetado, la quinta opción permite deshacer las acciones hechas sobre el formulario seleccionado, como el posicionamiento y reposicionamiento de controles, el cambio de texto de una etiqueta o el cambio de visualización de un control y la última cancela todas las distribuciones asignadas a todos los formularios y la que tengan sus controles.
    b. **Rojo**: Esta sección ubicada en el lado izquierdo está dividida en dos partes, la primera es la lista de formularios donde aparecen todos los formularios que estén dentro de la aplicación modelada y la segunda es la lista de controles donde aparecerán todos los controles que sean parte del formulario seleccionado.
    c. **Azul**: Es la sección de maquetado de los formularios, previamente se debe haber asignado una distribución al formulario proporcionando el número de filas y columnas para el *Grid* (cuadrícula) y definir el

posicionamiento de las etiquetas (izquierda o arriba con respecto al campo de captura) para empezar la distribución de componentes. La característica de la visualización de una cuadrícula en el área de maquetado se inspiró en la herramienta *Balsamiq*.

d. *Wireframe* de la interfaz gráfica de usuario del módulo de maquetado:

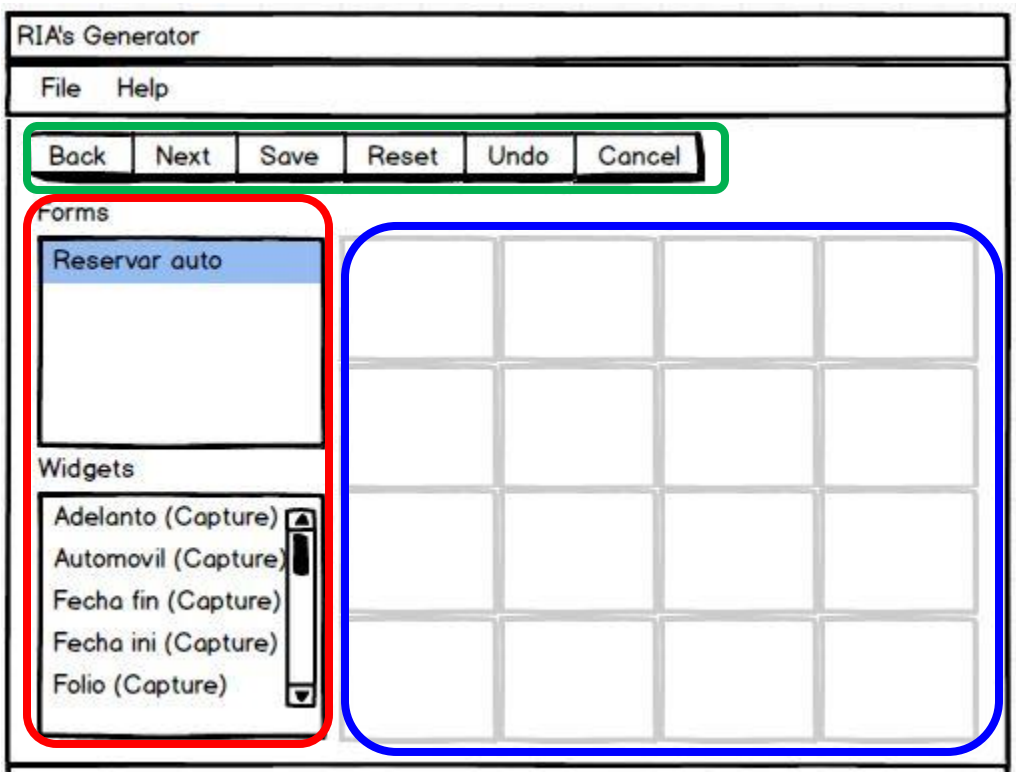

Fig. 3.1 Wireframe de la GUI del módulo de maquetado de formularios

3. **Distribución**: La manera en cómo se distribuirán los componentes de un formulario será a través de un *grid* (cuadrícula) previamente seleccionada, esta característica se basó en el *plugin* de NetBeans; el usuario debe indicar el número de filas y columnas para que se haga la distribución de componentes.
4. **Salida**: Adaptación de la representación intermedia basada en lo maquetado para cada formulario, esta característica es propia del módulo.
5. **Acciones**: No se aplicarán ninguna de las acciones de Cortar, Copiar, Pegar, Rehacer, que todas las herramientas permiten realizar por motivos de trazabilidad con lo modelado en IFML, es decir, no se permite incluir o eliminar controles respecto al modelo. La única excepción es la acción Deshacer que

está contemplada para que el usuario tenga la oportunidad volver sobre sus pasos en ciertas acciones sobre un formulario. Esta característica es propia del módulo de maquetado.

6. **Para qué dispositivos modela**: Esta herramienta modela únicamente la distribución de los componentes dentro de un formulario de una aplicación Web modelada previamente. Característica propia del módulo de maquetado.
7. **Dónde se ejecuta**: Es una aplicación de escritorio desarrollada en JavaFx, la aplicación ejecutable está empaquetada en un JAR que puede ser ejecutado en Windows, Linux y Mac.
8. **Qué tipo de código genera**: Después de maquetar los formularios, se modifican ciertos atributos de una representación intermedia, esta representación es la encargada de generar el código de salida, mismo que puede ser la combinación Java *Server Faces* con *PrimeFaces* o PHP con jQuery. Característica propia del módulo de maquetado.

Además de las características previamente mencionadas, se consideró incluir algunas otras debido a que toda la información de los formularios que recibe el módulo de maquetado viene ya definida desde una Representación Intermedia, por lo cual solo ciertas cosas deben ser modificables y que lo maquetado debe ser estrictamente consistente a lo generado en las aplicaciones de salida y para asegurar eso se incluyeron estas características extras:

- Al empezar el proceso de maquetado aparecerá la pantalla correspondiente al módulo con la lista de formularios disponibles para maquetar (ninguno estará seleccionado) y la lista de controles estará vacía así como el área de maquetado.
- Una vez seleccionado por primera vez un formulario se preguntarán las dimensiones del G*rid (*filas y columnas*)* y la posición de la etiqueta (*label*) con respecto a su campo de captura.
- El código resultante será responsivo, en jQuery soportado con divisiones (etiqueta DIV) + CSS y en *PrimeFaces* soportado con el elemento *PanelGrid* y con distribución interna (*layout*) de tipo tabular (*grid*), resaltando que no es

deseable que se use combinaciones de renglones (*row span*) o columnas (*col span*) ya que en esos casos no es posible asegurar el soporte responsivo.

- La lista de controles se llenará con los controles que tenga cada formulario modelado, no se mostrarán más controles de los ya estipulados y estará en constante cambio cuando se acomoden los controles dentro del *Grid*, ya que una vez que un control está dentro del *Grid* ya no será posible quitarlo a menos que se indique lo contrario con la acción Deshacer (*Undo*).
- Las etiquetas (*labels*) están asociadas con los controles que les corresponden, por lo que su asignación es en dúo, no hay asignaciones por separado.
- La posición de las etiquetas (*labels*) con respecto a sus campos de captura será de dos tipos a elegir por el usuario: arriba del campo o a la izquierda del campo.
- El texto de las etiquetas (*labels*) será modificable una vez que el control tenga asignada una ubicación en el G*rid*; haciendo clic derecho sobre el control, aparecerá un menú flotante con una opción para modificar el texto. Esta característica fue inspirada en Visual Paradigm (VP), las propiedades de los componentes en VP se modifican dando clic derecho al componente.
- Habrá un botón para deshacer acciones ("*Undo*") que permitirá al usuario deshacer ciertas acciones previamente realizadas:
    - Deshacer la acción de posicionar un control arrastrándolo de la lista de controles al *Grid*.
    - Deshacer el reposicionamiento de controles dentro del *Grid*.
    - Deshacer el cambio del texto de una etiqueta (*label*).
    - Deshacer el cambio de la visualización de un control de selección (simple o múltiple).
- La acción del botón Cancelar ("*Cancel*") hará que la aplicación cancele todos los formularios maquetados y muestre la interfaz tal y como al inicio de la ejecución del módulo de maquetado, eso quiere decir que se mostrará la lista de formularios sin selección alguna y la lista de controles estará vacía así como el área de maquetado.

- Para guardar el maquetado de un formulario es necesario que se coloquen todos sus controles en el *Grid*, de lo contrario aparecerá un mensaje de error que no permitirá guardarlo hasta haber asignado todos los controles de dicho formulario.
- Si el usuario no maquetó un formulario, la distribución correspondiente en la generación de código se hará por omisión en vertical.
- Si un formulario ya tiene almacenado su maquetado, se le permitirá al usuario volver a la distribución realizada y afectarla.
- El módulo de maquetado funcionará solo con el periférico de ratón.

3.2 Determinación de casos de uso

En la Fig. 3.2 se presenta el diagrama de casos de uso donde se especifican los requerimientos identificados para el módulo de maquetado a desarrollar.

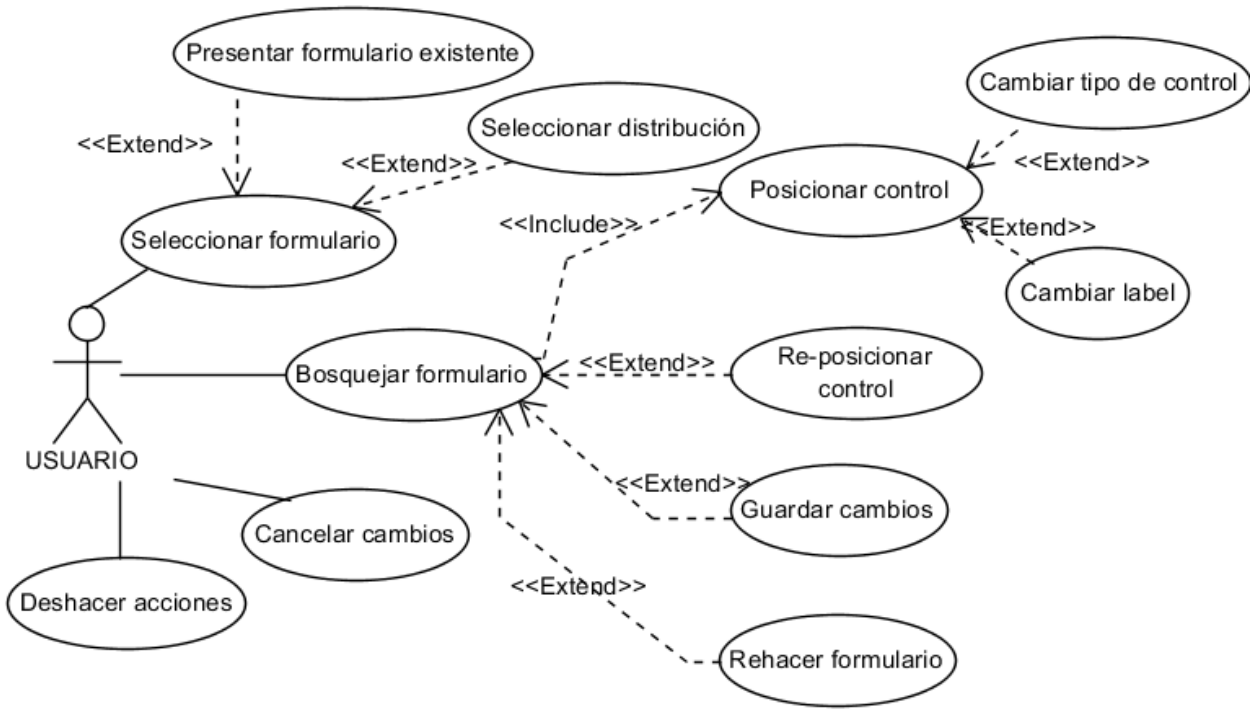

Fig. 3.2 Diagrama de casos de uso

Para entender mejor cada caso de uso, en la Tabla 3.2 se da una breve descripción de ellos.

Tabla 3.2 Descripción de los casos de uso

Nombre de caso de uso	Descripción
Seleccionar formulario	Permite la selección específica de un formulario del panel de formularios.
	Extend
	Seleccionar distribución: Permite al usuario determinar la distribución del número de columnas y renglones que tendrá el *Grid*. **Presentar formulario existente:** Si un formulario ya tiene una distribución asignada ésta se mostrará en el área de maquetado.
Bosquejar formulario	Permite al usuario distribuir controles de los formularios en el área de maquetado
	Include
	Posicionar control: Permite realizar la acción de arrastrar y soltar los controles dentro del *Grid.* **<<Extend>>** **Cambiar tipo de control:** Permite cambiar la representación visual de las listas de selección única o múltiple. **Cambiar etiqueta (*label*):** Permite editar el texto de la etiqueta (*label*) de un control.
	Extend
	Re-posicionar control: Permite cambiar de posición los controles dentro del *Grid* una vez que ya se hayan distribuido.
	Guardar cambios: Permite almacenar los cambios de la distribución de los controles en un “Almacenamiento temporal”.
	Rehacer formulario: Permite reiniciar la distribución completa de los controles de un formulario.
Cancelar cambios	Permite olvidar todas las distribuciones de controles realizadas sobre todos los formularios para comenzar a distribuir de

	nuevo.
Deshacer acciones	Permite deshacer la acción más reciente realizada sobre un formulario.

3.3 Aplicación de la metodología

La metodología elegida fue Scrum, por lo cual el desarrollo del proyecto de tesis está dividido en Sprints.

3.3.1 *Sprint* 0: Arquitectura del módulo de maquetado

Dada la complejidad que tiene el módulo y el soporte que necesita, se decidió colocar un *sprint* cero que corresponde a la definición de la arquitectura.

De acuerdo a la IEEE, una arquitectura representa “Conceptos fundamentales o propiedades de un sistema en un entorno definido, incorporado en elementos, las relaciones que existen entre ellos; y los principios que guían su diseño y evolución” [40]; con los requerimientos especificados anteriormente se determinó que la arquitectura a desarrollar debía ser fuerte y estar bien estructurada para ser capaz de soportar el desarrollo de los requerimientos funcionales y no funcionales complejos que se presentaron. En la Fig. 3.3 se muestra el primer nivel de la arquitectura para el módulo de maquetado mediante un diagrama de componentes.

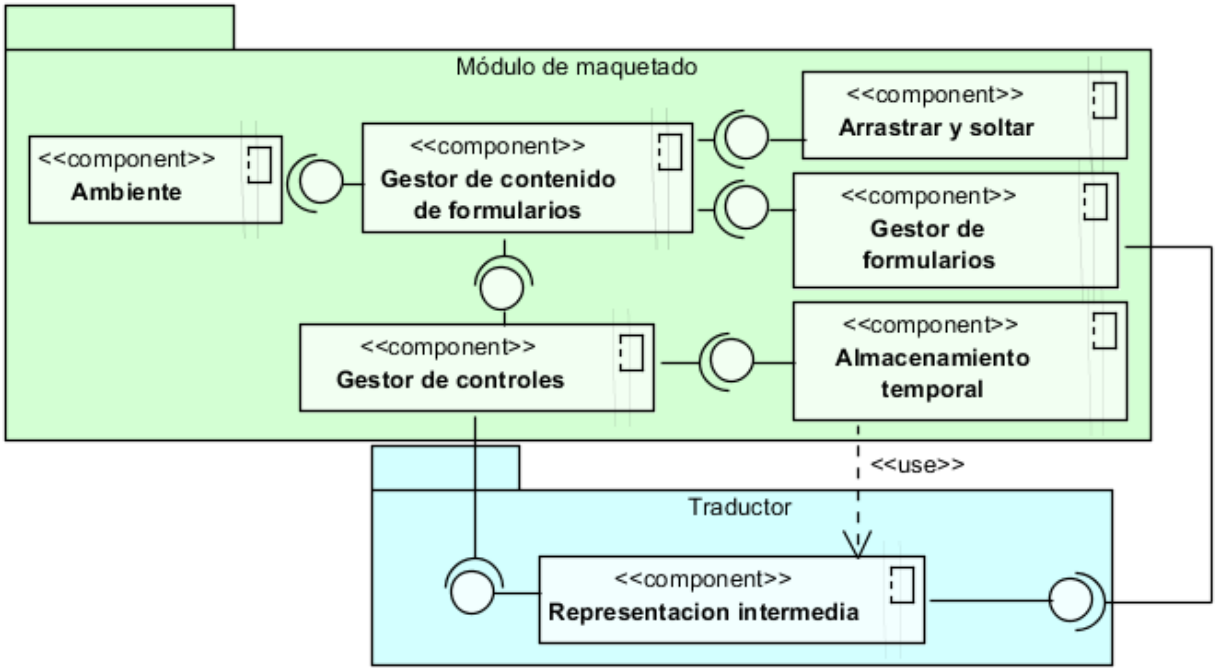

Fig. 3.3 Arquitectura del módulo de maquetado

En la Tabla 3.3 se describe brevemente la responsabilidad de cada uno de los componentes.

Tabla 3.3 Descripción de componentes de la arquitectura

Componente	Descripción
`Ambiente`	Responsable del pintado y control de inicialización del ambiente gráfico de las cuatro pantallas que soportan todo el proceso de ITOs_IBRAIN; también es responsable de configurar el funcionamiento de los botones en cada pantalla. 1. Pantalla de configuración 2. Pantalla de módulo de maquetado 3. Pantalla de resumen 4. Pantalla de resultados
`Drag and drop`	Responsable de realizar acciones de arrastrar y soltar componentes a través de la implementación de interfaces que soportan la funcionalidad de *Drag and Drop* de JavaFX.
`Almacenamiento temporal`	Responsable de conocer la información necesaria del posicionamiento de los controles dentro del *Grid* así como de

Componente	Descripción
	realizar una afectación puntual a la representación intermedia.
Gestor de formularios	Responsable del comportamiento de la lista de formularios y sus elementos, definir su representación visual, conocer los controles de cada uno de los formularios modelados en IFML y establecer la respuesta al clic sobre un elemento de la lista (formulario): el usuario define la configuración y las dimensiones que tendrá el formulario a maquetar por medio de un diálogo emergente o la presentación de los controles si ya estaba maquetado el formulario.
Gestor de controles	Responsable de darle comportamiento a la lista de controles, implementar el comportamiento de *Drag* en cada uno de los elementos en la lista (controles) y establecer la respuesta al clic derecho sobre los controles: mostrar un menú flotante.
Gestor de contenido de formularios	Inicializa la cuadrícula (*Grid*) de acuerdo a las medidas que define el usuario, determina cuáles celdas van a recibir controles y cuáles no, además de implementar en ciertas celdas la funcionalidad de *Drag and Drop.*

3.3.2 *Sprint* 1: Descripción del comportamiento esperado del módulo

Para detallar cada uno de los componentes que conforman la arquitectura se describió el comportamiento y las relaciones que iba a haber entre ellos mediante un diagrama de actividades (Fig. 3.4).

