Generación semiautomática de aplicaciones Web

De modelos en IFML a aplicaciones basadas en frameworks PHP Yii2 y Laravel

Damián Paolo Rotta
Gonzalo Sebastián Pallotta

Director: Ing. Héctor Javier Ruidias

Universidad Gastón Dachary
Departamento de Ingeniería y Cs. de la Producción
Ingeniería en Informática

Junio 2018

AGRADECIMIENTOS

Al **Ing. Hector Ruidias**, por acompañarnos en las etapas de gestación de la idea que se realiza en este proyecto y por su dirección en la redacción del presente trabajo.

A nuestro amigo y futuro ingeniero, **Héctor Emanuel Klikailo**, por compartir incertidumbres, angustias y conocimientos durante todo el proceso de elaboración de este trabajo.

Al **Ing. Edgardo Belloni**, por su dirección durante la elaboración del proyecto de este Trabajo Final de Carrera, informes, artículos y otros documentos relacionados.

A nuestro amigo y futuro ingeniero, **Patricio Bonsembiante**, por su ayuda desinteresada en la redacción de uno de los anexos de este Trabajo Final de Carrera y sus aportes en nuestro repositorio de Github.

Al **Dr. Diego Godoy**, por estar siempre dispuesto a darnos ánimo y retroalimentación sobre nuestro trabajo.

Al **Ing. Roberto Suenaga**, por estar siempre dispuesto a dedicarnos parte de su tiempo para escucharnos y darnos su opinión.

Al **Mgtr. Daniel Sachi**, por oficiar de revisor y señalarnos algunos puntos débiles en la redacción.

A la **Universidad Gastón Dachary** por ser el *alma mater* de nuestra formación como profesionales informáticos.

A los **Ing. Fernando Amatte**, **Ing. Edgardo Belloni, Ing. Fernando Aguirre, Ing. Karina Eckert, Lic. Eduardo Silva, C.P. Carolina Pérez e Ing. Jorge Ferrari,** que nos permitieron trabajar con esta idea en el marco de sus cátedras, permitiéndonos así enriquecer este trabajo desde distintos puntos de vista.

A todos los docentes que durante la carrera nos enseñaron mucho más de lo que se imaginan.

A nuestras familias, compañeros y amigos, por apoyarnos y aguantar infinidad de postergaciones.

PALABRAS CLAVES

Web Engineering, MDD *(Model-Driven Development)*, MDWE *(Model-Driven Web Engineering)*, IFML *(Interaction Flow Modeling Language)*, PHP Framework.

RESUMEN

La web es una de las facetas más importante de la Internet y ha evolucionado para dejar de ser aquella plataforma utilizada para publicar material estático a ofrecernos la posibilidad de entretenernos, realizar negocios y hasta aprender en línea. Sin embargo, a pesar de la evolución y extensión de tecnologías que buscan facilitar el desarrollo en el dominio Web éste sigue siendo una tarea difícil. El surgimiento y adopción de metodologías que promueven la aplicación de un enfoque de desarrollo dirigido por modelos, ciertamente, representa un paso significativo en la evolución de la Ingeniería Web y, por tanto, una buena noticia para la comunidad global de desarrolladores. Con tal enfoque se apunta a reducir el esfuerzo inherente al desarrollo de sistemas Web complejos, así como también, a mejorar la calidad, eficiencia y reusabilidad en tales desarrollos. El presente trabajo presenta una propuesta que tiene como objetivo facilitar la adopción de dicho enfoque atendiendo a las necesidades y tendencias actuales de los desarrolladores Web.

SIGLAS Y ACRÓNIMOS

ATL	Atlas Transformation Language
CIM	Computation Independent Model
CoC	Convention over Configuration
CRUD	Create-Read-Update-Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
DRY	Don't Repeat Yourself
EMF	Eclipse Modeling Framework
EMFTVM	EMF Tranformation Virtual Machine
EMOF	Essential MOF
ERD	Entity-Relationship Diagram
HTML	HyperText Markup Language
IDE	Integrated Development Enviroment
IEEE	Institute of Electrical and Electronics Engineers
IFML	Interaction Flow Modeling Language
IGU	Interfaz Gráfica de Usuario
JSP	JavaServer Pages
KISS	Keep It Simple, Stupid
M2M	Model-to-Model
M2T	Model-to-Text
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDWE	Model-Driven Web Engineering
MOF	Meta-Object Facility
MVC	Model-View-Controller / Modelo-Vista-Controlador
NDT	Navigational Development Techniques
OCL	Object Constraint Language
OMG	Object Management Group
PaaS	Platform as a Service
PHP	PHP Hypertext Processor
PIM	Platform Independent Model
PSM	Platform Specific Model
RIA	Rich Internet Applications
TFC	Trabajo Final de Carrera
UML	Unified Modeling Language
UP	Unified software development Process
URI	Universal Resource Identifier
URL	Uniform Resource Locator
UWE	UML-Based Web Engineering
WebML	Web Modeling Language
WSDM	Web Site Design Model
XMI	XML Metadata Interchange
XML	Extensible Markup Language

SUMARIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

La Web ha transitado un largo camino desde sus comienzos en los años 90. Tradicionalmente una aplicación Web era simplemente un conjunto de archivos conectados mediante hipervínculos que presentaban información usando textos y gráficos limitados y llegaban al usuario mediante el uso de un navegador. Con el tiempo esta idea de generación estática de contenido fue mutando, acercándose más al dinamismo de las aplicaciones de escritorio tradicionales. Este cambio en la concepción de la Web dio pie a la búsqueda de nuevas tecnologías y maneras de pensar el desarrollo Web.

Con base en estas nuevas tecnologías surgieron paradigmas de desarrollo y utilización de software impensados hace algunos años atrás. Tal podría ser el caso de la orientación a servicios y el *cloud-computing.* La rápida evolución y extensión de tecnologías que buscan facilitar el desarrollo en este dominio Web se puede explicar, al menos en parte, considerando que pese a la gran cantidad de nuevos ambientes de desarrollo y de la sofisticación de las herramientas utilizadas en ellos, construir aplicaciones web sigue siendo una tarea difícil.

Una de las mayores dificultades con las que se encuentra el desarrollo de aplicaciones Web es que los cambios de requerimientos se dan de manera particularmente constante y con un costo de adaptación relativamente bajo. Todos los días aparecen nuevas posibilidades que conllevan a nuevos requerimientos y esto diferencia al dominio Web de otros dominios de utilización del software donde los usuarios y clientes no están tan conscientes del surgimiento de todas estas nuevas posibilidades [1].