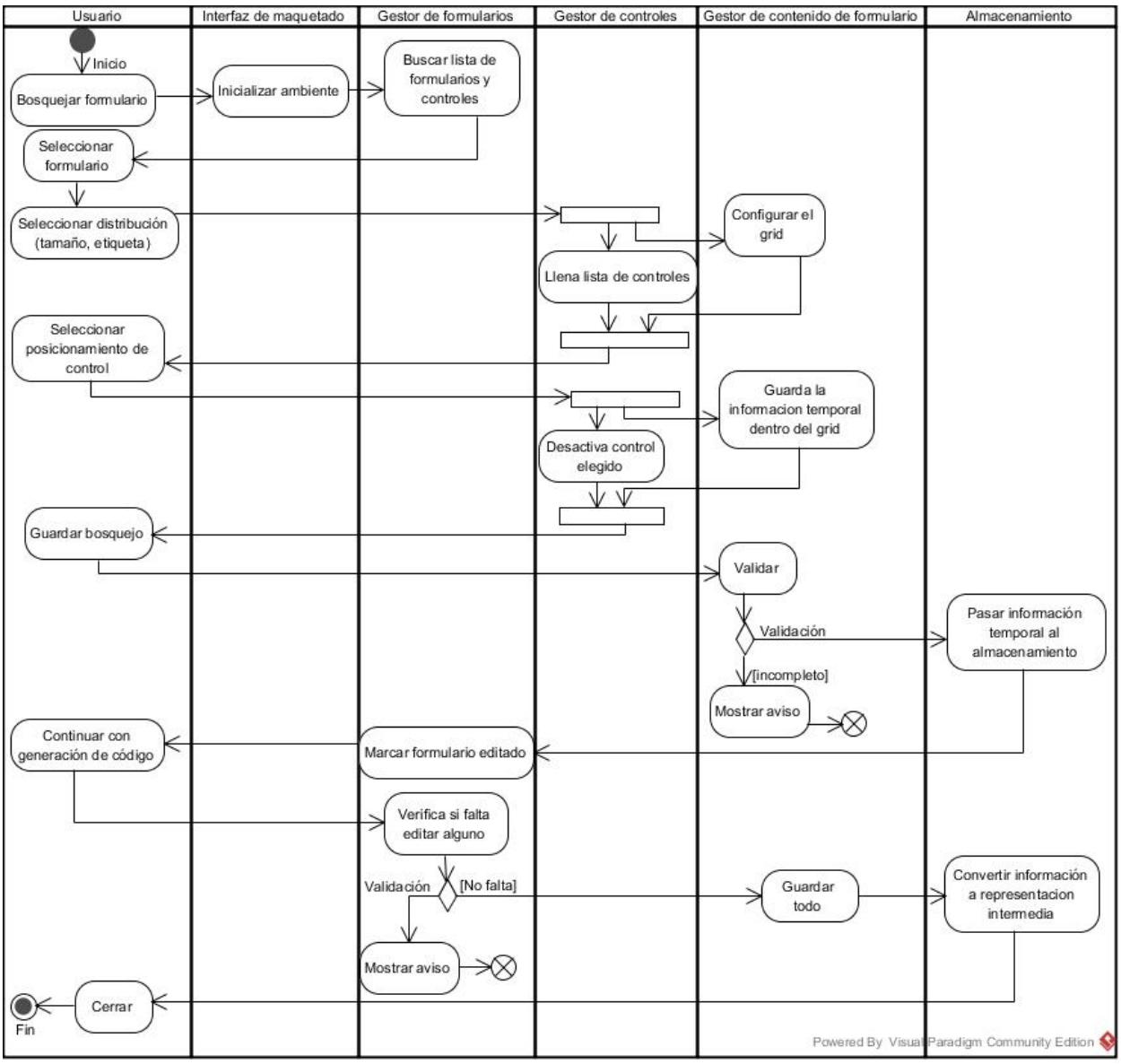

Fig. 3.4 Diagrama de actividades del módulo de maquetado

Este diagrama ayudó a detallar las funcionalidades ya identificadas, lo cual hizo más sencillo identificar elementos internos de los componentes de la arquitectura y delegar las responsabilidades a cada uno.

3.3.3 *Sprint* 2: Desarrollo de los componentes de la arquitectura

Cada componente de la arquitectura tiene un objetivo específico y están relacionados de tal manera que permita dar solución al desarrollo de requerimientos complejos que se necesitan para el funcionamiento del módulo y su integración a ITOs_IBRAIN.

3.3.3.1 Drag and Drop

Brinda el soporte a eventos de *Drag and Drop* con la tecnología JavaFX. Esta tecnología asigna directamente el comportamiento a los objetos que lo implementan, sin embargo el módulo requiere la asignación de esos eventos en diferentes tipos de objetos y momentos; como en la lista de controles disponibles que contiene el texto que identifica al control y en ciertas celdas del *Grid* que muestra imágenes y texto que representan a los controles y sus etiquetas. Por consiguiente, no se usó la asignación directa de este comportamiento a los objetos mencionados, sino que se crearon las interfaces `SourceDDI` y `TargetDDI`, la primera representa al origen de donde se toman los elementos para arrastrarlos (una lista o una celda) y la segunda representa al destino donde se sueltan los elementos (una celda).

También se diseñaron las clases específicas para listas que permiten arrastrar sus elementos (la lista de controles lo hace, pero la lista de formularios no) y para las celdas dentro del *Grid*, aunque no todas las celdas del *Grid* permiten recibir controles y soportar *Drag and Drop* (DD), esto lo determina la orientación de la etiqueta con respecto al control, si la posición es izquierda las columnas pares no permiten la funcionalidad de soltar controles pero si la posición es arriba, las filas pares son las que no permiten la funcionalidad de soltar controles; esto se determinó así por un requerimiento no funcional que dice que la asignación de controles dentro del *Grid* debe hacerse en dúo (control con etiqueta).

En la figura Fig. 3.5 se muestran las clases correspondientes al componente de *Drag and Drop*.

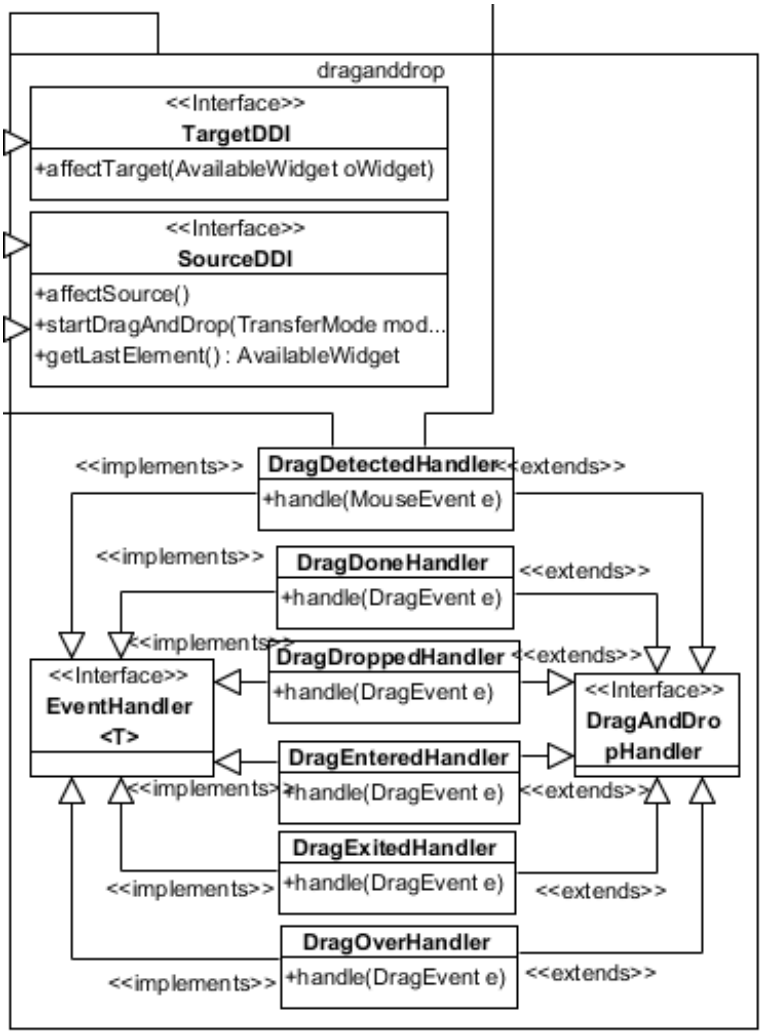

Fig. 3.5 Diagrama de clases del componente drag and drop

SourceDDI (Interfaz)

Representa el origen de la acción de arrastrar y soltar. Inicializa la funcionalidad de *Drag and drop* y determina el funcionamiento que tiene una celda ocupada o la lista de controles cuando un elemento fue arrastrado y soltado satisfactoriamente. También conoce el último elemento que fue soltado y arrastrado para brindar soporte a la funcionalidad de “Deshacer” (*Undo*).

TargetDDI (Interfaz)

Representa el destino de la acción de arrastrar y soltar. Permite detallar el funcionamiento que se tiene al soltar un control dentro de una celda si ésta se encuentra vacía.

DragDetectedHandler

Inicializa el ambiente para trabajar con la funcionalidad *Drag and Drop* en el origen (lista de controles o celda ocupada).

DragDoneHandler

Valida si el objeto arrastrado se aceptó en el destino (`TargetDDI`), de ser así afecta el espacio de origen (`SourceDDI`); por ejemplo, deshabilita la entrada de la lista o vacía la celda de origen junto con la celda asociada para la etiqueta.

DragDroppedHandler

Evalúa si el objeto arrastrado es aceptable; si lo acepta entonces afecta la celda destino (`TargetDDI`) y avisa al origen (`SourceDDI`) que fue aceptado.

DragEnteredHandler

Afecta la visualización del elemento para mostrar que llega a un área donde será aceptado.

DragExitedHandler

Afecta la visualización del elemento para mostrar que sale de un área donde era aceptado.

DragOverHandler

Afecta la visualización del elemento para mostrar que se encuentra sobre un área donde es aceptado.

3.3.3.2 Enviroment

Es el componente responsable de inicializar el ambiente de trabajo del Módulo y Generador, da soporte al manejo de múltiples escenas (pantallas) a lo largo de la ejecución de la aplicación a partir de una barra de herramientas (Fig. 3.6); esta barra es común a todas las pantallas y controla el avance entre las distintas fases con los botones *Back* y *Next*, como en un flujo de trabajo, además de presentar las operaciones particulares que soporta la pantalla de maquetado mediante botones que se habilitan y deshabilitan.

Fig. 3.6 Representación visual del ToolBar

A continuación se describen las clases correspondientes a este componente, en la Fig. 3.7 se muestra su diagrama de clases.

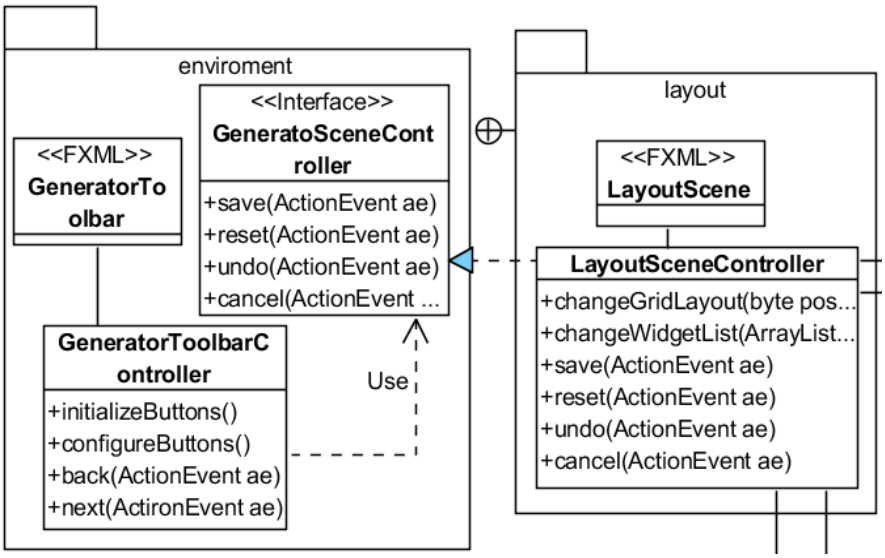

Fig. 3.7 Diagrama de clases del componente environment

GeneratorSceneController (Interfaz)

Determina los métodos para manipular los botones de la barra de acuerdo a las características particulares de la pantalla.

Generator ToolBar

Representación visual, en FXML, de los elementos que forman la barra de herramientas y sus botones: *Back, Next, Save, Reset, Undo, Cancel*.

Generator ToolBar Controller

Es el controlador de la barra de herramientas; conoce cuál es la pantalla actual (misma que implementa la interfaz `GeneratorSceneController`) y, a partir de ahí, determina cómo mostrar sus propios botones y cuál es la pantalla a mostrar al oprimir el botón *Back* o el botón *Next*.

Paquete `layout`

En JavaFX cada pantalla se separa en la parte visual, descrita en FXML, y la parte de control, descrita como una clase de Java con anotaciones para relacionarse con los elementos visuales, lo que resulta una implementación del patrón arquitectónico MVC (Modelo Vista Controlador). En este caso, cada pantalla y sus requerimientos particulares se encuentran en un paquete específico. En el paquete `layout`, se encuentra el FXML `LayoutScene` que contiene todos los elementos visuales que conforman al módulo de maquetado (Fig. 3.1**¡Error! No se encuentra el origen de la referencia.**), con etiquetas que hacen referencia a objetos de clases propias: `FormList`, `WidgetList` y `FrmGridLayout`.

Su controlador es `FormContentManager`, el cual implementa la interfaz `LayoutSceneController`. Este controlador tiene relación directa con las representaciones visuales de la lista de formularios disponibles (`FormList`), la lista de los controles disponibles (`WidgetList`) y del *Grid* (`FrmGridLayout`); además, es el encargado de inicializarlos una vez que ha obtenido la Representación Intermedia (RI). Y finalmente, determina la funcionalidad particular para los botones *Save, Reset, Undo* y *Cancel*, los cuales sólo aparecen en el módulo de maquetado.

3.3.3.3 Form Content Manager

Es el componente responsable del comportamiento del *Grid* donde se realiza el maquetado de los formularios. En la Fig. 3.8 se muestra el diagrama de clases.

GridLayoutCell

Representa a cada celda dentro del *Grid* e implementa las interfaces `SourceDDI` y `TargetDDI`.

FrmGridLayout

Clase que extiende del elemento propio de JavaFx `GridPane` y es la encargada de tener la representación visual del *Grid*. Se encarga de inicializar la cuadrícula, instancía y configura celdas dependiendo de la selección del usuario (número de columnas, número de renglones, posición esperada de las etiquetas de los controles); además

determina cuáles celdas del *Grid* permiten recibir controles (*Target*) y cuáles no dependiendo de la orientación de la etiqueta (izquierda o arriba).

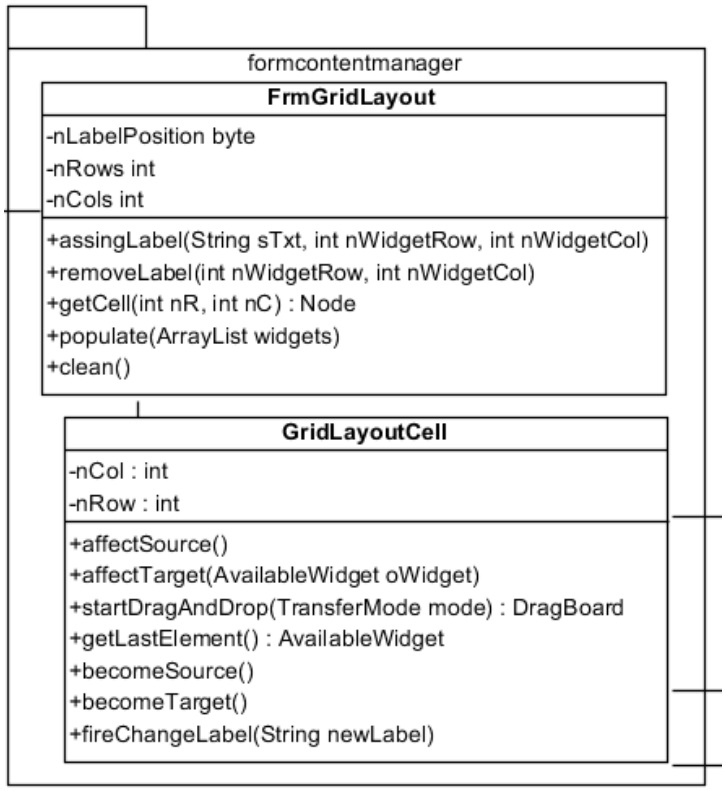

Fig. 3.8 Diagrama de clases del componente formcontentmanager

3.3.3.4 Form manager

Es el responsable del comportamiento que tiene la lista de formularios disponibles, de acuerdo al modelo IFML original, así como de su representación visual en la escena. En la Fig. 3.9 se muestra el diagrama de clases correspondiente.

FormList

Clase de la representación visual de la lista de formularios disponibles.

AvailableForm

Clase para la representación interna de un formulario y sus características para efectos del maquetado; por ejemplo, si ya fue maquetado o no, el número de columnas y renglones indicado por el usuario así como la posición esperada de la etiqueta entre

otras características; los métodos permiten establecer la configuración del formulario, guardar en el almacenamiento temporal el formulario y reiniciar la información de los formularios (almacenados o no).

ChangeListenerHandler

Clase que maneja el evento de cambio de selección sobre la lista de formularios.

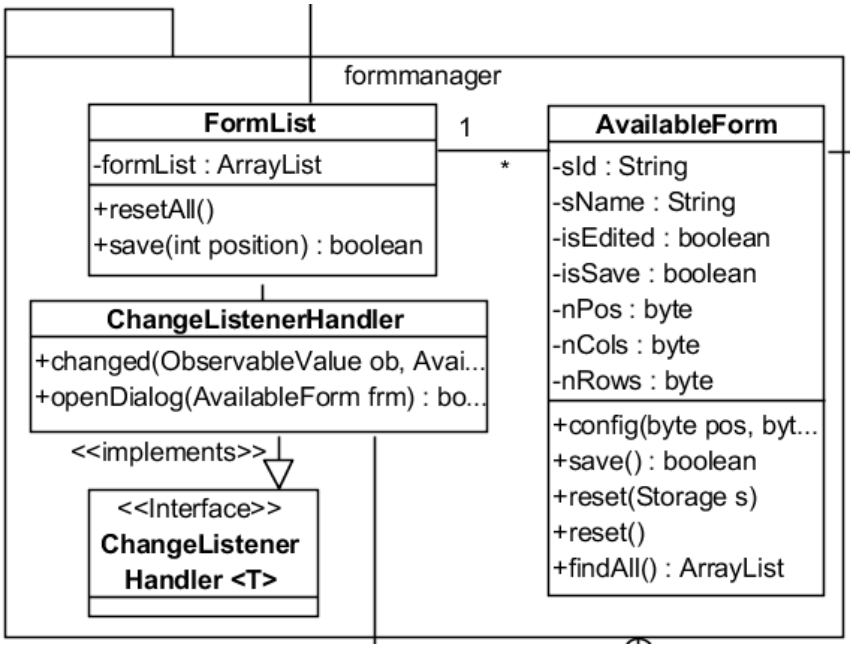

Fig. 3.9 Diagrama de clases del componente formmanager

3.3.3.5 Widget manager

Componente responsable del comportamiento que tiene la lista de controles disponibles de acuerdo al formulario seleccionado, establece la funcionalidad de arrastrar los controles desde la lista hasta el *Grid* y su representación visual en la escena. En la Fig. 3.10 se muestra el diagrama de clases del componente.

WidgetList

Clase que representa el comportamiento de la lista de controles y que implementa la interfaz `SourceDDI` para soportar el comportamiento de arrastrar los controles de la lista hacia el *Grid*.

AvailableWidget

Clase para la representación interna de un control para efectos del maquetado, por ejemplo la imagen de su representación visual, el tipo de control y el posicionamiento que tiene dentro del *Grid*. Además asocia los eventos a los que responde el control.

ContextMenuRequestedHandler

Clase que maneja el evento de clic derecho sobre un control y que presenta un menú flotante para configurar dicho control.

WidgetSettingsMenu

Clase para el menú flotante con las opciones que permiten cambiar el texto de una etiqueta o cambiar la representación visual de un control.

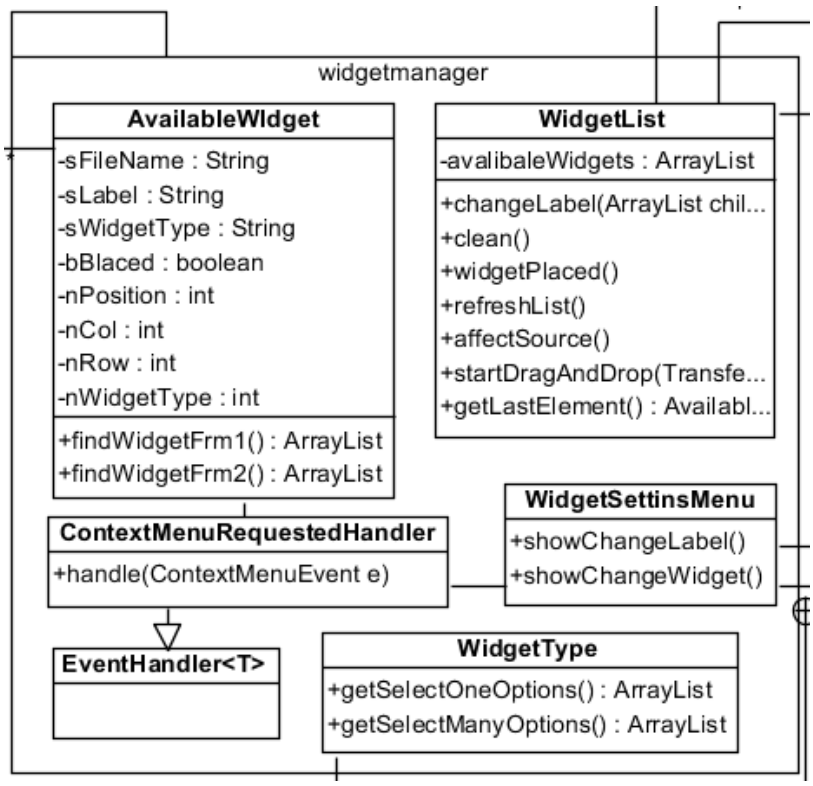

Fig. 3.10 Diagrama de clases del componente widgetmanager

3.3.3.6 Storage

Es un almacenamiento temporal que conoce la información necesaria del posicionamiento de los controles dentro del *Grid* por cada formulario guardado y es

capaz de realizar la afectación a la Representación Intermedia (RI). La RI es una estructura jerárquica que forma el Generador, a partir de los modelos IFML y UML, y que contiene los elementos a construir de la aplicación de forma abstracta, es decir, independiente del lenguaje de salida.

Storage

El paquete y la clase mostrados en la Fig. 3.11 corresponden al concepto de almacenamiento interno de maquetado. Esto es, un espacio especial donde se guardan los formularios una vez maquetados si el usuario da clic en el botón *Save*; este espacio es el único que se relaciona con la fase de Representación Intermedia del Generador.

El almacenamiento local es necesario para evitar afectaciones indeseables sobre la Representación Intermedia debido a que ésta es la fuente para la generación de código final; solo cuando el usuario indica que desea generar dicho código (botón *Next*) se refleja el almacenamiento local en los elementos necesarios de la Representación Intermedia para continuar con el proceso original del Generador.

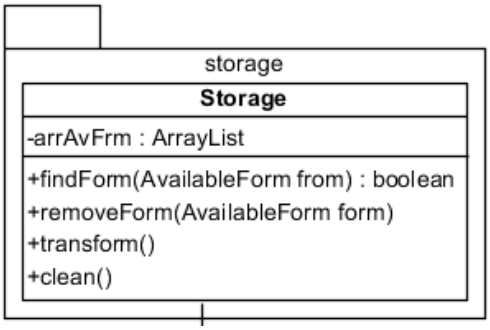

Fig. 3.11 Diagrama de clases del componente storage

3.3.4 *Sprint* 3: Integración del Módulo de maquetado con el Generador de Aplicaciones Enriquecidas

Una vez construidas las funcionalidades propias del Módulo y probadas con datos fijos (formularios y controles) se pasó a la integración del Módulo con el generador ITOs_IBRAIN. El proceso normal del Generador consta de los siguientes pasos:

1. Fase de análisis léxico y gramatical

2. Fase de análisis semántico
3. Fase de representación intermedia
4. Fase de creación de código

El proceso del módulo se agregó entre el paso 3 y 4 quedando de la siguiente manera:

1. Fase de análisis léxico y gramatical
2. Fase de análisis semántico
3. Fase de representación intermedia
4. **Obtención de RI**
5. **Módulo de maquetado**
6. **Afectación de RI**
7. Fase de creación de código

En la Fig. 3.12 se muestra el diagrama de paquetes que se modeló, resultante de la integración de las dos aplicaciones en un solo proyecto.

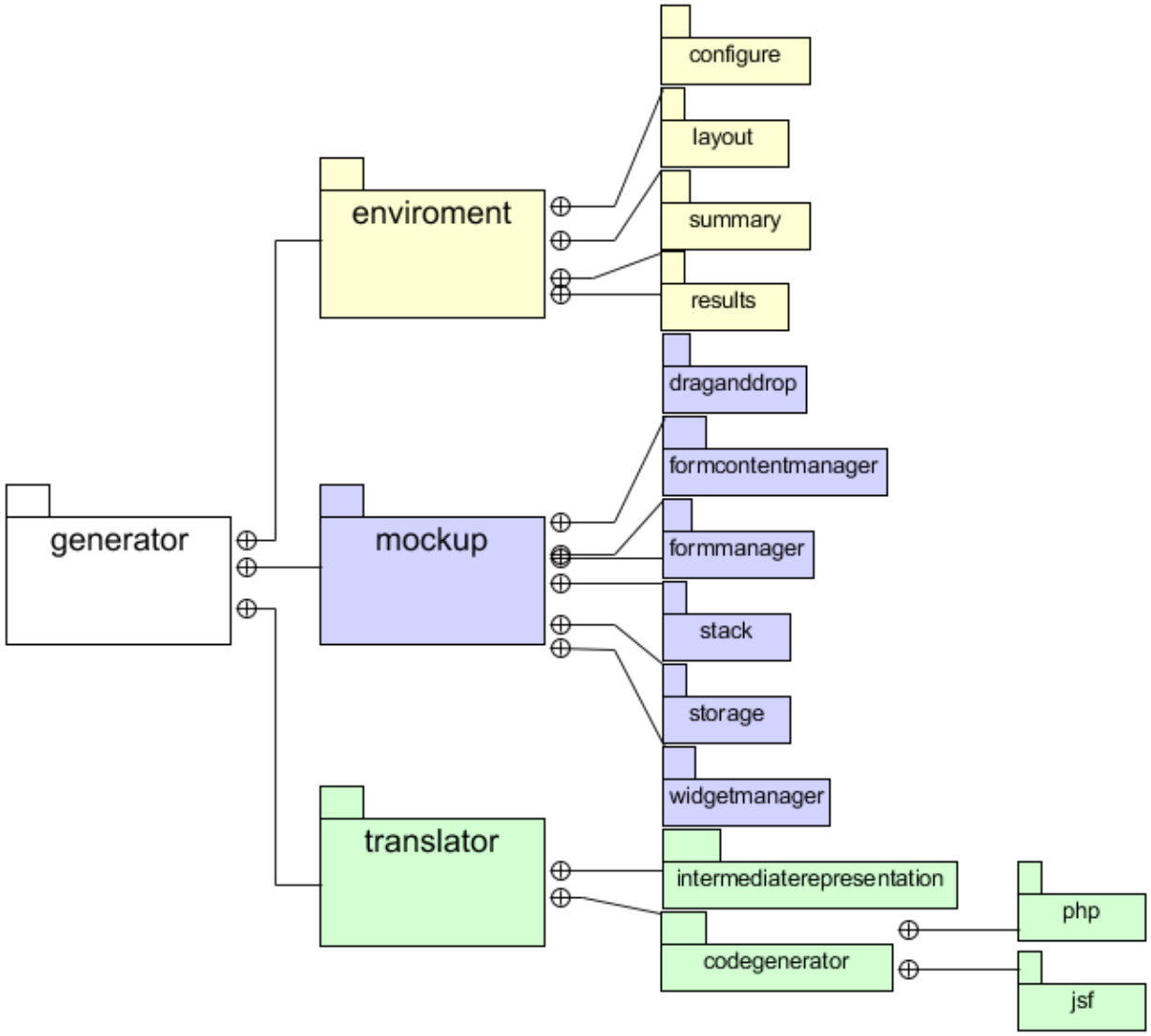

Fig. 3.12 Diagrama de paquetes general de la integración

Como se aprecia en la Fig. 3.12, la ITOs_IBRAIN se dividió en tres paquetes diferentes, el primero llamado `environment`, contiene a su vez cuatro paquetes para cada una de las pantallas (`configure`, `layout`, `summary` y `results`) que determinan la parte visual de la aplicación. Las clases de cada uno de esos paquetes siguen el patrón arquitectónico MVC, donde se encuentra la vista (archivo `FXML`) y el controlador de cada una de las pantallas en su paquete correspondiente.

En el paquete `mockup` se encuentra toda la funcionalidad correspondiente al Módulo de maquetado; está separada en seis paquetes que establecen la separación de los componentes diseñada en la arquitectura.

Finalmente, el paquete `translator`, contiene todas las clases que realizan la generación de código, el cual a su vez se divide en más paquetes para separar los componentes que la conforman, en la Fig. 3.12 se muestran sólo los paquetes de la RI y `codegenerator` ya que en esos paquetes se encuentran las clases a las que se les realizaron modificaciones; estas modificaciones serán descritas con mayor detalle a continuación.

De esta manera, las dos aplicaciones quedaron integradas en una sola, aunque aún sin los enlaces necesarios para la obtención ni la afectación a la RI de ITOs_IBRAIN.

3.3.5 *Sprint* 4: Obtención de la RI del Generador de Aplicaciones Enriquecidas

Para preparar el ambiente y que el Generador fuera capaz de pasarle la información de los formularios y sus controles al Módulo de Maquetado fue necesario afectar ciertas clases de la RI. Estos cambios se realizaron en dos etapas.

Etapa 1

La RI es una estructura de datos abstracta con forma de árbol jerárquico, donde cada uno de los nodos representa elementos que contiene la Aplicación Web modelada en IFML. Los nodos importantes para el Módulo de maquetado son dos; el primero corresponde a la representación de un formulario (clase `IR_Form`) y el segundo corresponde a la representación de los controles internos (clase `IR_Widget`).

En el nodo raíz de la aplicación, representado por un objeto de clase `IR_Application`, se agregó un método `findForms`, que es capaz de recorrer sus nodos, encontrar a los que representan formularios, almacenarlos en un arreglo y retornarlo.

Etapa 2

El controlador del `ToolBar` invoca al método `findForms` para obtener un arreglo de objetos con los formularios encontrados, y los pasa a la clase que representa la lista de formularios (`FormList`) para inicializar los valores de cada formulario encontrado junto con sus controles respectivamente.

3.3.6 *Sprint* 5: Afectación de la RI del Generador de Aplicaciones Enriquecidas

Para la afectación de código, también se realizaron pasos previos antes de afectar directamente la RI. Esto por un requerimiento no funcional que dice: “Si un usuario no define una distribución específica para un formulario, entonces el módulo le asignará una por omisión: la etiqueta irá a la izquierda con respecto al campo de captura y la distribución del *Grid* será en dos columnas de forma vertical”. Todos estos cambios se realizaron en tres etapas:

Etapa 1

En el controlador de la pantalla del Módulo de maquetado se agregó el método `completeRI` que completa la información necesaria para afectar la RI en caso de que el usuario no haya maquetado algún formulario en el módulo.

Ahí se recorre el `Storage` (almacenamiento temporal) y se comparan los formularios que estén almacenados contra la lista de formularios original y, si existe un formulario sin distribución asignada, entonces al formulario se le asigna la distribución por omisión y se almacena en el `Storage` y finalmente se ejecuta el método `transform`.

Etapa 2

Para que el método `transform` funcionara, se agregaron atributos y métodos a las clases de la Representación Intermedia `IR_Form` e `IR_Widget` para que fueran capaces de recibir los datos almacenados en el `Sorage` proveniente del módulo de maquetado, en la Tabla 3.4 se enlistan los atributos y métodos agregados a estas clases.

Tabla 3.4 Atributos y métodos agregados en las clases IR_Form e IR_Widget

IR_Form	
nPos	Atributo que almacena la orientación de la etiqueta con respecto al campo de captura
nRows	Atributo que almacena el número de filas que tendrá la distribución

nCols	Atributo que almacena el número de columnas que tendrá la distribución
sortWidgets	Método que ordena los controles del formulario con respecto a la distribución que tengan en el *Grid*.
IR_Widget	
nCol	Atributo que almacena en qué columna de la distribución se posiciona ese control
nRow	Atributo que almacena en qué fila de la distribución se posiciona ese control
sMockupType	Atributo que almacena la representación visual que tiene el control para el caso de controles de selección simple o múltiple

Una vez realizados los ajustes anteriores, se terminó el método `transform`, que se encarga de obtener todos los formularios que estén en el almacenamiento temporal y reflejar su información en la RI, al enviar los siguientes datos a la clase `IR_Form`: la posición de la etiqueta, el número de columnas y renglones de la distribución y por cada control que tenga el formulario envía a `IR_Widget`: la columna y el renglón donde fue colocado, el texto de la etiqueta del control y el tipo de control asignado.

Etapa 3

Con el fin de que los controles de cada formulario estuvieran en el orden en el que aparecen en el Grid, de arriba abajo y de izquierda a derecha, y así facilitar la escritura de código en la generación de la aplicación resultante, en `IR_Form` se agregó el método `sortWidgets`, este método se ejecuta después del método `transform` y obtiene todos los controles asociados a un formulario y ordena esos controles en función al posicionamiento que tengan en las variables `nCol` y `nRow`; esto también incluyen aquellas celdas que se dejaron vacías, si el método encuentra que en el posicionamiento hay una celda sin control asignado, agrega un nuevo nodo de `IR_Widget` al formulario pero con un tipo de control denominado como “`NullField`”, este tipo de control representa una celda vacía en el *Grid*.

3.3.7 *Sprint* 6: Modificación de la generación de código para establecer las distribuciones maquetadas

El *Sprint* número seis se centró en identificar las clases que generan formularios en las tecnologías de salida: JSF con PF y PHP con jQuery y analizar las secciones de código donde se realiza la generación de controles y botones.

Una vez realizado lo anterior, se procedió a modificar y/o agregar métodos y variables que fuesen claves para que el código resultante de los formularios diera como resultado un diseño visual equivalente al realizado en el módulo de maquetado.

Como el proceso de generación se implementó por medio del patrón arquitectónico *Abstract Factory*, lo primero que se modificó fueron las clases enlistadas en la Tabla 3.5.