Tales circunstancias dieron lugar al surgimiento de lenguajes de modelado web, que han tenido un rol importante en los últimos tiempos. A partir del final de la década de los 90, se evidenció que los lenguajes de modelado existentes (teniendo a UML (*Unified Modeling Language*) como máximo exponente en la materia) no alcanzaban a cubrir la necesidad de expresar los nuevos tipos específicos de comportamiento de las aplicaciones web respecto a: navegación, composición de las páginas, interactividad, etc.

En [2] se sugiere que la manera más inteligente de enfrentarse a los retos en la evolución del software web es usar enfoques dirigidos por modelos *(Model-Driven)*. Esto es así porque este paradigma pretende aumentar el nivel de abstracción en el que se piensa acerca de las aplicaciones web. Este aumento de abstracción puede lograrse a partir del uso de determinados modelos y de la derivación de programas de manera semiautomática por medio de la transformación de estos modelos.

Además, la utilización de modelos para el desarrollo de aplicaciones da ventajas adicionales porque permite describir funcionalidades complejas sin entrar en detalles de implementación.

La mayoría de los enfoques metodológicos surgidos en los últimos diez años adoptan notaciones tradicionales de los lenguajes de modelado (ERD (*Entity-Relationship Diagram*), UML, etc.). Sin embargo, la navegación o el comportamiento interactivo que deberían tener las páginas estaban expresadas en un notación completamente nueva y, por tanto, desconocida para los desarrolladores que quisieran usarlos.

Es de destacar que la web que hoy conocemos es posible gracias al auge y la adopción de estándares, pero pese a ello, por contraparte en el ámbito de la ingeniería de Software, existe cierta renuencia a incorporarlos a través de tecnologías, metodologías y prácticas en general.

Esto es así porque usualmente se concibe al desarrollo de software como una tarea creativa que se vería limitada por procedimientos metodológicos que podrían parecer (o ser) demasiado burocráticos y porque los estándares existentes carecen de ciertas cualidades esenciales. Algunas críticas concretas, por ejemplo, al estándar UML pueden verse en [3], [4]. Sin embargo, la comunidad del desarrollo de software debe aceptar que la existencia de nuevos estándares es una señal de progreso y un punto inicial para construir, de manera conjunta, nuevas ideas y mejorar las que ya existen.

Esta situación comenzó a cambiar hacia principios de 2013, cuando un consorcio internacional sin fines de lucro destinado a la definición de estándares para la industria del software conocido como OMG (*Object Management Group*), ha convertido al lenguaje de modelado de aplicaciones web IFML (*Interaction Flow Modeling Language*) en estándar de modelado. IFML está diseñado, en líneas generales, para expresar el contenido de las aplicaciones web y su interacción con los usuarios y para definir el comportamiento dinámico de las aplicaciones web que se modelan. Estas características lo convierten en una metodología más que interesante para los desarrolladores web.

La expresividad de IFML se evidencia en su capacidad de expresar la interacción del usuario y el comportamiento de la aplicación en dominios: HTML (*HyperText Markup Language*) tradicionales, RIAs (*Rich Internet Application*), móviles, de escritorio, multicanales y *context-aware* e incluso interfaces de control humano-máquina embebidas. Además, como mecanismos de extensión, IFML provee un metamodelo completo conforme a MOF (*MetaObject Facility*) -intercambiable en formato XMI (*XML Metadata Interchange*)- y especificaciones como *profile* de UML.

Mediante la utilización de este estándar IFML, y atendiendo a necesidades y tendencias actuales de los desarrolladores Web, el presente trabajo presenta una propuesta que facilita la adopción del enfoque MDWE (*Model-Driven Web Engineering*). Esta propuesta aborda uno de los trabajos futuros propuestos en [5] y detalla la problemática planteada en una Idea-Proyecto presentada como parte de un Informe Final de Práctica Profesional Supervisada[1,2].

Todo el código escrito y generado como parte de este proyecto es abierto y se encuentra subido al repositorio IFML2PHP[3] en Github.

PLANTEAMIENTO DEL PROBLEMA

El Desarrollo de Software Dirigido por Modelos, como disciplina ingenieril, cuenta con la gran ventaja de basar un proceso de producción en modelos conceptuales. No obstante, en el contexto particular de la Ingeniería Web Dirigida por Modelos, aún después 25 años, no fue posible encontrar una metodología que cubra el ciclo de vida de desarrollo completo. Una ilustración gráfica de esta situación puede verse en la Figura 1.1, donde el eje horizontal representa las fases de un ciclo de vida de desarrollo genérico y el eje vertical el momento aproximado en el cual surge cada enfoque metodológico.

1 *Ref. Expte. N°020/17-PPS* – Universidad Gastón Dachary.

2 Pasantía realizada en el contexto del Proyecto de I+D adjudicado y registrado por la Sec. de Investigación y Desarrollo de UGD –Código / N° 11 en Res. de Rectorado UGD N° 18/A/14; Área/s Temática/s: Tecnologías de la Información; Ingeniería de Requerimientos; Ingeniería Web– *"Estrategias basada en ontologías para reducir el gap semántico entre modelos CIM en el marco de MDA"*. Directora: Dra. María Laura Caliusco (UTN FR Santa Fe); Docentes-Investigadores (UGD): Ing. Héctor Ruidias e Ing. Edgardo A. Belloni.