Tabla 3.5 Descripción de las clases abstractas modificadas para la generación de código

Clase abstracta modificada	Descripción
`Abstract_Form_CodeGenerator`	Se agregó una variable de tipo cadena para definir la sangría de los formularios y el método `iterateFormChildren` que permite recorrer cada uno de los hijos de un formulario (controles o ligas/botones) y crearlos. Este método recibe, entre otras cosas, la posición de la etiqueta y el número de columnas con que cuenta el formulario.
`Abstract_Widget_CodeGenerator` **`Abstract_Link_CodeGenerator`**	Se agregaron dos variables de tipo `byte` para almacenar el posicionamiento de la etiqueta y el número de columnas que tiene la distribución del formulario y se agregaron cuatro variables más, de tipo cadena, relativas a la sangría del código. Y al método encargado de crear el control

	(`createWidget` y `createLink` según corresponda) se le agregaron dos parámetros de entrada más, uno que recibe la posición de la etiqueta y otro que corresponde al número de columnas del formulario.

3.3.7.1 Generación de Java ServerFaces con PrimeFaces

Las modificaciones que se realizaron para afectar la generación de código en JSF y PF se hicieron en las clases relacionadas con formularios, controles y ligas/botones del paquete: `tesis.generator.translator.codegenerator.jsf`.

Clase: JSF_Form_Code Generator

El método `iterateFormChildren`, heredado de la clase abstracta definida en el apartado anterior, es el encargado de crear los elementos anidados dentro del formulario, principalmente controles y botones o ligas, por lo que tiene que informar a los hijos en qué posición va la etiqueta asociada.

Además, en el método que genera el contenido del `XHTML`, se agregó la etiqueta `Panel Grid`, esta etiqueta es la que asegura que el formulario generado sea responsivo, ya que ajusta su contenido al tamaño de la pantalla. Adicionalmente, se agregaron dos atributos más a esa etiqueta, el primero fue el atributo `columns` con el número de columnas definido por el usuario y el segundo fue el atributo `layout` de tipo `Grid`. Esto con el fin de asegurar que la distribución que el usuario haya realizado en el maquetado sea responsiva.

Clase: JSF_Widget_CodeGenerator

Se agregó la definición de sangría para las etiquetas de los controles, se agregaron dos parámetros más al método que permite crear un nuevo tipo de control y en la elección (`switch`) que determina qué control crear, dependiendo del tipo que tenga asignado, se agregó el tipo `NullField` (celda vacía).

Para los tipos de control `NullField`, `Field`, `SelectionField` *y* `MultiSelectionField` se invoca al método `generateRowsColums` que permite ir definiendo la distribución que tendrán los controles con base en la orientación que tenga la etiqueta que el usuario haya elegido (arriba, izquierda). Este método se encarga de realizar la distribución correspondiente para cada control en el formulario, esta distribución se realiza a través de las etiquetas `p:colum` y `p:row`.

Por otra parte, en la generación de los tipos de control `SelectionField` *y* `MultiSelectionField` se realizaron algunas modificaciones ya que, en la primera versión de ITOs_IBRAIN, para un control de tipo selección simple se generaba únicamente una lista desplegable (`SelectOneMenu`) y para la selección múltiple se generaba una lista de cajas de selección (`SelectManyCheckbox`), mientras que en esta versión se soportan otros controles equivalentes.

Para el tipo de control `SelectionField`, se verifica si el tipo de control es “selección simple” entonces la representación visual elegida por el usuario pudo ser: 1) Lista de Radios (`SelectOneRadio`), 2) Lista de selección simple (`SelectOneList`) o 3) Lista desplegable (`SelectOneMenu`); por el contrario, si el tipo de control es “selección múltiple”, entonces su representación visual elegida pudo ser: 1) Lista de selección múltiple (`SelectManyList`) o 2) Lista de cajas de selección (`SelectManyCheckbox`). Cualquiera que sea el control de selección, se invoca al método `generateRowsColums` para definir su distribución en el formulario.

Además, se agregó el método `generateLabel` que asigna al atributo `value` de la etiqueta `p:outputlabel` el texto de etiqueta asignado durante el maquetado.

Clase: JSF_Link_Code Generator

Los cambios fueron similares a los realizados en la clase `JSF_Widget_CodeGenerator`, la diferencia es el tipo de control que esta clase representa en la generación: liga de navegación, botón para abrir un diálogo emergente, botones para las operaciones de registro, modificación o borrado asociados al elemento IFML “`multipleForm`” y, finalmente, botón que está asociado directamente a una acción de un formulario, es a la generación de este último elemento al cual se le

realizaron los cambios ya que es el único que se muestra en el maquetado. Al igual que en la clase `JSF_Widget_CodeGenerator`, se agregó el método `generateRowsColums` que define la posición del botón correspondiente; además, también se agregó el método `generateLabel` que escribe la etiqueta `outputLabel` y define el atributo `value` con el nombre que tenga el botón.

Clase: JSF_ValidationRule_CodeGenerator

Esta clase es la encargada de asignar las validaciones asociadas a un control, a la cual sólo se agregó una variable nueva llamada `sHtmlValidator`, esta variable almacena la etiqueta o conjunto de etiquetas que realizan la validación al control asociado, una vez con el validador, esa variable se asocia a un objeto y se retorna a la clase que la invocó (`JSF_Widget_CodeGenerator`).

3.3.7.2 PHP con jQuery

Las modificaciones para afectar la generación de código en PHP y jQuery se realizaron en clases del paquete `tesis.generator.translator.codegenerator.php`. Las clases relevantes que se modificaron para la generación de código pertinente al módulo de maquetado están enlistadas a continuación y se describen, a grandes rasgos, las afectaciones realizadas en cada una de ellas.

Clase: PHP Form Code Generator

Se agregó la definición de la sangría para la división HTML (`DIV`) que contiene a la etiqueta `Form` y se agregaron dos parámetros más a la invocación de los objetos de las clases `PHP_Widget_CodeGenerator` y `PHP_Link_CodeGenerator`, estos fueron: el parámetro que indica la orientación que tendrá la etiqueta con respecto al control y el que define cuántas columnas tiene la distribución del *Grid.*

El método que genera el contenido HTML del formulario se modificó ya que, en la primera versión, la distribución del contenido de un formulario se realizaba mediante tablas de HTML (`table`) que no son responsivas. Para asegurar el comportamiento responsivo se sustituyó la distribución mediante tablas por una mediante divisiones (`DIV`).

Clase: PHP_Widget_CodeGenerator

Se agregaron dos parámetros más al método que permite crear un nuevo tipo de control, se realizó la definición de la sangría que llevan los controles, las columnas y las filas del código en HTML para la distribución y en la selección que determina qué tipo de control crear, se agregó el tipo `NullField`.

Para los tipos de control `NullField`, `Field`, `SelectionField` *y* `MultiSelectionField` se incluyó la llamada al método `generateRowsColums` que permite definir la distribución que tendrán los controles con base en la orientación que tenga la etiqueta que el usuario haya elegido (arriba, izquierda) . Este método se encarga de realizar la distribución para cada control que encuentra en el formulario a través de divisiones HTML (`DIV`) asignando a cada uno de ellos, en su atributo `class`, un estilo especial dependiendo del número de columnas que tiene ese formulario y la existencia o inexistencia de listas o áreas de texto en la misma línea (estos controles tienen un alto mayor al del resto). El estilo asociado a la clase se encuentra definido en el archivo de hoja de estilo `responsive.css` que asegura que los formularios de la aplicación resultante sean responsivos.

Igual que en el caso de la combinación de lenguajes JSF + PF, fue necesario modificar la generación de controles de selección. Para controles de selección simple se tienen 1) Lista de Radios (campos de tipo `radio`), 2) Lista de selección simple (campo de tipo `select`) o 3) Lista desplegable (campo de tipo `select` con atributo `size=3`); para controles de selección múltiple: 1) Lista de selección múltiple (campo de tipo `select` con atributos `multiple` y `size=3`) o 2) Lista de cajas de selección (campos de tipo `Checkbox`). Cualquiera que sea el control de selección, se invoca al método `generateRowsColums` para definir su distribución en el formulario.

Además, se agregó un método `generateLabel` que escribe el texto que va a llevar la etiqueta `Label` de acuerdo al maquetado realizado.

Clase: PHP_Link_CodeGenerator

Los cambios fueron similares a los realizados en la clase `PHP_Widget_CodeGenerator`, la diferencia es el tipo de control que esta clase

representa en la generación: liga de navegación, botón para abrir un diálogo emergente, botones para las operaciones de registro, modificación o borrado asociados al elemento IFML "`multipleForm`" y, finalmente, botón que está asociado directamente a una acción de un formulario, es a la generación de este último elemento al cual se le realizaron los cambios ya que es el único que se muestra en el maquetado. Al igual que en la clase `PHP_Widget_CodeGenerator`, se agregó el método `generateRowsColums` que define la posición del botón correspondiente.

Clase: PHP_ValidationRule_CodeGenerator

Esta clase es la encargada de asignar las validaciones asociadas a un control, a la cual sólo se agregó una variable nueva llamada `sHtmlValidator`, esta variable almacena la etiqueta o conjunto de etiquetas que realizan la validación al control asociado, una vez con el validador, esa variable se asocia a un objeto y se retorna a la clase que la invocó (`PHP_Widget_CodeGenerator`).

Clase: PHP_Enviroment_CodeGenerator

Para asegurar que los formularios definidos en PHP fueran responsivos en la generación se crea un nuevo archivo llamado `responsive.css` que contiene medidas de porcentajes ya establecidos que están en función del número de columnas que el usuario haya definido en la configuración para el formulario a generar. Este archivo se almacena en el componente `resources.php` y, cuando se crea la aplicación resultante, esta clase se encarga de escribir ese archivo en la carpeta CSS para asegurar que todos los formularios de la aplicación sean responsivos.

Clase: PHP_Site_CodeGenerator

En esta clase sólo se agregó en el encabezado la etiqueta `Link` con el atributo `href` que hace referencia al archivo de estilo `responsive.css` para que la aplicación resultante reconozca al archivo que permite hacer responsivos los formularios.

3.4 Pruebas de validación de la integración del Módulo al Generador

Se realizaron las siguientes pruebas para validar que los cambios realizados a la generación de código funcionaban correctamente y que lo modelado en el módulo se veía realmente reflejado en los formularios de las RIAs generadas.

Validación de distribución de formularios

El primer modelo en IFML, ilustrado en la Fig. 3.13, se realizó para validar la distribución, el formulario `Persona` cuenta con diez controles de captura, cuatro controles de selección simple, uno de selección múltiple y un botón.

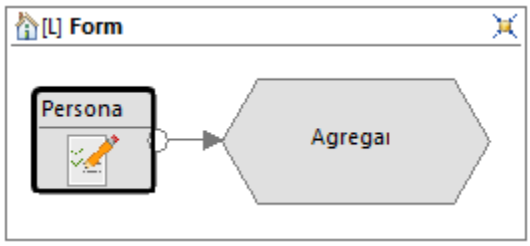

Fig. 3.13 Modelo en IFML de un formulario asociado a un Action

La configuración asignada para la distribución de este formulario en el módulo de maquetado fue 4 columnas por 8 filas y la etiqueta arriba del control como se presenta en la Fig. 3.14; en este caso, la representación visual del control `Cargo` se cambió por Radios, los controles `Estado`, `Ciudad` y `País` se cambiaron por listas de selección simple y el control `Pasa tiempos` por una lista de selección múltiple. Además, el texto de las etiquetas de algunos controles fue cambiado ya que necesitaban caracteres especiales como acentos y signos de interrogación que no se colocan en el modelado.

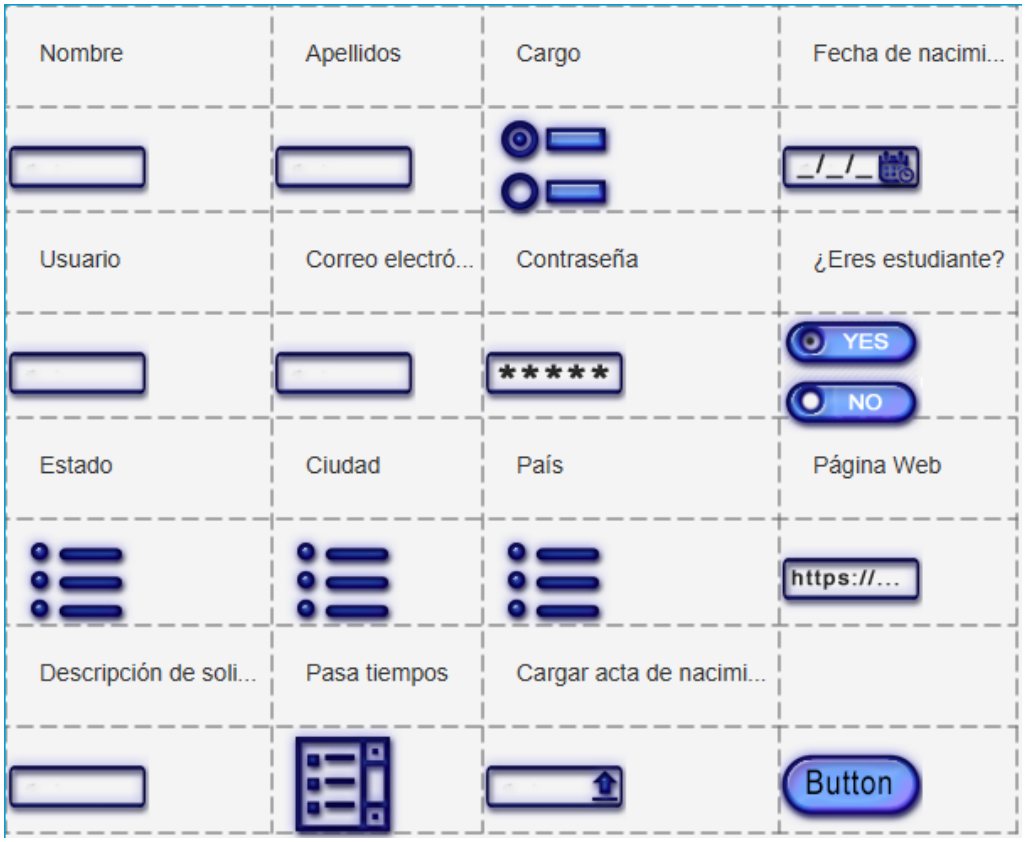

Fig. 3.14 Área de maquetado con controles distribuidos del formulario Persona

En la Fig. 3.15 se muestra el mismo formulario `Persona`, resultante de la generación, donde se comprueba la maquetación; cuenta con la distribución de controles así como con los cambios de representación visual y de texto de etiquetas tal como se indicó en el maquetado.

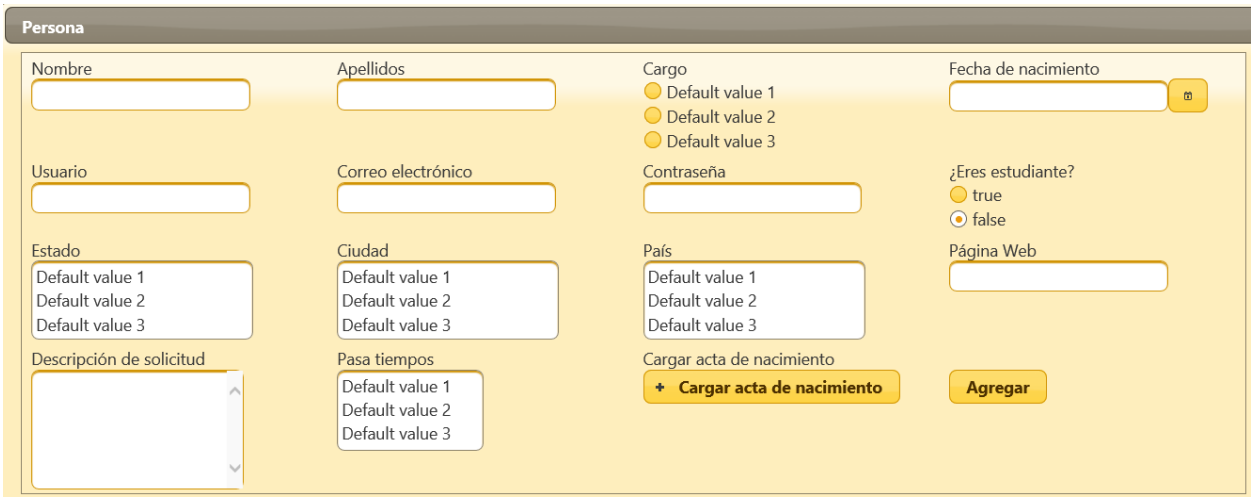

Fig. 3.15 Distribución del formulario Persona obtenido del Generador

Validación de distribución de formularios con validaciones

El segundo modelo, ilustrado en la Fig. 3.16, valida la distribución de controles que tienen asociadas validaciones.

Fig. 3.16 Modelo en IFML de un formulario con validaciones asociado a un Action

Las validaciones de los campos del formulario `Validaciones` se muestran en la Fig. 3.17.

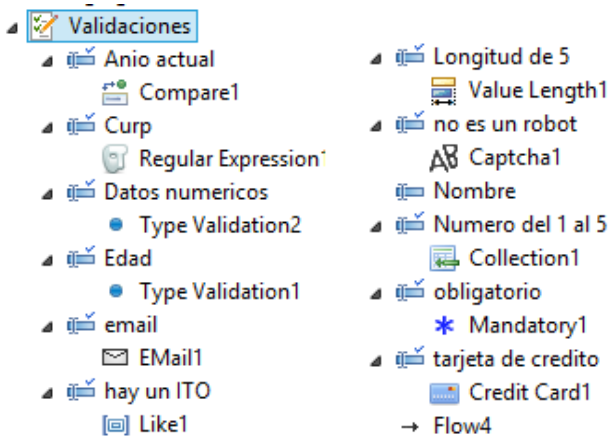

Fig. 3.17 Validaciones de los campos del formulario Validaciones

En la Fig. 3.18 se muestra la distribución de los controles realizada para este formulario con validaciones, con la configuración siguiente: 4 columnas por 7 filas y la etiqueta a la izquierda con respecto al control, además algunos textos de las etiquetas fueron modificados con letras acentuadas o se colocaron caracteres especiales.

Fig. 3.18 Área de maquetado con controles distribuidos del formulario Validaciones

En el módulo de maquetado no se hace alguna diferencia visual para indicar que ese campo contiene una validación; sin embargo si el campo de captura tiene asociado un validador desde IFML, el generador hace los ajustes necesarios para que el código resultante tenga la distribución que el usuario haya realizado y se mantenga responsivo. En la Fig. 3.19 se muestra el formulario `Validaciones`, resultante de la generación, el cual tiene correspondencia con lo que se maquetó en el módulo (Fig. 3.18) y además, en la parte inferior, se muestra que efectivamente contiene las validaciones que se han asignado a cada uno de los campos del formulario citado.

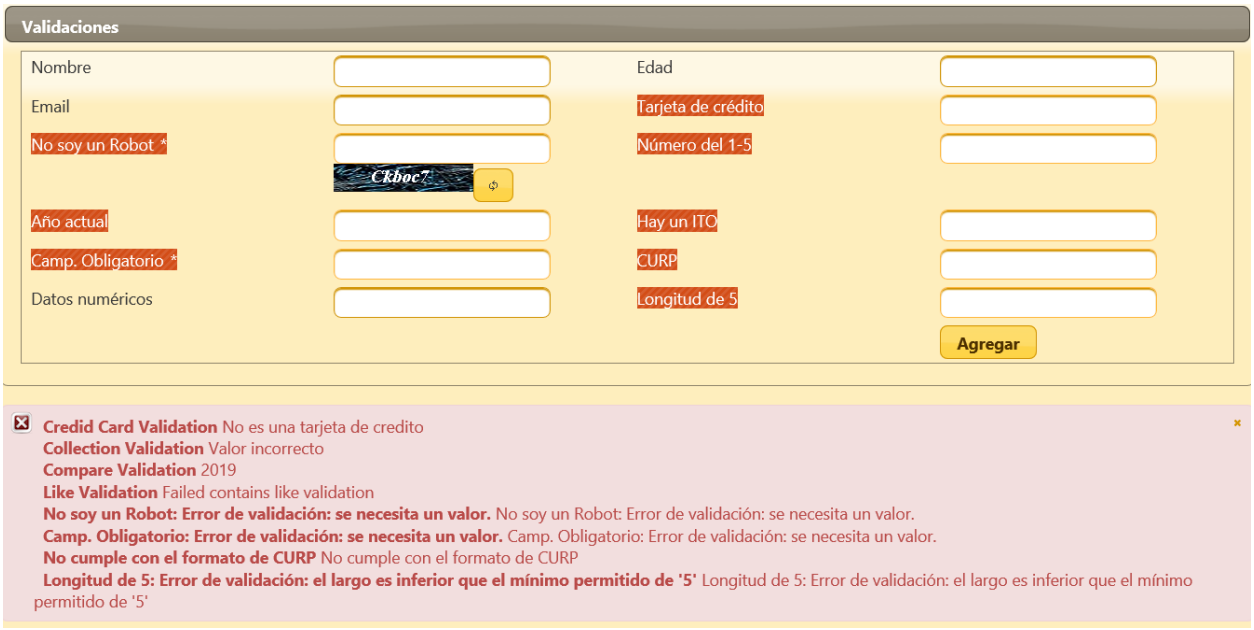

Fig. 3.19 Distribución del formulario Validaciones obtenido del Generador

Capítulo 4 Resultados

Para comprobar que el Módulo de Maquetado integrado a ITOs_IBRAIN tiene un impacto positivo en el diseño visual de los formularios resultantes y que con esa implementación el usuario prescinde de invertir tiempo en realizar las distribuciones de los controles desde el código fuente de las vistas de la aplicación, se puso a prueba el generador en diversos casos de estudio.

4.1 Caso de estudio FastRent

El caso de estudio que se presenta en este documento de tesis es el mismo que se usó en la tesis que desarrolló la primera fase del Generador de Aplicaciones Enriquecidas, el cual se llama *FastRent*, aunque este caso de estudio cuenta con la mayoría de los elementos en IFML que son afectados por el Generador, para este caso sólo se tomaran en cuenta los modelos, el diseño de los maquetados y páginas de la aplicación generada que cuenten con formularios.

4.1.1 Planteamiento caso de estudio 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.

4.1.2 Modelado en IFML del caso de estudio FastRent en WebRatio Web Platform

Después de identificar los requerimientos funcionales de la aplicación *FastRent*, se realizó el modelo de dominio (Fig. 4.1) y el modelo navegacional en IFML (Fig. 4.2) con la herramienta *WebRatio Web Platform.*

4.1.2.1 Modelo de Dominio del caso de estudio *FastRent*

En la Fig. 4.1 se muestra el modelo de dominio realizado para la aplicación a generar *FastRent*, este modelo contiene cuatro entidades de negocio, mismas que se convertirán en clases de modelo en la aplicación resultante.

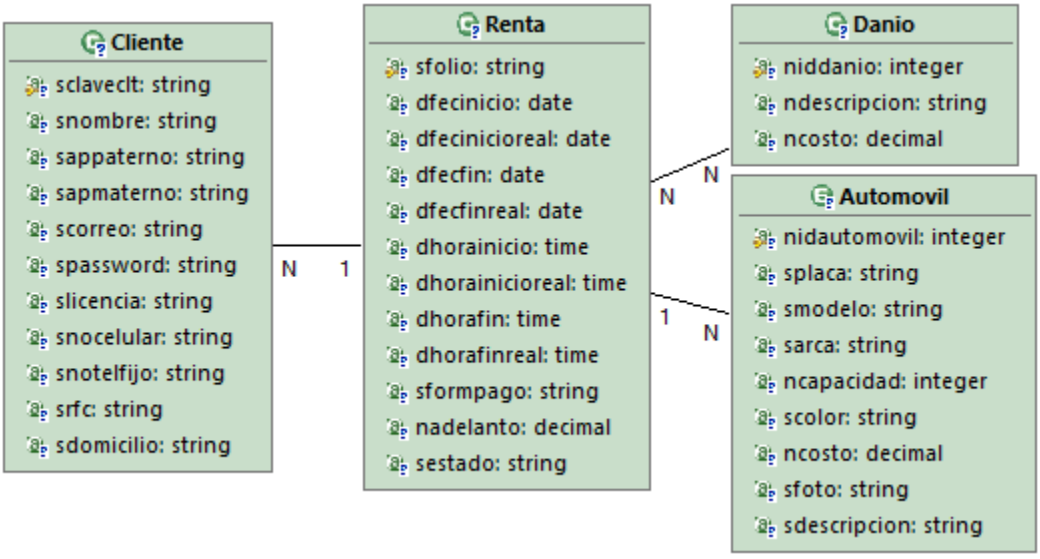

Fig. 4.1 Modelo de dominio de la aplicación FastRent realizado en WebRatio Web Plataform

4.1.2.2 Modelo Navegacional del caso de estudio FastRent

El modelo navegacional de la Fig. 4.2, cuenta con una página inicial (`home`) llamada `Principal` que contiene un mensaje de bienvenida y un botón que despliega un diálogo emergente llamado `Contacto`, mismo que tiene un formulario con validaciones.

Desde la página `Principal`, es posible acceder, por medio de una liga de navegación, a la página: `Registro cliente`; mientras que sólo desde la página `Catálogo` es posible acceder a la página `Reservación`.

La página `Registro cliente` contiene un mensaje con las instrucciones para realizar el registro y un formulario que le permite a un visitante registrarse como cliente para realizar una reservación.

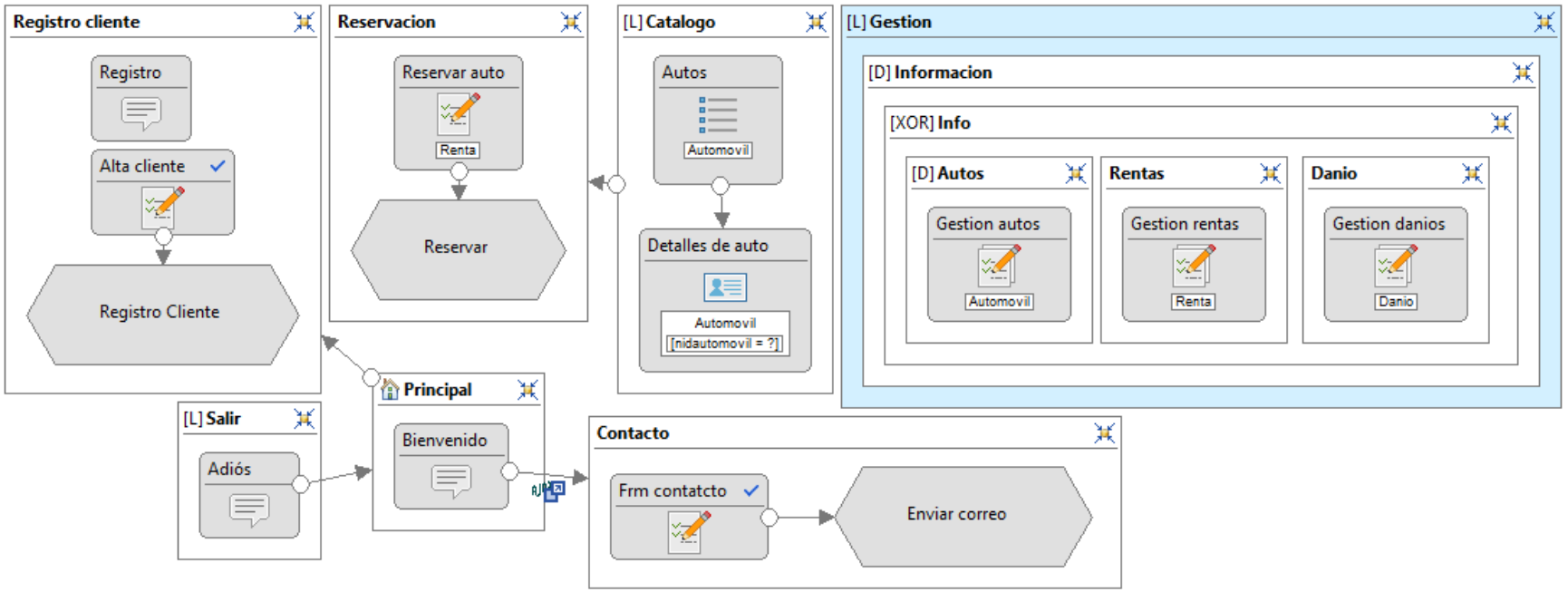

Fig. 4.2 Modelo navegacional en IFML del caso de estudio FastRent

La página `Catálogo` tiene una lista (`List`) de los autos disponibles, la cual está asociada a un elemento `Details` que le da la opción al cliente de consultar los detalles de cada uno de los autos enlistados si así lo desea; esa página tiene un flujo de navegación, que va hacia la página `Reservación`, ésta contiene un formulario que permite realizar la reservación de un automóvil. Por otra parte, la página `Gestión` está modelada como Área, ya que cuenta con un tipo de componente alternativo `(XOR)` que se traduce como pestañas; dentro de este componente se modelaron tres páginas, una por cada pestaña, cada una de ellas contiene un elemento `MultipleForm` que permite realizar la gestión de la entidad que tenga relacionada. Y, finalmente, se tiene la página `Salir`, que contiene solo un mensaje que despide al usuario y cuanta con un flujo de navegación hacia la página `Principal`.

Los formularios del modelo navegacional susceptibles de maquetarse son: `Reservar auto`, `Alta cliente` y `Frm contacto`.

El formulario `Reservar auto` se modeló con siete campos de captura, un campo de selección simple y uno de selección múltiple; los campos y sus tipos de dato se muestran en la Fig. 4.3.

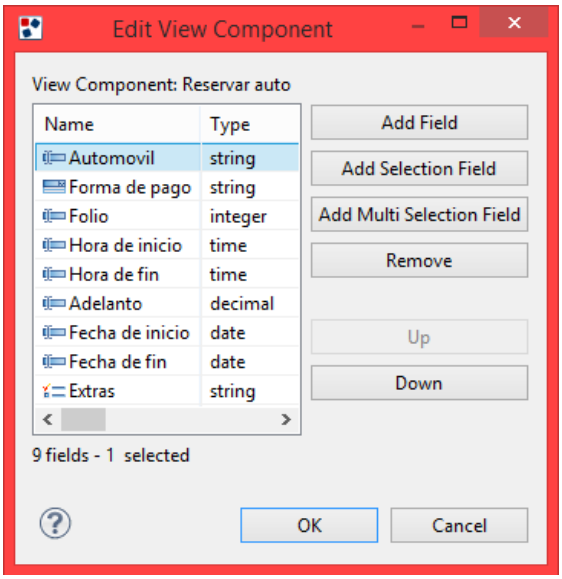

Fig. 4.3 Caso de estudio FastRent: Lista de controles para el formulario Reserva auto

En este formulario, a los campos `Forma de pago` y `Extras`, se les agregaron `Slots`. Los `Slots` en *WebRatio Web Platform* se aplican a campos de captura de datos para definir valores específicos. Como aparece en la Fig. 4.4, al campo `Forma de pago` se le agregaron los siguientes `Slots: Crédito, Efectivo y Transferencia`; mientras que al campo `Extras: Aire acondicionado, Pantalla táctil, Automático, Estándar, Polarizado y Rastreo satelital.`

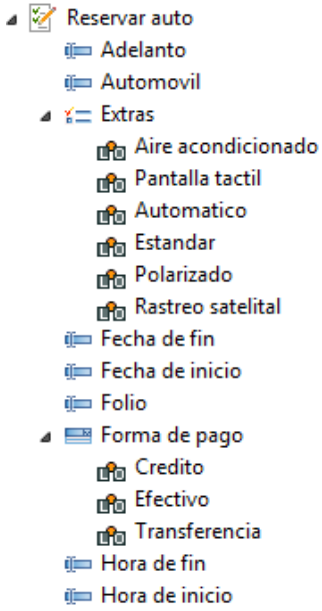

Fig. 4.4 Caso de estudio FastRent: Slots asigndos a los campos Extras y Forma de pago de Reservar auto

En el formulario `Alta Cliente` se incluyeron trece campos de captura que aparecen en la Fig. 4.5, que cuentan con los siguientes tipos de dato: `String, Integer, Password y Boolean.`

A algunos de los campos del formulario `Alta Cliente` se les asignaron validaciones (Fig. 4.6); para los campos `Clave, Contrasenia, Licencia` y `Nombre` una validación de tipo obligatoria, para el campo `Correo` una validación de tipo correo electrónico (`email`) y para el campo `RFC`, una validación de tipo expresión regular que define el formato aceptable de un RFC en México.

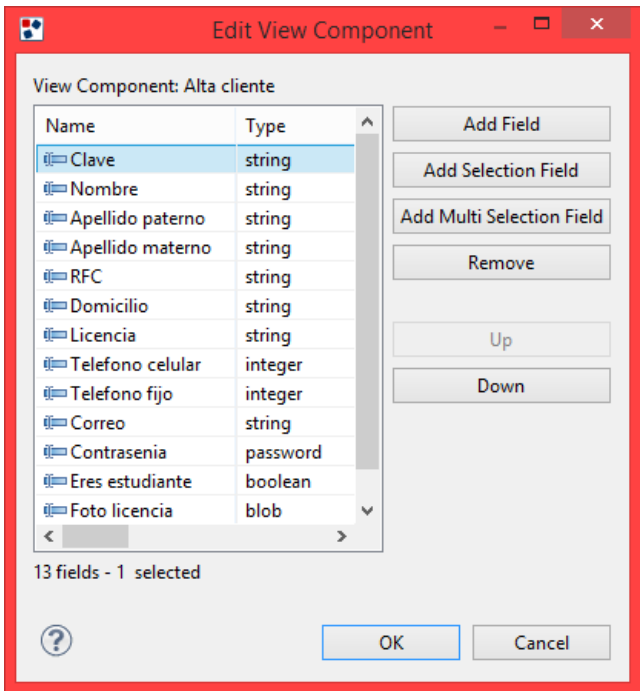

Fig. 4.5 Caso de estudio FastRent: Lista de controles para el formulario Alta Cliente

Para el último formulario, `Frm contacto`, se asignaron tres campos de captura, los cuales aparecen enlistados en la Fig. 4.7, dos de tipo `String` y uno de tipo `Text`.

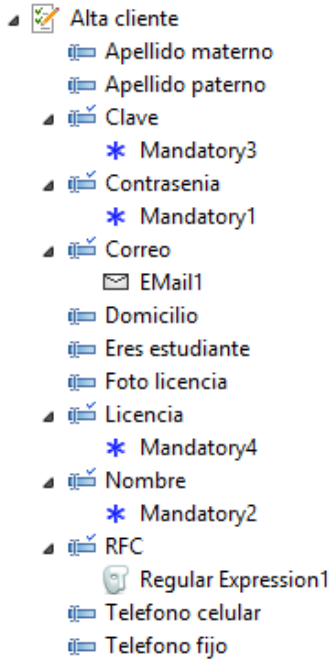

Fig. 4.6 Caso de estudio FastRent: Lista de las validaciones asignas a los controles del formulario Alta Cliente

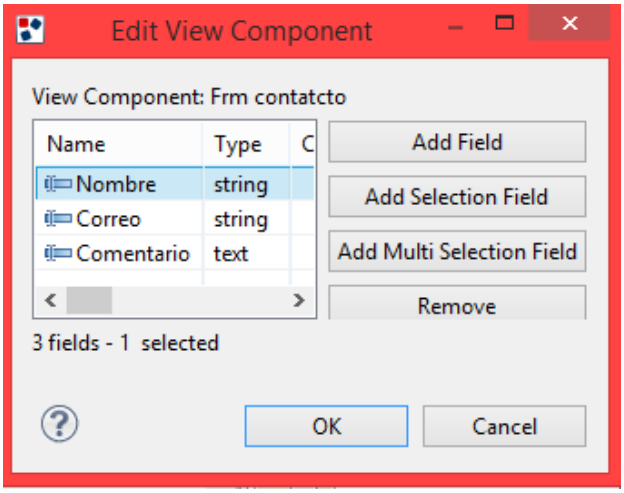

Fig. 4.7 Caso de estudio FastRent: Lista de controles para el formulario Frm contacto

A los campos del formulario `Frm contacto` se les asignaron las siguientes validaciones (Fig. 4.8): para los campos `Comentario` y `Nombre` una validación de campo obligatorio y para el campo `Correo` una validación de tipo correo electrónico.