3 https://github.com/Dipiert/ifml2php

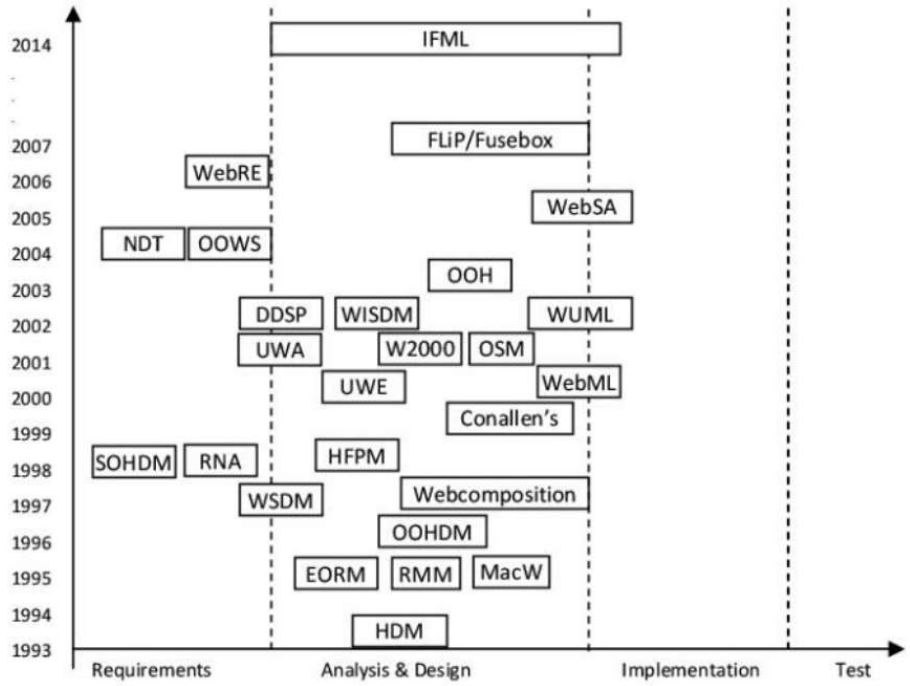

Figura 1.1. Evolución y alcance de los procesos de desarrollo Web más conocidos [77]

A esta carencia de soporte, se suma que la mayor parte de estos métodos de desarrollo se enfocan en las fases de análisis y diseño o a un aspecto muy específico como en el caso del diseño de sitios web desde una perspectiva centrada en el usuario, también conocido como WSDM (*Web Site Design Model*)[6].

Estos inconvenientes se ven agravados por el hecho que, no solo hay multitud de enfoques, cada uno encarando solo un conjunto de aspectos del ciclo de vida, sino que están definidos sin tener en cuenta a los demás y no son compatibles entre sí. Podría tratarse, entonces, del extendido *"Not invented here syndrome"* (El síndrome de lo que no ha sido inventado aquí) con numerosos autores proponiendo notaciones diferentes para representar conceptos muy parecidos de maneras distintas.

Aún salvando los obstáculos conceptuales mencionados, no será posible la aplicación práctica de los logros de integración de modelos si no existe un conjunto de herramientas adecuadas para facilitar la aplicación de modelos, técnicas, transformaciones y mantenimiento de la coherencia del modelo en vistas a lograr, al final, código ejecutable.

Después de analizar algunos procesos de desarrollo de la Ingeniería Web Dirigida por Modelos como NDT (*Navigational Development Techniques*)[7], UWE (*UML-Based Web Engineering*)[8] e IFML[9]. Se eligió trabajar con ésta última por motivos que se consignarán en detalle en el apartado 2.4 del Estado del Arte.

La adopción de esta metodología, aún con todas sus ventajas, no es ideal: entre los desafíos emergentes, se señala el de extenderla para permitir la generación de código ejecutable en plataformas tecnológicas Web distintas a las que provee. La plataforma tecnológica cubierta, al momento, es JSP (*JavaServer Pages*)[10] y la generación de código hacia ella se hace utilizando la herramienta propietaria WebRatio [11]. Se descubre, por tanto, que es pertinente para el ámbito MDWE definir un modelo de diseño intermedio e

independiente que abstraiga características comunes entre plataformas cuya adopción actual sea más extendida.

OBJETIVOS

Objetivo General

- Proponer un metamodelo que constituya un modelo MVC Web para *frameworks* PHP que pueda integrarse a la metodología IFML.

Objetivos Específicos

- Enunciar un marco conceptual sobre Ingeniería Web, Model-Driven Development, Model-Driven Web Engineering y el soporte tecnológico actual que sustentan estos conceptos.
- Construir un metamodelo que mediante la abstracción de características comunes de una serie de *frameworks* PHP constituya un modelo MVC Web para éstos.
- Verificar el metamodelo construido, respecto a una serie de requisitos, mediante la implementación de reglas de transformación entre modelos y de modelo a código que cubran las abstracciones propuestas.

ANTECEDENTES

Aunque a la fecha no existen soluciones que generen código PHP (PHP Hypertext Preprocessor) a partir de modelos IFML, sí se encontraron propuestas MDD (*Model-Driven Development*) que partiendo desde modelos PIM (*Platform-Independent Model*) culminan en la generación tanto de código PHP plano como para *frameworks* de desarrollo PHP. Dichas propuestas son las siguientes:

- El *plugin* UML2PHP5 [12] diseñado para la herramienta de modelado “DIA”, el cuál genera automáticamente cierta estructura de código PHP basado en un diagrama de clases UML. Cabe destacar que este *plugin* no presenta avances desde Mayo de 2006.
- Un conjunto de herramientas provistas por Enterprise Architect [13] para la realización de transformaciones de modelo a código. En lo que se refiere a las transformaciones hacia PHP, convierte elementos PIM a elementos específicos del lenguaje -como ser clases PHP. La generación de métodos accesores y mutadores, de acuerdo a las opciones que fueron establecidas al momento de la transformación, ayuda a mantener la encapsulación de éstas clases.
- Un metamodelo del *framework* PHP CodeIgniter, presentado en [14] y utilizado para la generación de un archivo XML (*Extensible Markup Language*) que contenga los componentes principales de éste *framework* a partir de un diagrama de clases UML.
- Proyectos en repositorios Git, como [15] presentan la idea de la generación automática de aplicaciones PHP completas desde diagramas de clases UML. Además, como en el caso del proyecto referenciado, por medio de Doctrine se generan todas las operaciones típicas de un CRUD (Create-Read-Update-Delete) para el *framework* PHP Symfony.

- Un ejemplo particular de un posible *mapping* (implícito) entre un diagrama IFML y código PHP plano presentado en [16], En dicho ejemplo, se mezcla el código perteneciente a Modelos, Vistas y Controladores en un mismo *script*.

ORGANIZACIÓN DEL DOCUMENTO

El resto de este documento se organiza de la siguiente manera:

En la sección 2, se introduce el estado del arte de las técnicas, herramientas y paradigmas involucrados en el desarrollo de este Trabajo Final de Carrera. En esta sección se cumple el objetivo referido al desarrollo de un estado del arte como un marco conceptual tecnológico.

La sección 3 describe el metamodelo propuesto, las decisiones tomadas durante su construcción y una descripción de las clases que lo componen. El desarrollo de esta sección y la siguiente ayudan a alcanzar el objetivo referido a la construcción de un metamodelo.

Mientras que, en la sección 4, se especifica el diseño e implementación de las reglas de transformación tanto de modelo a modelo como de modelo a texto.

En la sección 5, se consigna el caso de estudio utilizado para la verificación de que las abstracciones propuestas en el metamodelo están cubiertas, son válidas y útiles para la generación aplicaciones Web. En esta sección y la siguiente se desarrolla la propuesta que apunta a cumplir con el objetivo de la verificación del metamodelo propuesto.

La sección 6, contiene una descripción y un análisis de los resultados obtenidos por las reglas de transformación, incluyendo las estructuras de archivos e interfaces gráficas de usuarios producidos. Esta sección también contiene las medidas realizadas sobre el metamodelo y sobre el conjunto de reglas modelo a modelo construidas.

Finalmente, en la sección 7, se presentan la conclusión y líneas de trabajo futuro.

2. ESTADO DEL ARTE

En este capítulo se presenta la base conceptual sobre la que se sustenta el presente trabajo y es el resultado de la búsqueda, revisión, compilación y reseña sistemática de conceptos, soluciones y tecnologías existentes en el contexto disciplinar considerando el dominio de aplicación referidos.

2.1. INGENIERÍA WEB

La **Ingeniería Web** ha sido definida como una disciplina emergente que promueve el empleo de enfoques sistemáticos, disciplinados y cuantificables, para lograr el desarrollo eficiente de sistemas y aplicaciones Web con atributos de alta calidad –cf.: [17], [18]. En particular, dicha disciplina científico-tecnológica se enfoca en la proposición, estudio sistemático, experimentación y mejora continua de metodologías, técnicas y herramientas que constituyan el soporte esencial del desarrollo de aplicaciones Web a lo largo de todo su ciclo de vida. De esta manera, abarca procesos de ingeniería de requisitos, diseño arquitectónico y detallado, construcción, evaluación y evolución, que tienen en cuenta características y aspectos que diferencian a las aplicaciones Web de otros tipos de sistemas de información, software o aplicaciones tradicionales –cf.: [19],[20].

Un paso importante en la evolución de la Ingeniería Web lo constituye el surgimiento de metodologías que se fundan en la aplicación de un enfoque ingenieril de desarrollo dirigido o guiado por modelos: ***Model-Driven Development*** **(MDD).**

2.2. MDD (MODEL-DRIVEN DEVELOPMENT)

Este enfoque ha sido definido como: *"Un paradigma de desarrollo que usa modelos como artefactos principales del proceso de desarrollo. Usualmente, en MDD la implementación es generada (semi)automaticamente desde los modelos" [21]*. MDD propone reducir el esfuerzo inherente al desarrollo de aplicaciones y sistemas complejos, así como también, mejorar la calidad y eficiencia de los desarrollos.

2.2.1. Ventajas

Entre otras de las ventajas de aplicar MDD, concretamente, se pueden citar las señaladas en [22] , a saber:

- Ya que el código y otros artefactos (como la documentación) se genera automáticamente desde los modelos, MDD implica una reducción de los costos de desarrollo. Si bien se debería considerar el costo de desarrollo o comprar transformaciones, es esperable que este costo se amortice mediante la reutilización de dichas transformaciones.
- El progreso de la tecnología hace que los componentes de software se vuelvan obsoletos rápidamente pero el uso de MDD ofrece una mejor adaptación a los cambios tecnológicos ya que los modelos de alto nivel están libres de detalles de la implementación, lo cual facilita la adaptación a los cambios que pueda sufrir la plataforma tecnológica subyacente o la arquitectura de implementación.
- Cuando se agrega una nueva función, sólo es necesario desarrollar el modelo específico para esa nueva función. El resto de la información necesaria para generar los artefactos de implementación ya ha sido capturada en las transformaciones y

puede ser re-utilizada y, por ello, MDD significa una mejor adaptación a los cambios de requisitos.

- La aplicación manual de las prácticas de codificación y diseño es una tarea propensa a errores. A través de la automatización, MDD favorece la generación consistente de artefactos del desarrollo.
- MDD incrementa la confianza en el desarrollo de nuevas funcionalidades y reduce los riesgos mediante la reutilización de artefactos..
- Los modelos omiten detalles de implementación que no son relevantes para entender el comportamiento lógico del sistema. Por ello, los modelos están más cerca del dominio del problema. MDD mejora la comunicación con los usuarios mediante la reducción de la brecha semántica entre los conceptos que son entendidos por los usuarios y el lenguaje en el cual se expresa la solución. Esta mejora en la comunicación influye favorablemente en la producción de software mejor alineado con los objetivos de sus usuarios.
- Utilizando MDD, las primeras etapas del desarrollo se focalizan en las actividades de modelado, por lo cual existe brinda la posibilidad de postergar la toma de decisiones de carácter tecnológico.

2.2.2. Metamodelos

Con el fin de utilizar MDD se definen y proveen mecanismos útiles para la especificación de abstracciones relevantes en términos de **metamodelos.**

Los metamodelos son herramientas conceptuales que proveen una solución a la multiplicidad de vocabularios y enfoques. Un metamodelo según [23] es una representación abstracta de conceptos, no enfocada en la terminología o en la manera en la que se expresan si no solamente en el concepto en sí.

La ventaja de contar con un metamodelo correcto es que éste minimiza la brecha comunicacional entre un concepto específico de un dominio y su implementación. Implementar la lógica de negocios en el mismo lenguaje que utiliza el experto del dominio, disminuye el riesgo de que un desarrollo que no se corresponda con sus requerimientos

Los modelos conceptuales o específicos de dominio –a distintos niveles de detalle y perspectiva- que permiten tratar de manera separada aspectos diferentes, e inicialmente, con independencia de la plataforma de destino del desarrollo, deben construirse conforme a algún metamodelo.

Las implementaciones específicas para determinadas plataformas tecnológicas se generan de manera (semi) automática a partir de transformaciones sistemáticas de modelos.

2.2.3. Testing

En el contexto de MDD, al igual que cualquier otro paradigma de desarrollo, es necesario probar el software obtenido con el objetivo de verificar y validar el producto final [24]. Para realizar tal tarea, los ingenieros de software enfrentan una serie de dificultades al momento de probar el software, como ser: limitaciones de tiempos, complejidad técnica y/o interpretación de código [25]. Por esta razón, el *testing* es una de las fases que más recursos demandan en el ciclo de vida de desarrollo de sistemas [26]. También, es

importante destacar que los costos que conllevan la realización de pruebas tienen directa relación con la complejidad del software a ser probado [27].