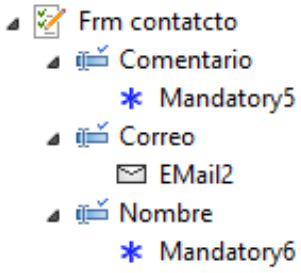

Fig. 4.8 Caso de estudio FastRent: Lista de las validaciones asignas a los controles del formulario Frm contacto

Una vez que se definieron el modelo de dominio y el modelo navegacional en IFML, se exportó todo el proyecto y se extrajo el XML de la aplicación.

4.1.3 Generador de Aplicaciones Enriquecidas en funcionamiento en el caso FastRent

La pantalla de configuración es la primera que aparece al ejecutar ITOs_IBRAIN donde el usuario selecciona la tecnología de salida y la distribución general de la aplicación, después elige los archivos XML y XMI (el XMI es opcional, se trata del modelo UML generado con Visual Paradigm versión 14) que describen los modelos de negocio, dominio y navegacional; también indica el directorio donde se encuentran los archivos de estilo - CSS y JS si la tecnología de salida es PHP con jQuery o un archivo JAR si la

tecnología de salida es JSF con PF – y, por último, ingresar una ruta de salida donde se generarán las carpetas y los archivos de la aplicación resultante. La Fig. 4.9 muestra la primera pantalla con la configuración para la generación de la aplicación *FastRent*.

Fig. 4.9 Caso de estudio FastRent: Pantalla de Configuración del Módulo de maquetado

La segunda pantalla del generador corresponde al módulo de maquetado, en la Fig. 4.10 se muestra la distribución de todos los controles realizada para el formulario `Reservar auto`, su configuración fue de 4 columnas por 6 filas y la orientación de la etiqueta arriba del control. Por omisión, si el generador identifica un campo de tipo selección, lo traduce a un combo desplegable; sin embargo, es posible cambiar esta representación, en este caso el control `Forma de pago` se cambió por radios. Además, la palabra `Automóvil` fue acentuada con una de las funcionalidades que ofrece el módulo de maquetado.

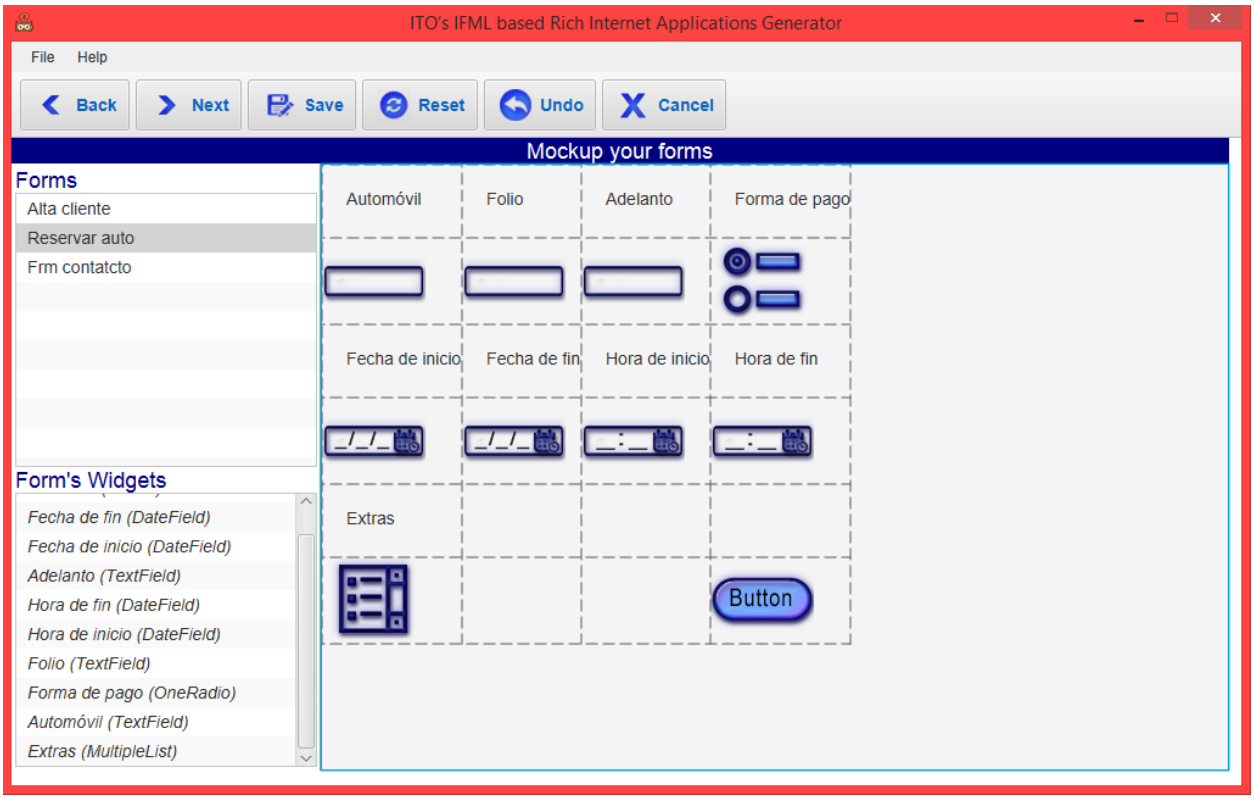

Fig. 4.10 Caso de estudio FastRent: Maquetado del formulario Reservar auto

En la Fig. 4.11 se muestra la distribución de controles realizada para el formulario `Alta Cliente`, la configuración para este formulario fue la siguiente: 4 columnas por 8 filas y la etiqueta ubicada arriba del control. En este formulario algunas etiquetas (`label`) como: `Teléfono`, `¿Eres un estudiante?` y `Contraseña` fueron modificadas, respecto al modelo original, para que estuvieran bien escritas en español.

Para el maquetado del formulario `Reservar auto`, se le asignó una configuración de dos columnas por cuatro filas y la orientación de la etiqueta arriba del control. Para este formulario solo se realizó la distribución de sus controles, la cual aparece en la Fig. 4.12

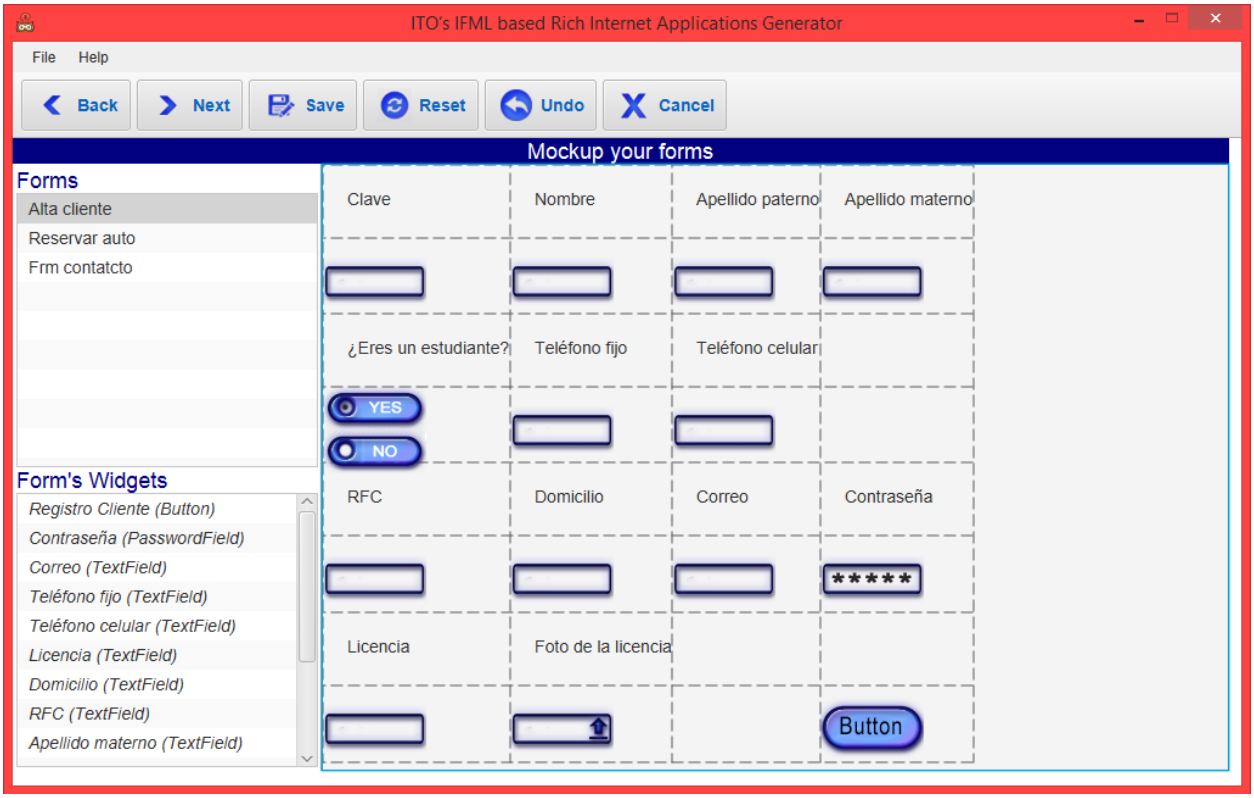

Fig. 4.11 Caso de estudio FastRent: Maquetado del formulario Alta Cliente

.

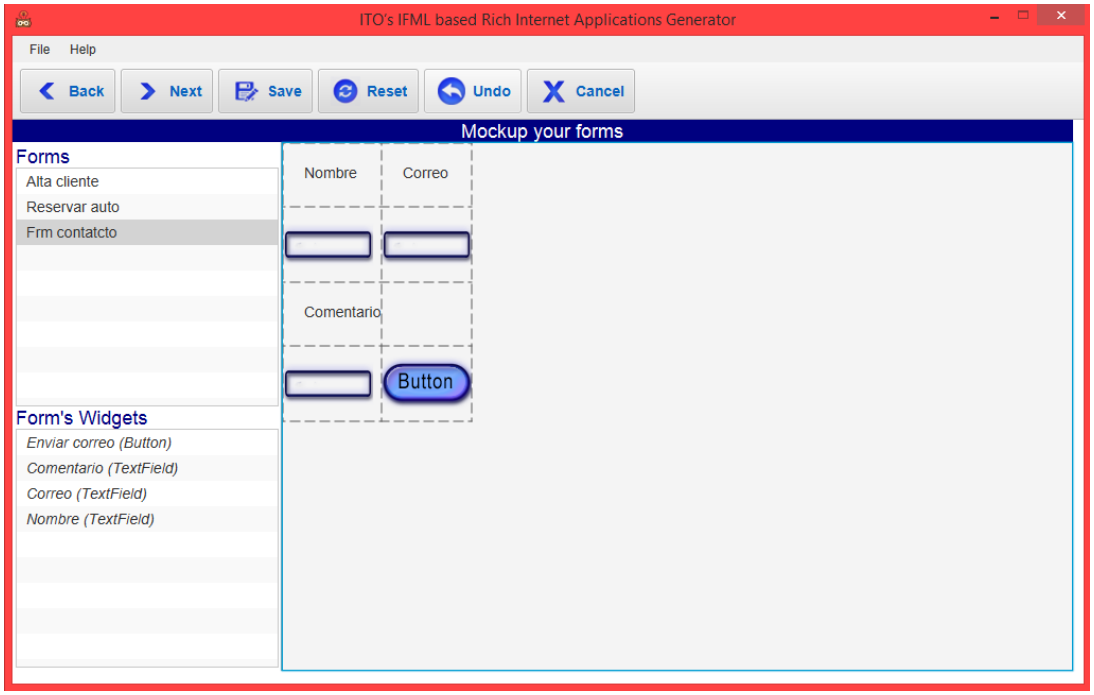

Fig. 4.12 Caso de estudio FastRent: Maquetado del formulario Frm contacto

La tercera pantalla que muestra ITOs_IBRAIN corresponde al Resumen de la Configuración (Fig. 4.13), donde aparece toda la información proporcionada en la pantalla de Configuración.

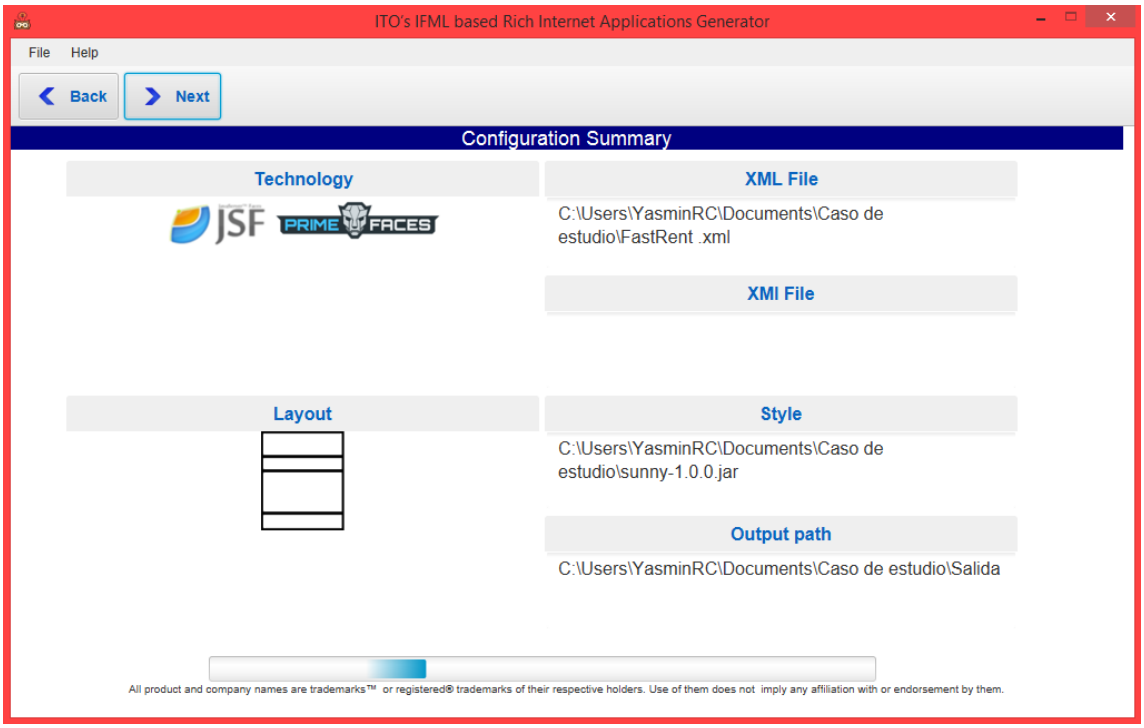

Fig. 4.13 Caso de estudio FastRent: Pantalla de Resumen del Módulo de maquetado

La cuarta y ultima pantalla de ITOs_IBRAIN es Resultado de la Aplicación (Fig. 4.14), en esta pantalla aparece el listado de la ruta de todos los archivos generados para la aplicación resultante.

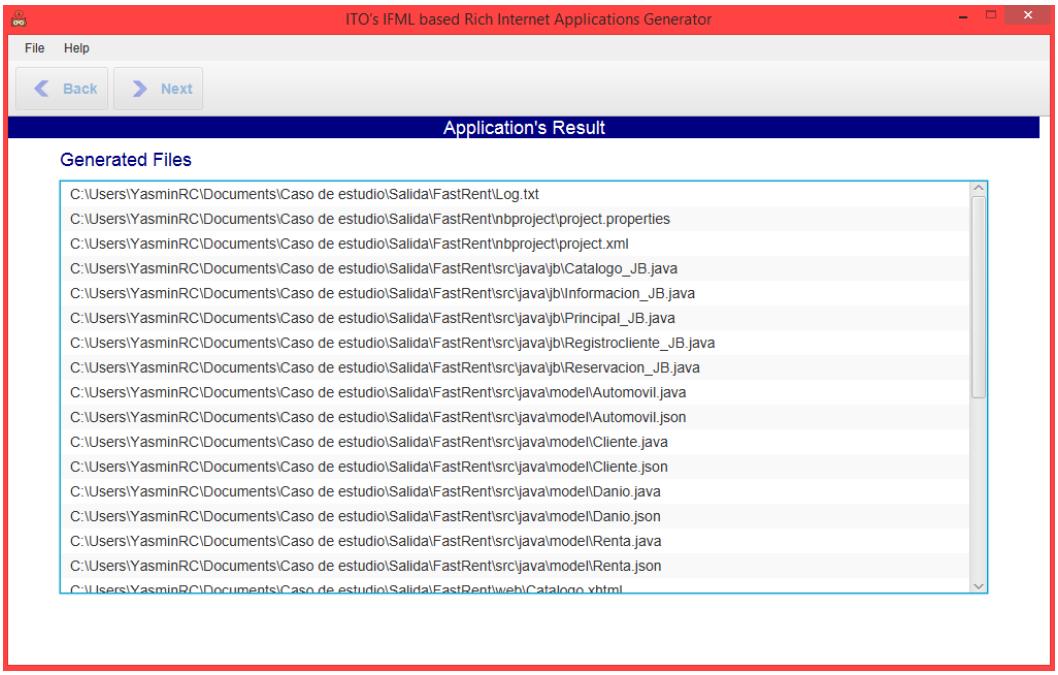

Fig. 4.14 Caso de estudio FastRent: Pantalla de Resultado del Módulo de maquetado

4.1.4 Aplicación generada en la combinación de Java ServerFaces y PrimeFaces

Para el caso de JSF y PF, el código resultante corresponde además a un proyecto Web del IDE NetBeans (Fig. 4.15), esto le facilita al usuario final visualizar, manipular y modificar el código generado. Sin embargo, es importante destacar que el código fuente generado funciona en cualquier IDE que soporte proyectos Web en Java.