Se considera que a partir del uso de MDD como paradigma de desarrollo, los productos finales cuentan con una mayor calidad, aunque esto no disminuya la importancia de hacer *testing*. La Figura 2.1 muestra el llamado Modelo en V, muy conocido por los profesionales del *testing*, y que será utilizado en esta sección con el fin de lograr una explicación más gráfica de las particularidades del *testing* en MDD. La parte superior representa las distintas etapas del proceso de desarrollo de un producto Software, mientras que, la parte inferior modela los distintos tipos de *tests* realizados en cada una de estas etapas.

Las últimas etapas de la parte superior (Diseño del Sistema y Diseño de Componentes) y las últimas etapas de la parte inferior (*Testing* de los componentes e *Testing* de la Integración de Sub-sistemas) pueden ser automatizadas utilizando MDD. Permitiendo así que el rol de *tester* pueda estar más enfocado a la verificación y validación de los modelos, permitiendo así validar mejor la adecuación del sistema a las necesidades del negocio y de los requerimientos no funcionales.

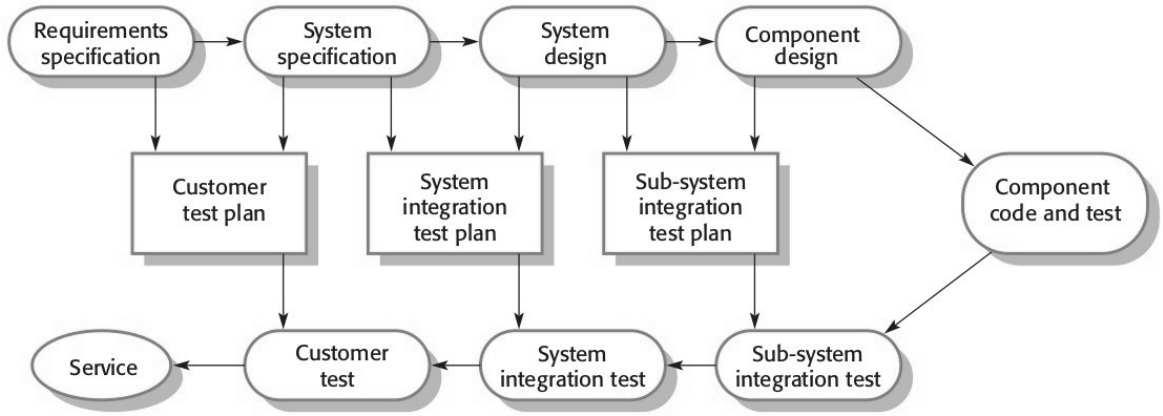

Figura 2.1. Modelo en V. [29]

Todo estos *tests* no serían posibles si los modelos no son precisos, completos y correctos. Para asegurar tales características se debe contar con un mecanismo verificador del modelo que se use para verificarlo en función a ciertas reglas de consistencia y remover errores.

Toda esta automatización puede provocar que se considere que el uso de MDD como paradigma de desarrollo hace que el rol del *tester* en el desarrollo de software deje de ser relevante[4], esto no es así ya que:

- Es necesario validar los modelos.
- Se deben probar sistemas complejos en ambientes complejos y que se ajusten a los requerimientos del negocio.
- Deben probarse y verificarse las herramientas MDD.
- Ciertos enfoques MDD incluyen modelos técnicos y/o código manual, lo que significa que es necesario que los *tests* de más bajo nivel sean producidos de manera manual.

4http://www.theenterprisearchitect.eu/blog/2010/11/03/model-driven-development-the-end-of-the-test-profession/

2.3. PROPUESTA DE INFRAESTRUCTURA

2.3.1. OMG (Object Management Group)

Desde su creación en 1997, la OMG busca ayudar a los usuarios a resolver problemas de integración, proveyendo especificaciones abiertas, interoperables e independientes del proveedor.

La necesidad de especificaciones que manejen la integración a través del ciclo de vida completo: desde el modelo de negocios hasta el diseño del sistema, la construcción de componentes, el ensamblado, la integración, el despliegue, la gestión y la evolución del sistema, hacen que el estándar **MDA (Model-Driven Architecture)** propuesto por la OMG sea el próximo paso en la resolución de problemas de integración.[28]

2.3.2. MDA (Model-Driven Architecture)

Es la definición de un enfoque de especificación de sistemas computacionales que separa la especificación de la funcionalidad del sistema de la especificación de la implementación de esa funcionalidad en una plataforma tecnológica específica.

Cómo se expresa en [29], se recomiendan 3 tipos de modelos abstractos del sistema en la utilización de MDA, a saber:

- *CIM (Computation Independent Model)*: modela las abstracciones del dominio importantes para el sistema. Se pueden desarrollar varios CIM que reflejen diferentes vistas del sistema. A veces también se los llama modelos de dominio.
- *PIM (Platform Independent Model)*: modela las operaciones del sistema sin referencia a su implementación. Usualmente se describe utilizando modelos UML[30] que muestran la estructura estática y el comportamiento del sistema.
- *PSM(Platform Specific Models)*: son el resultado de transformar el PIM a distintas plataformas tecnológicas específicas. Pueden existir varias capas de PSM cada una agregando algún detalle específico.

La Figura 2.2 indica el orden de las transformaciones esenciales en el proceso MDD: Una herramienta que soporte MDD tomará un CIM como entrada, lo transformará en un PIM, tomará éste como entrada y lo transformará en uno o varios PSMs.