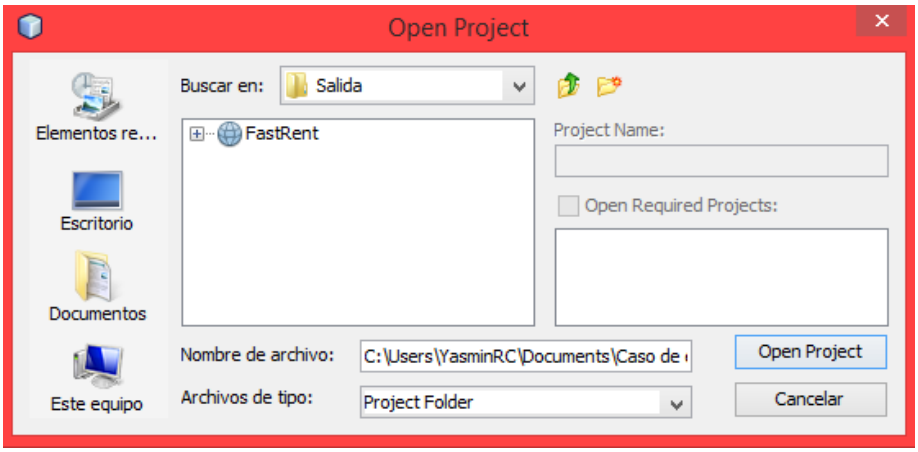

Fig. 4.15 Caso de estudio FastRent en JSF + PF: Aplicación generada

Al ejecutar la aplicación en el servidor, se visualiza la página principal con el mensaje de bienvenida y un botón (Fig. 4.16).

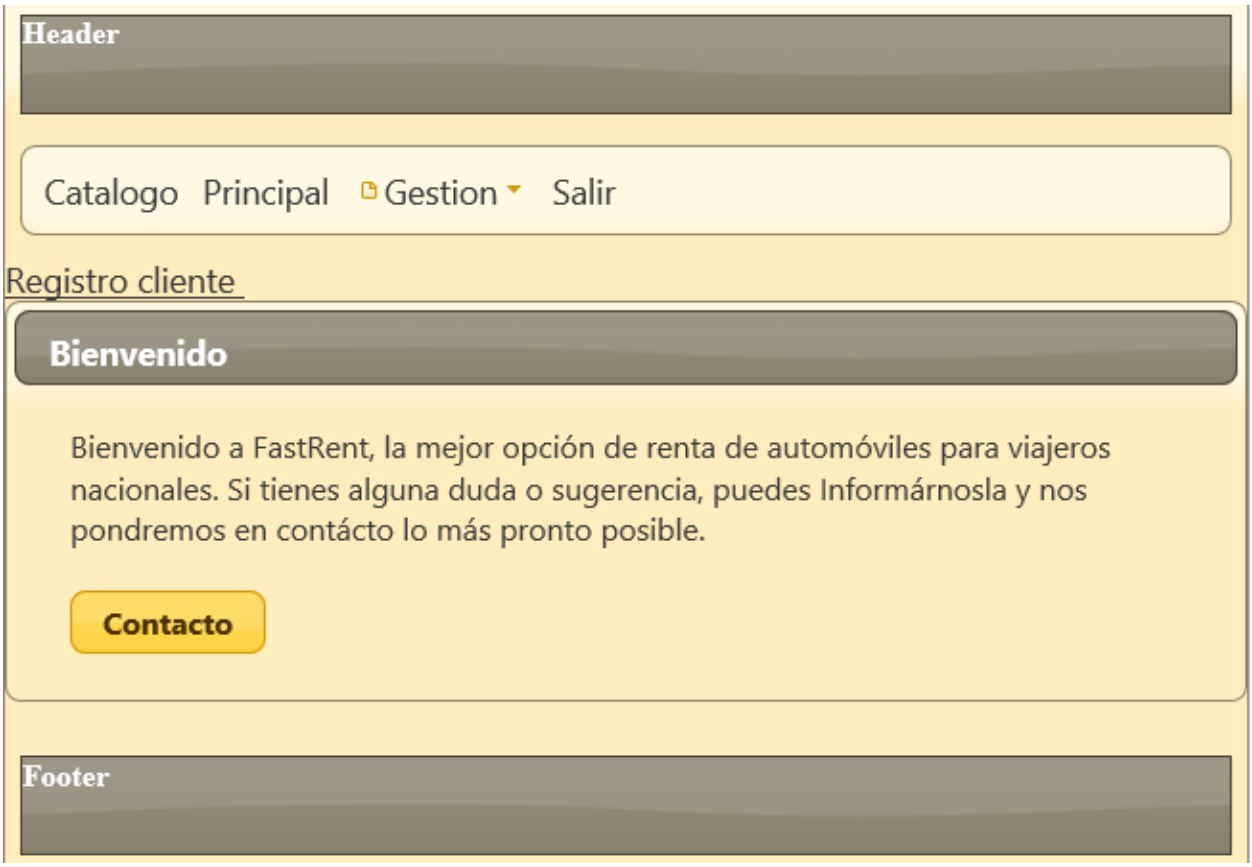

Fig. 4.16 Caso de estudio FastRent en JSF + PF: Página Principal de la Aplicación

El botón `Contacto` despliega un diálogo emergente, el cual contiene un formulario ilustrado en la Fig. 4.17 y que corresponde con la distribución realizada en el módulo de maquetado en la Fig. 4.12 y que además cuenta con las validaciones asignadas en *WebRatio Web Platform* en la Fig. 4.8.

En la Fig. 4.18 se muestra el formulario `Alta cliente` resultante de la generación, este formulario cumple con la distribución maquetada en la Fig. 4.11 y cumple con las validaciones asignadas en *WebRatio Web Plataform* en la Fig. 4.6.

Fig. 4.17 Caso de estudio FastRent en JSF + PF: Formulario Contacto

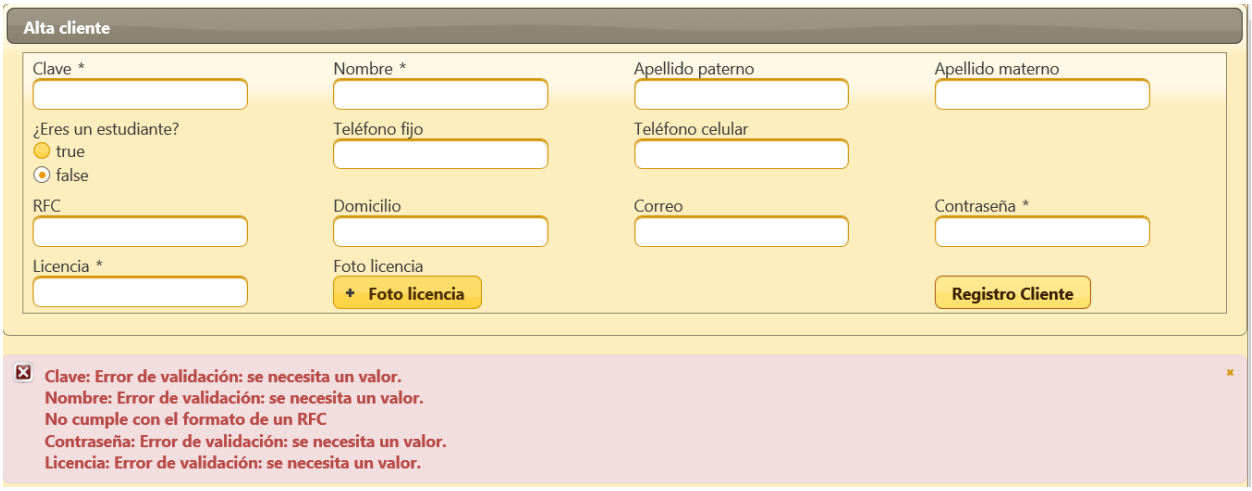

Fig. 4.18 Caso de estudio FastRent en JSF + PF: Formulario Alta cliente

En la Fig. 4.19 se muestra el formulario que permite realizar a un cliente la reservación de un auto; el formulario respeta el maquetado hecho en la Fig. 4.10; también se comprueba el cambio de tipo control para el caso de Forma de Pago y de Extras, así como la inclusión del acento a Automóvil.

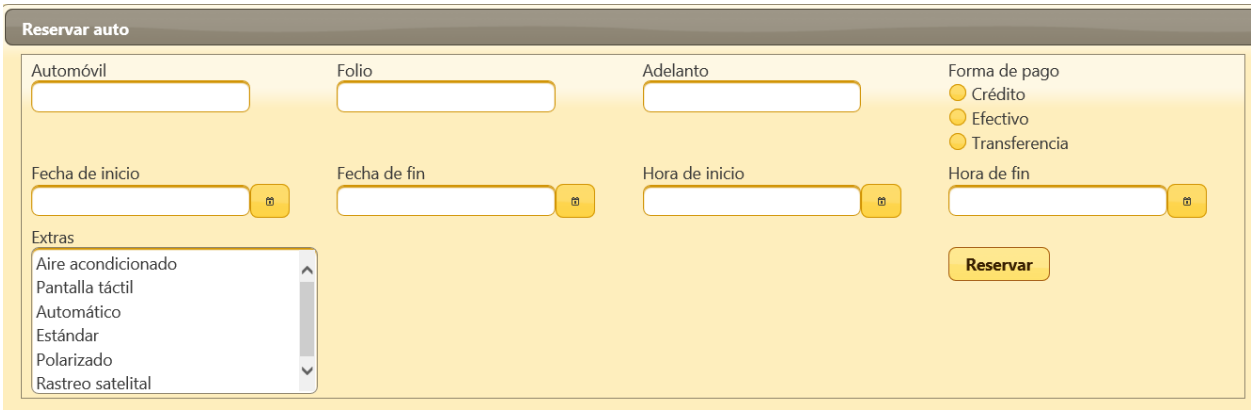

Fig. 4.19 Caso de estudio FastRent en JSF + PF: Formulario Reservar auto

4.1.4.1 Formularios responsivos en JSF con PF

Cuando la Aplicación reconoce que el tamaño de la pantalla es pequeño, la distribución de los campos se adapta, colocándose etiquetas y controles uno debajo del otro de forma vertical. En la Fig. 4.20 se muestra un ejemplo de cómo la aplicación cambia la distribución cuando se tiene una pantalla de celular, en este caso se usó la herramienta que ofrece el visualizador *Firefox*, asignando el tamaño de pantalla que tiene un iPhone 6/7/8. De esta forma se demuestra que la aplicación resultante es responsiva.

Fig. 4.20 Caso de estudio FastRent en JSF + PF: Formulario responsivo Alta cliente

4.1.5 Aplicación generada con la combinación de PHP y jQuery

Para el mismo modelo y maquetado, la aplicación resultante con las tecnologías PHP y jQuery (Fig. 4.21) cumple con el patrón arquitectónico MVC y es una aplicación lista para instalarse en un servidor y acceder a ella mediante un visualizador Web.

Fig. 4.21 Caso de estudio FastRent en PHP + jQuery: Aplicación generada

En la Fig. 4.22 se muestra la página principal de la aplicación *FastRent* en PHP y jQuery, la cual cuenta con tres elementos en la barra de navegación, dos enlaces accesibles sólo desde esta página y un botón (`Contacto`) que permite desplegar un diálogo emergente. Esta pantalla es consistente con el modelo navegacional realizado en IFML que se muestra en la Fig. 4.2 .

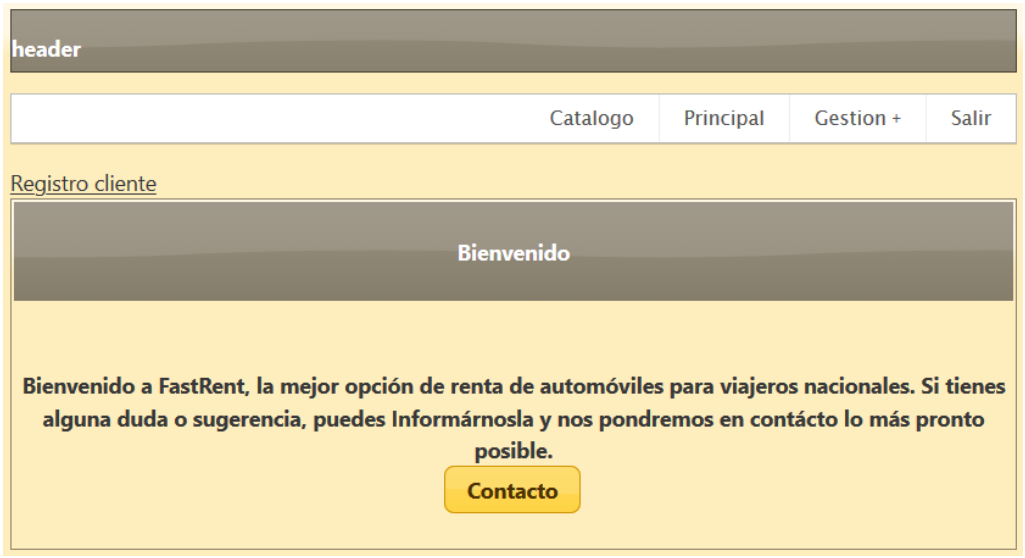

Fig. 4.22 Caso de estudio FastRent en PHP + jQuery: Página Principal de la Aplicación

Al dar clic al botón `Contacto` de la página principal, se despliega el diálogo ilustrado en la Fig. 4.23, este formulario cuenta con la distribución de controles realizada en la Fig. 4.12 y con las validaciones asignadas en la Fig. 4.8.

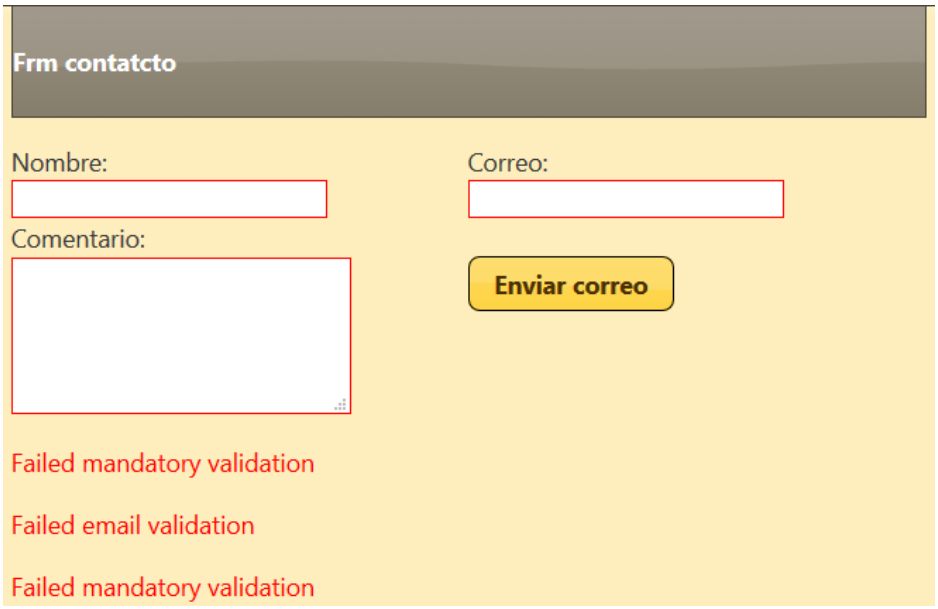

Fig. 4.23 Caso de estudio FastRent en PHP + jQuery: Formulario Contacto

Para registrar un nuevo cliente, se obtuvo el formulario de la Fig. 4.24, este formulario cuenta con la distribución de controles realizada en la Fig. 4.11 y los validadores asignados desde *WebRatio Web Platform* a ciertos controles, como se muestra en la Fig. 4.6.

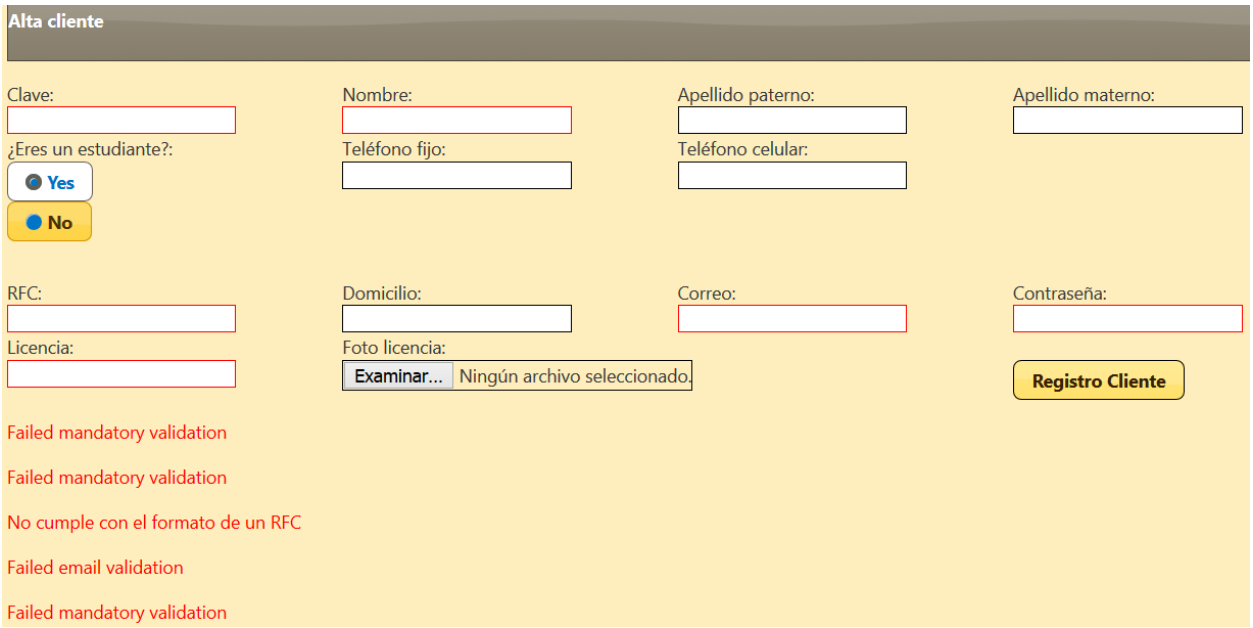

Fig. 4.24 Caso de estudio FastRent en PHP + jQuery: Formulario Alta cliente

En la Fig. 4.25 se muestra el formulario resultante `Reservar auto`, este formulario cuenta con la distribución realizada en Fig. 4.10.