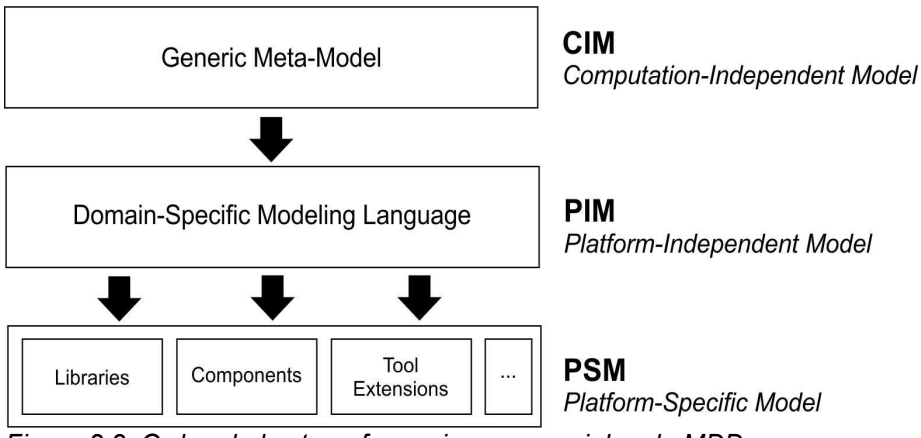

Figura 2.2. Orden de las transformaciones esenciales de MDD

El código ejecutable puede generarse usando la misma herramienta o quizá otra, a partir de un PSM dado.

Las herramientas para transformar modelos a modelos y modelos a código, se llaman de transformación M2M (Model-to-Model) y M2T(Model-to-Text), respectivamente.

2.4. MDWE (MODEL-DRIVEN WEB ENGINEERING)

Teniendo en cuenta que las ventajas que promueve MDD podrían ser aprovechadas en el ámbito del desarrollo Web, surge entonces el concepto de **MDWE**. Término el cual refiere a la adopción del paradigma dirigido por modelos en metodologías de desarrollo Web. Dentro de la última década, se han realizado numerosos esfuerzos de investigación y desarrollo en el contexto de MDWE, surgiendo así diferentes propuestas de nuevas metodologías, extensiones de lenguajes de modelado, herramientas y *frameworks* para facilitar su aplicación, así como también, criterios de evaluación de su calidad –cf. [16], [23], [31]–[42]

2.4.1. Metodologías MDWE

En cuanto a las metodologías de MDWE existentes se consideraron aquellas que brindan un mejor soporte al ciclo de vida de desarrollo Web en cuánto a especificación de requerimientos, separación de *concerns*, y por ende, de metamodelos y transformaciones. Las metodologías seleccionadas para su estudio fueron:

- NDT (*Navigational Development Techniques*): es un enfoque metodológico de MDWE que cubre el ciclo de vida completo en el desarrollo de aplicaciones web, incluido la fase de pruebas y control de calidad. Originalmente, éste método fue creado solo para ingeniería de requerimientos para el desarrollo de aplicaciones web, gradualmente fue evolucionando hasta convertirse en una metodología.
- UWE (*UML-Based Web Engineering*): es una metodología orientada a objetos, iterativa e incremental basada en UP (*Unified Software Development Process*)[43] que usa exclusivamente técnicas, notaciones y extensiones UML.

  La metodología UWE está destinada a proveer soporte en el desarrollo de aplicaciones Web y RIA con especial foco en la sistematización, personalización y la generación semiautomática de código. -cf.: [44], [45].
- IFML (*Interaction Flow Modeling Language*): el lenguaje estándar de modelado de flujos de interacción IFML está inspirado en la experiencia de más de 10 años de WebML (*Web Modeling Language*) [41] y WebRatio. Está diseñado para ser aplicado en la especificación e implementación de aplicaciones móviles y Web complejas con especial foco en la expresión del contenido, las interacciones del usuario y el control del comportamiento del *front-end* de aplicaciones software.

Después de experimentar y comparar entre sí las metodologías mencionadas anteriormente, se optó por el uso de IFML por los siguientes motivos: -cf.: [46], [47]

- Hereda la simplicidad y la expresividad de WebML, mejorándolo por la supresión de notación que no pertenece a *concerns* de interacción como cuestiones de procesos de negocio y agrega a los eventos de la interfaz de usuario como “ciudadanos de primera clase” en la notación.
- Con el fin de integrar IFML con el universo UML (y de tener un metamodelo claramente definido), el lenguaje soporta (y utiliza) el mismo tipo de mecanismos de extensión ya provistos por UML -como estereotipos, valores etiquetados, etc. Esto le

permite ser extensible en la definición de nuevos conceptos, por ejemplo, interfaz o tipos de eventos con dispositivos novedosos.

- Es conciso, evita la redundancia y reduce el número de tipos de diagramas y conceptos necesarios para expresar la interfaz saliente y las decisiones de diseño respecto a la interacción.
- Asegura la implementabilidad, es decir, soporta la construcción de *frameworks* de transformación de modelos y generadores de código que puedan mapear un PIM en un PSM adecuado para, por fin, convertirlos en aplicaciones ejecutables en un amplio rango de plataformas tecnológicas y dispositivos de acceso.
- Asegura la reutilización a nivel de modelo, es decir, da soporte a la definición de patrones de diseño que puedan ser almacenados, documentados, buscados, devueltos y reutilizados en otras aplicaciones.
- Permite la aplicación de reglas de inferencia a nivel de modelado, mediante las cuáles pueden aplicarse automáticamente patrones de modelado y detalles en donde sea que éstos puedan inferirse del contexto. Esto da la posibilidad de que los modeladores eviten la necesidad de especificar información que puede inferirse. Un ejemplo de esto es la inferencia automática de los parámetros que necesitan ser pasados de un componente a otro.

2.4.2. IFML (Interaction Flow Modeling Language)

La especificación del enfoque elegido incluye un *profile* que define una sintaxis basada en UML para expresar modelos IFML. En particular, este *profile*, extiende conceptos de los diagramas UML: de clases, de máquina de estados y de estructuras compuestas.

Metamodelo IFML

Está dividido en 3 paquetes: *Core*, *Extension* y *DataTypes*.

- El **paquete *Core*** contiene los conceptos que construyen la infraestructura de interacción en términos de *InteractionFlowElements*, *InteractionFlows*, and *Parameter*s.
- El **paquete *Extension*** contiene conceptos concretos que extienden al paquete *Core* con comportamientos más complejos.
- El **paquete *DataTypes*** contiene los tipos de datos *custom* definidos por IFML.

El metamodelo IFML usa los tipos de datos básicos del metamodelo de UML, especializa un número de metaclases de UML como la base para metaclases IFML y asume que IFML *DomainModel* está representado en UML.

Aunque, con fines de concisión, IFML condense todas sus perspectivas en un solo diagrama llamado *Interaction Flow Diagram, [46]* puede darse una descripción de alto nivel del metamodelo de IFML, estructurada en las siguientes áreas de *concern*:

- IFML Model
- Interaction Flow Model
- View Elements
- Interaction Flow Elements
- Events
- Specific View Components
- Parameters
- Expression

- ContentBinding

En el Anexo A, se especifica el detalle de cada una de estas perspectivas y en el Anexo B se detalla el significado de su notación.

Proceso de Desarrollo

La Figura 2.3 esquematiza un posible proceso de desarrollo y el lugar que ocupa IFML en ese flujo de actividades.

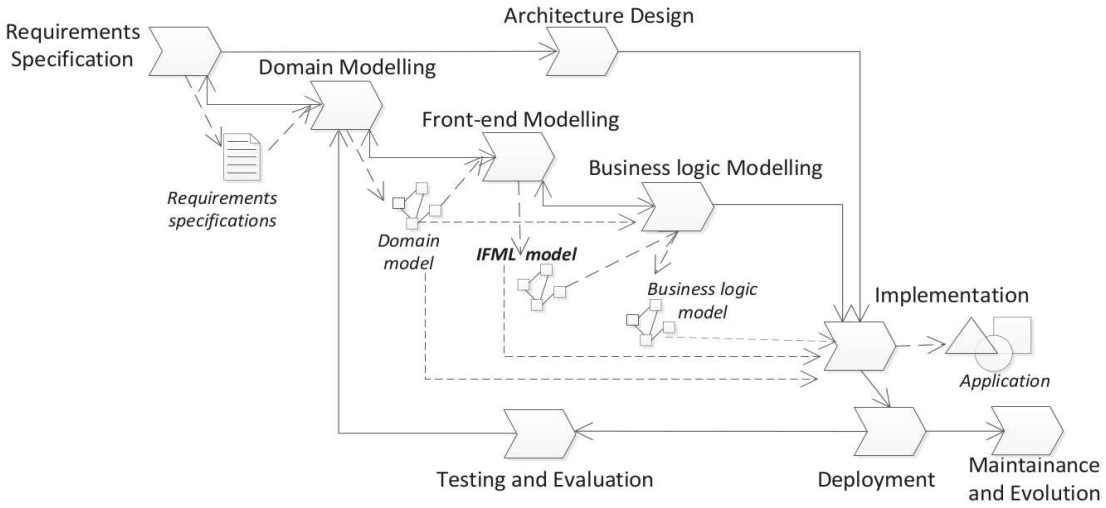

Figura 2.3. El rol de IFML en el proceso de desarrollo de una aplicación interactiva. [16]

En dicha figura, puede verse que IFML cobra particular relevancia a partir de la etapa de modelado de la IGU (Interfaz Gráfica de Usuario). En esta etapa el diseñador puede usar este IFML para especificar la organización del front-end en:

- Vistas y su estructura interna (otras vistas y/o componentes) y,
- Eventos expuestos por cada vista y como éstos disparan acciones de la lógica de negocios y actualizan la interfaz.

2.5. HERRAMIENTAS DE TRANSFORMACIÓN M2M – M2T

El desarrollo del prototipo propuesto como solución utilizará un conjunto de componentes de software integrados a EMF(*Eclipse Modeling Framework*)[5], el cual incluye soporte para:

- El uso de lenguajes de modelado como EMOF (*Essential MOF*) el cuál es parte de la especificación MOF[6] 2.0 de la OMG.
- El uso de lenguajes de transformación de modelo a modelo (en particular, ATL (*ATL Transformation Language*)[7]) y modelo a texto (en particular, Acceleo[8]).

Respecto a los lenguajes de transformación Modelo a Modelo (M2M) se utilizará ATL.

5 https://www.eclipse.org/modeling/emf/
6 http://www.omg.org/mof/
7 https://eclipse.org/atl/
8 https://www.eclipse.org/acceleo/

ATL, según [48] ,es un lenguaje de transformación de modelos que permite obtener un conjuntos de modelos destino a partir de un conjunto de modelos fuente. Un programa de transformación ATL está compuesto de reglas que definen como se aparean los elementos del modelo fuente con el fin de crear e inicializar los elementos de los modelos destino.

En ATL existen 2 tipos de reglas: reglas *matched* y *lazy*. La principal diferencia es que las primeras no pueden (y no necesitan) ser invocadas explícitamente. La invocación de éstas reglas se produce cuando el *parser* de ATL encuentra un modelo origen que es entrada a una regla de transformación dada. Las segundas, las reglas *lazy,* necesitan de una invocación explícita para ser ejecutadas.

Existe un IDE (*Integrated Development Enviroment*) construido sobre la plataforma Eclipse que provee una serie de herramientas de desarrollo como *debuggers*, resaltado de sintaxis, que facilitarán el desarrollo de las transformaciones ATL. En la lectura del presente trabajo, es importante comprender cuáles son y para que sirven las llamadas unidades ATL. Dichas unidades son:

- Módulos: permiten especifican las operaciones M2M.
- Queries: permiten computar valores primitivos -tales como strings o enteros - desde los modelos de origen.
- Libraries: son las únicas unidades ATL que pueden ser importadas desde otras unidades ATL, incluidas las libraries en si mismas.

Por otro lado, respecto a los lenguajes de transformación Modelo a Texto (M2T) es importante destacar que aunque se puede transformar manualmente los modelos abstractos a código, el poder real de MDD proviene de automatizar este proceso [44]. Estas transformaciones automáticas no solo aclerarán el proceso sino que también resultarán en código de mejor calidad ya que las transformaciones podrán capturar las buenas prácticas, la conformidad a estándares y/o a guías de estilo y asegurar que el proyecto de desarrollo las emplee de manera consistente. Además, el código generado:

- Simplifica la mantenibilidad dada la separación existente entre el modelado de la aplicación y el código generado [49].
- Reduce la probabilidad de cometer errores dado que, por ejemplo, permite la implementación de cambios en la aplicación mediante la actualización del modelo sin necesidad de acceder al sistema de archivos u otros recursos del sistema: solo el código generado accede a las librerías del sistema. Esto resulta una idea fundamental en servicios PaaS (*Platform as a Service*).

2.6. PHP (PHP HYPERTEXT PROCESSOR)

Considerando que las plataformas destino de la generación del código deben cumplir con la condición de ser ampliamente extendidas, se analizó la tendencia actual de uso de lenguajes de programación que se emplean en el desarrollo Web.

Dicha tendencia puede observase en la Figura 2.4 y la Figura 2.5, donde se ve claramente la popularidad del lenguaje PHP, a nivel mundial y a nivel país, respectivamente.