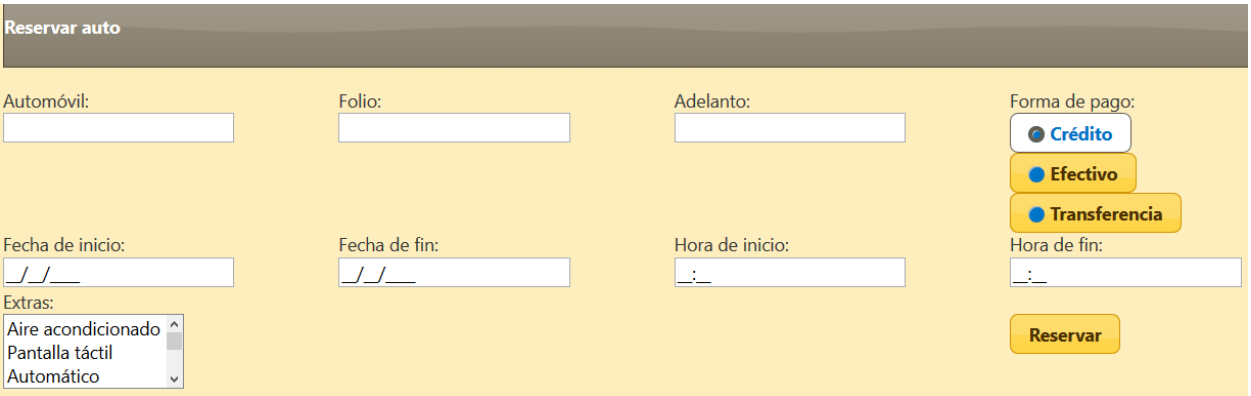

Fig. 4.25 Caso de estudio FastRent en PHP + jQuery: Formulario Reservar auto

Además cabe mencionar que los textos de las etiquetas cambiados en el módulo de maquetado se reflejan correctamente en las etiquetas de los campos que aparecen en los formularios de la aplicación generada para PHP con jQuery.

4.1.5.1 Formularios Responsivos en PHP con jQuery

Cuando la aplicación reconoce que el tamaño de la pantalla es pequeño, las etiquetas y los campos se colocan uno debajo del otro de forma vertical. En la Fig. 4.26 se muestra un ejemplo de cómo la aplicación cambia la distribución cuando se visualiza en la pantalla de un celular, en este caso se usó la herramienta que ofrece el visualizador *Firefox*, asignando el tamaño que tiene un iPhone 6/7/8. De esta manera se comprueba que la aplicación resultante es responsiva también para la combinación de tecnologías PHP y jQuery.

Fig. 4.26 Caso de estudio FastRent en PHP + jQuery: Formulario Alta Cliente Responsivo

4.2 Casos de estudio realizados en las estancias

Durante el periodo de estancias profesionales realizadas en el Instituto de Investigación y Desarrollo Tecnológico de la Armada de México (INIDETAM), se realizaron cuatro casos de estudio más.

Los casos de estudio consistieron en modelar cuatro aplicaciones que ya están funcionando en el INIDETAM, pasar los modelos correspondientes por el generador y comparar los resultados tanto de las aplicaciones mismas como de los tiempos de desarrollo. Debido al acuerdo de confidencialidad con el INIDETAM no es posible presentar el detalle de estos casos; sin embargo, se comprobó que las aplicaciones

resultantes del generador son equivalentes a las desarrolladas por el método tradicional y que efectivamente se disminuye el tiempo de construcción aun cuando se sume el tiempo de modelado.

4.2.1 Sistema de Información de la Armada de México (SIAM)

SIAM es una aplicación Web desarrollada en PHP que se encarga de enviar a los usuarios (empleados de INIDETAM) tareas que deben realizar o documentos dirigidos a ellos, consultar esos elementos y recibir notificaciones una vez recibidas. Esta aplicación cuenta con niveles de acceso (múltiples usuarios), uno de los usuarios es el capturista, que se encarga de hacer la gestión (Altas, bajas, modificaciones y consultas) de los documentos y asignar las tareas a los empleados, esta asignación puede ser a un solo empleado o a múltiples, por lo cual también tiene la opción de consultar múltiples empleados asociados a una tarea o documento. Y finalmente otro capturista es el que se encarga de dar de Alta y Modificar información de los empleados del instituto.

SIAM tuvo un periodo de desarrollo del 23 de Enero al 16 de Junio del 2017, con actividades como presentación del equipo de desarrollo con los directivos de la Dirección de Simulación de Modelado y Simulación, capacitación, análisis, entrevistas, desarrollo de diagramas y modelos, desarrollo de la base de datos, implementación de la vista de la aplicación (tablas, formularios, notificaciones), conexión con la base de datos, generación de reportes, implementación de correo electrónico, pruebas internas, externas y elaboración de la documentación pertinente.

Según el cronograma del proyecto, el tiempo que se propuso para el desarrollo que concierne únicamente con la vista de la aplicación debía realizarse en dos semanas pero el tiempo real en el que se desarrolló fue de tres.

El periodo de estancias duró un mes, el tiempo dedicado para la realización del modelo de dominio y el modelo navegacional en IFML de este caso de estudio fue de cinco días.

Las actividades realizadas en ese periodo de tiempo fueron: Obtención de documentación de SIAM, análisis de requerimientos, identificación de entidades y

atributos de las clases para el modelo de dominio, modelado de dominio, modelado navegacional, realizar pruebas con el Generador, maquetar la distribución más adecuada para todos los formularios y validar la Aplicación generada.

El tiempo y esfuerzo que le tomó al desarrollador realizar la GUI de la aplicación y darle funcionamiento es mayor en comparación con el tiempo que se invirtió para obtener algo equivalente con el generador. Además que la aplicación obtenida cuenta con la arquitectura MVC; donde en la parte del Modelo, la aplicación cuenta con todas las clases definidas en el modelo de dominio y con métodos listos para ser enlazados con una base de datos y obtener o hacer afectaciones a datos reales de un repositorio, el Controlador contiene las variables de todos los datos que va a recibir de la Vista y los métodos que permiten aplicar la navegación, la búsqueda de información y las validaciones, y finalmente la Vista contiene todos los elementos visuales modelados en IFML (formularios, tablas, botones, ligas, barra de menú, entre otros) y, en el caso de los formularios, se presentan ya con los controles distribuidos y son responsivos ya que al visualizarse en celulares todos los controles se redistribuyen de tal manera que se apilan uno debajo del otro.

Capítulo 5 Conclusiones y recomendaciones

Este capítulo se centra en describir las conclusiones a las que se llegó después de desarrollar el proyecto de tesis e implementarlo, las contribuciones que se obtuvieron con el Módulo de Maquetado y una serie de recomendaciones para incorporar al Generador a futuro.

5.1 Conclusiones

La principal contribución que se tuvo al desarrollar el Módulo de Maquetado e integrarlo a ITOs_IBRAIN fue entregar formularios con un mejor diseño visual, ya que por medio del maquetado se le permite al usuario realizar la distribución de controles personalizada a través un *Grid* situado en el área de maquetado que implementa eventos *Drag and Drop* bajo el soporte de JavaFX con el fin de hacer la interacción de la aplicación más sencilla. Además, el módulo le da mucho más control al desarrollador sobre la aplicación que va a obtener, ya que le permite cambiar la representación visual de controles de selección simple o de selección múltiple modelados en *WebRatio Web Platform* y también le permite cambiar los textos de las etiquetas de los controles con el fin de eliminar errores de ortografía.

Otro de los aportes importantes del Módulo de Maquetado fue la arquitectura que se diseñó, ya que es capaz de soportar los requerimientos funcionales y no funcionales de una aplicación que permite el maquetado de controles e implementa el comportamiento de *Drag and Drop* que ofrece JavaFX en elementos que son texto e imágenes al mismo tiempo dependiendo del lugar en que se encuentren. También la arquitectura se diseñó de tal forma que es capaz de crear los enlaces necesarios para incorporarse en el proceso propio del Generador sin afectar lo previamente desarrollado y sólo recibir la Representación Intermedia (RI) para obtener la información esencial de los formularios y realizar modificaciones puntuales a la RI con el fin de que los cambios realizados en maquetado se vieran reflejados en la aplicación generada dependiendo de la tecnología de salida elegida.

Con lo anterior mencionado se confirma que se cumplió el objetivo general y cada uno de los objetivos específicos marcados para el desarrollo de este proyecto de tesis.

5.2 Recomendaciones

Actualmente el proceso de generación del Generador se realiza en una sola ejecución, es decir, se configura la generación, si hay formularios se maquetan, si no, se realiza la generación y se obtiene la aplicación; este proceso no tiene ningún otro requerimiento si las aplicaciones modeladas en IFML son pequeñas, pero en los casos donde se tengan aplicaciones de mayor envergadura es posible que resulte inviable, ya que debe maquetar todo en una sola ejecución. Por otra parte, si el usuario requiere volver a maquetar uno o más formularios, se obliga a maquetar todos los formularios encontrados. Por lo anterior se propone como recomendación permitirle al usuario guardar el maquetado de los formularios en un archivo para su uso posterior, que contenga la configuración de la aplicación de salida y adicionar una opción al Generador en la pantalla de Configuración que le permita al usuario ingresar el archivo almacenado si es que existe y que el Generador sea capaz de mostrarle al usuario los formularios maquetados que haya guardado con anterioridad.

Por otro lado, durante las múltiples pruebas realizadas en los casos de estudio se detectaron otros aspectos de mejora para agregar al módulo: 1) hacer una distinción visual en la lista de formularios, a los formularios que ya están maquetados y guardados, 2) mostrar más información del formulario (nombre y lista de controles) al momento de asignar la configuración del *Grid* y 3) así como se permite modificar los textos de las etiquetas de los controles, también sería deseable permitir modificar los textos de los nombres de los formularios dado que sirven como títulos en la vista.

Productos académicos

Yasmin Rosales Cruz, Beatriz Alejandra Olivares Zepahua, Ignacio López Martínez, Celia Romero Torres, Luis Ángel Reyes Hernández.

"Comparativa de herramientas de maquetado como técnica de obtención de requerimientos no funcionales".

Tecnologías Útiles para la Sustentabilidad Energética para Beneficio de la Sociedad

CIINDET 2019, Cuernavaca Morelos, México

Estado: Presentado

Yasmin Rosales Cruz, Beatriz Alejandra Olivares Zepahua, Ignacio López Martínez, Celia Romero Torres, Luis Ángel Reyes Hernández.

"Arquitectura de un módulo de maquetado desarrollado en JavaFX que complementa un Generador de Aplicaciones Enriquecidas".

Abstraction & Application Revista electrónica de la Facultad de Matemáticas de la Universidad Autónoma de Yucatán.

CONISOFT 2019, UNAM, CDMX, México

Estado: Presentado

Referencias

[1] J. Dulh, “Rich Internet Applications”, *IDC white papers*, nov-2003. [En línea]. Disponible en: https://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf. [Consultado: 12-sep-2018].

[2] J. Conallen, *Building Web applications with UML*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[3] A. Guevara-Benites, “Frontend y Backend”, *DevCode Tutoriales*, 21-sep-2016. [En línea]. Disponible en: https://devcode.la/tutoriales/frontend-y-backend/. [Consultado: 13-sep-2018].

[4] S. Estévez, “Desarrollo de un generador de Aplicaciones Enriquecidas de Internet modeladas bajo el patrón arquitectónico MVC usando UML e IFML”, Instituto Tecnológico de Orizaba, Orizaba, Veracruz, 2018.

[5] J. D. Gauchat, *El gran libro de HTML5, CSS3 y Javascript*. Marcombo, 2012.

[6] M. Brambilla y P. Fraternali, *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufmann, 2014.

[7] J. Niño, *Tareas básicas I (Sistemas operativos monopuesto)*. Editex, 2011.

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

[9] “JavaServer Faces.org”, *Sitio oficial*. [En línea]. Disponible en: http://www.javaserverfaces.org/. [Consultado: 13-sep-2018].

[10] J. F.- js.foundation, “jQuery”, *¿Qué es jQuery?*, 2018. [En línea]. Disponible en: https://jquery.com/. [Consultado: 13-sep-2018].

[11] H. van der Linden, G. Boers, H. Tange, J. Talmon, y A. Hasman, “A multi disciplinary EPR system.”, *Int. J. Med.*, pp. 70, 2–3, 149–160, 2003.

[12] M. Warcholinski, “What Is the Difference Between Wireframe, Mockup and Prototype?”, *Blog Brainhub.eu*, 20-abr-2016. .

[13] “Sketch - El kit de herramientas de diseño digital”. [En línea]. Disponible en: https://www.sketchapp.com/. [Consultado: 05-mar-2019].

[14] Balsamiq Studios, LLC, “Balsamiq Wireframes”, *Balsamiq Wireframes*. [En línea]. Disponible en: https://balsamiq.com/wireframes/. [Consultado: 04-mar-2019].

[15] V. J. Muñoz-Eslava, *El nuevo PHP. Conceptos avanzados.* Vicente Javier Eslava Muñoz, 2013.

[16] M. Morales-Sanchez, *Manual de Desarrollo Web basado en ejercicios y supuestos prácticos.* Lulu.com, 2012.

[17] R. Lerdorf, K. Tatroe, y P. MacIntyre, *Programming PHP*. O’Reilly Media, Inc., 2006.

[18] “PrimeFaces | Ultimate UI Framework for Java EE”, *PrimeFaces*. [En línea]. Disponible en: https://www.primefaces.org/. [Consultado: 13-sep-2018].

[19] S. Gálvez-Rojas, *Java a Tope: Traductores Y Compiladores Con Lex/yacc, Jflex/cup Y Javacc.* Sergio Gálvez Rojas.

[20] F. Ricca, G. Scanniello, M. Torchiano, G. Reggio, y E. Astesiano, “Assessing the Effect of Screen Mockups on the Comprehension of Functional Requirements”, en *ACM Transactions on Software Engineering and Methodology*, 2014, vol. 24, pp. 1–38.

[21] M. Brambilla, A. Mauri, M. Franzago, y H. Muccini, “A Model-Based Method for Seamless Web and Mobile Experience”, presentado en Mobile!’16, Amsterdam, Netherlands, 2016, pp. 33–40.
[22] D. Raneburger, H. Kaindl, R. Popp, V. Šajatovi´c, y A. Armbruster, “A Process for Facilitating Interaction Design through Automated GUI Generation”, presentado en SAC’14, Gyeongju, Korea, 2014, pp. 1324–1330.
[23] M. Hamdani, W. Haider-Butt, M. Anwar-Waseem, y F. Azam, “A Systematic Literature Review on Interaction Flow Modeling Language (IFML)”, presentado en ICMSS 2018, Wuhan, China, 2018, pp. 134–138.
[24] A. C. Oran, E. Nascimento, G. Santos, y T. Conte, “Analysing Requirements Communication Using Use Case Specification and User stories”, presentado en Proceedings of 31st Brazilian Symposium on Software Engineering, Fortaleza Cerá, Brzil, 2017, pp. 214–223.
[25] W. S. Lasecki, J. Kim, N. Rafter, O. Sen, J. P. Bigham, y M. S. Bernstein, “Apparition: Crowdsourced User Interfaces That Come To Life As You Sketch Them”, presentado en CHI 2015, Seoul, Republic of Korea, 2015, pp. 1925–1934.
[26] M. Brade, A. Sehl, y R. Groh, “Between the Lines: A Comparative Study of Freeform-Based Knowledge- Map-Creation with Paper and Tablet”, presentado en CHI’16 Extended Abstracts, San Jose, CA, USA, 2016, pp. 3006–3012.
[27] S. Ha, J. Park, y J. Lee, “Increasing Interactivity of Paper Prototyping with Smart Pen”, presentado en Proceedings of HCI, KOREA, 2015, pp. 76–82.
[28] I. Bouchrika, L. Ait-Oubelli, A. Rabir, y N. Harrathi, “Mockup-based Navigational Diagram for the Development of Interactive Web Applications”, presentado en ISDOC’13, Lisbon Portugal, 2013, pp. 27–32.
[29] M. Machado, R. Couto, y J. Creisssac, “MODUS: Model-based user interfaces prototyping”, presentado en EICS ’17, Lisbon Portugal, 2017, pp. 111–118.
[30] C. Bernaschina, S. Comai, y P. Fraternali, “Online Model Editing, Simulation and Code Generation for Web and Mobile Applications”, presentado en ACM 9th International Workshop on Modelling in Software Engineering (MiSE), Italia, 2017, pp. 33–39.
[31] A. Rodríguez, P. González, y G. Rossi, “Sketching for designing enactive interactions”, presentado en Interaccion’14, Puerto de la Cruz, Tenerife, Spain, 2014, pp. 39–40.
[32] I. Koren y R. Klamma, “The Exploitation of OpenAPI Documentation for the Generation ofWeb Frontends”, presentado en WWW’18, Lyon France, 2018, pp. 781–787.
[33] R. M. L. M. Moreira y A. C. R. Paiva, “Towards a Pattern Language for Model-Based GUI Testing”, presentado en EuroPLoP’14, Irsee, Germany, 2014, pp. 26–34.
[34] K. Frajták, M. Bureš, y Jelínek Ivan, “Transformation of IFML Schemas to Automated Tests”, presentado en RACS’ 15, Prague, Czech Republic, 2015, pp. 509–511.
[35] P. N. Cortez-Herrera, “Metodología SCRUM utilizada en asesorias de tesis para alumnos de ingeniería”, en *8 congreso internacional de computación*, Taxco de Alarcón, Guerrero, México, 2018, vol. 9.

[36] “UXPin - The Premier UX Design Platform”. [En línea]. Disponible en: https://www.uxpin.com/. [Consultado: 12-mar-2019].
[37] “JFormDesigner - detalle del complemento NetBeans”. [En línea]. Disponible en: http://plugins.netbeans.org/plugin/39703/jformdesigner. [Consultado: 13-mar-2019].
[38] “Diseño de UX rápido y herramientas de estructura alámbrica”. [En línea]. Disponible en: https://www.visual-paradigm.com/features/ux-design-and-wireframe-tools/. [Consultado: 18-mar-2019].
[39] “Herramientas web modernas | Visual Studio”, *Visual Studio*. [En línea]. Disponible en: https://visualstudio.microsoft.com/es/vs/features/web/. [Consultado: 18-mar-2019].
[40] “ISO / IEC / IEEE 42010: Modelo conceptual”. [En línea]. Disponible en: http://www.iso-architecture.org/42010/cm/. [Consultado: 03-abr-2019].