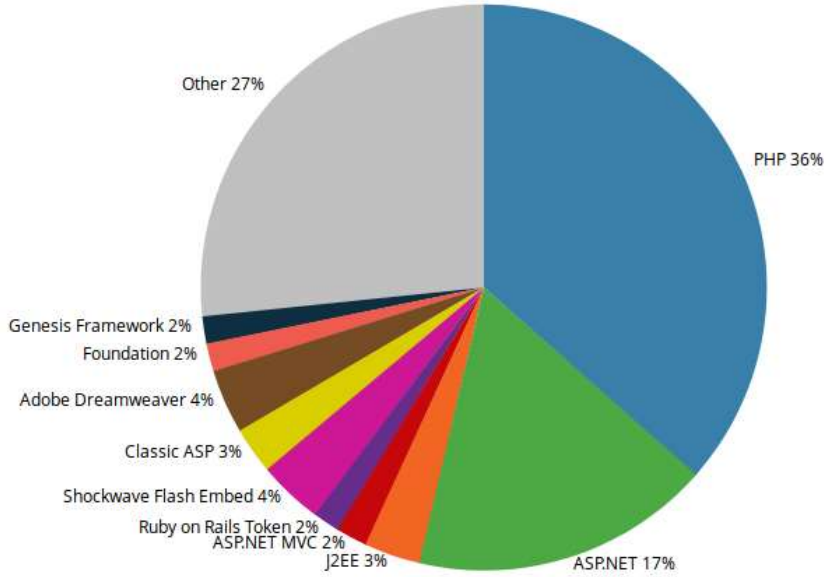

Figura 2.4. Estadística de lenguajes usados mundialmente para programación web a octubre del 2017 [79]

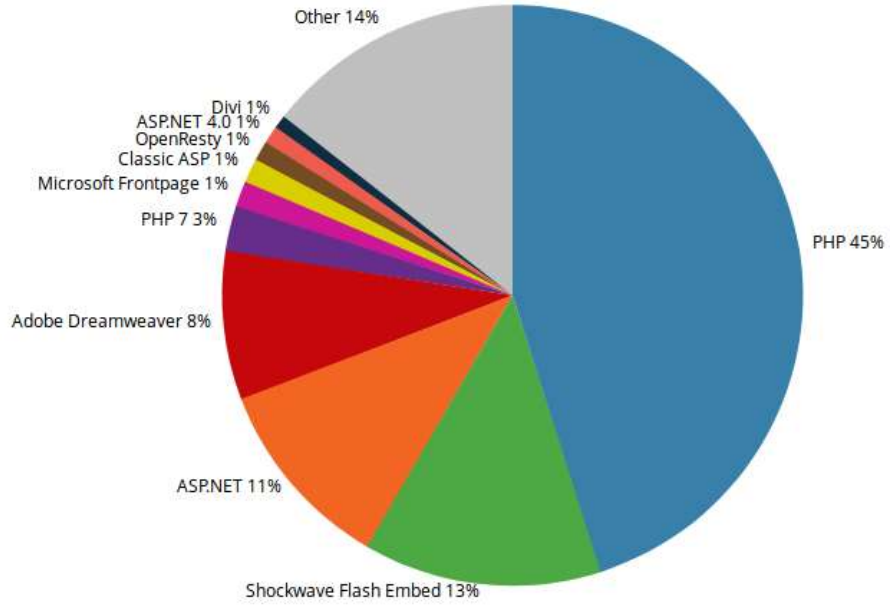

Figura 2.5. Estadística de lenguajes usados en Argentina para programación web a octubre del 2017[74]

Para implementar el modelo IFML, se consideró la utilización de **PHP**, un lenguaje de programación nacido en 1995 para el desarrollo Web, que se ejecuta del lado del servidor - cf.: [50].

Durante la implementación realizada con PHP, la lógica de negocios se entremezcla con las consultas a las bases de datos y las etiquetas de presentación de la interfaz gráfica. A causa de estas mezclas se dificulta el mantenimiento y la escalabilidad del desarrollo [51]. Como respuesta a estos problemas, (y muchos otros) nacen los ***frameworks* de desarrollo**.

A pesar de su extendida utilización, PHP cuenta con varias y, no menos extendidas, desventajas, como por ejemplo, parte de los nombres de su librería estándar son inconsistentes como se puede apreciar en la Tabla 2.1.

Tabla 2.1. Ejemplos de Inconsistencias en la librería estandar de PHP.

Inconsistencia	Ejemplo

Guión / No guión	`strpos/str_rot13`
	`php_uname/phpversion`
	`base64_encode/urlencode`
	`gettype/get_class`

“To” / “2”	`ascii2ebcdic`, `bin2hex`, `deg2rad`, `strtolower`, `strtotime`
Objeto + verbo / verbo + objeto	`base64_decode`, `str_shuffle`, `var_dump`, `create_function`, `recode_string`
Orden de los parámetros	`array_filter($in,$callback)` / `array_map($callback, $in)` `strpos($haystack, $needle)` / `array_search($needle, $haystack)`
Falta de Generalidad	Funciones que ordenan: `array_multisort`, `arsort`, `asort`, `ksort`, `krsort`, `natsort`, `natcasesort`, `sort`, `rsort` ,`uasort`, `uksort`, `usort`. Funciones que encuentran texto: `ereg`, `eregi`, `mb_ereg`, `mb_eregi`, `preg_match`, `strstr`, `strchr`, `stristr`, `strrchr`, `strpos`, `stripos`, `strrpos`, `strripos`, `mb_strpos`, `mb_strrpos`

Los *frameworks* de desarrollo PHP ayudan a lidiar con estos inconvenientes del lenguaje.

2.6.1. Frameworks De Desarrollo PHP

Los *frameworks* de desarrollo son una colección de clases concretas y abstractas pensadas para ser adaptadas y extendidas con el fin de crear aplicaciones [29] que encapsulan patrones arquitectónicos y de diseño y otras implementaciones para solucionar diferentes problemas. Un *framework* también encapsula principios y paradigmas de programación comunes como, por ejemplo, DRY (*Don’t Repeat Yourself)[52]*, KISS (*Keep It Simple, Stupid)[53]* o CoC (*Convention over Configuration)[54]*.

Entonces, podemos decir que los *frameworks* ofrecen un conjunto de elecciones que facilitan la continua toma de decisiones que implica el desarrollo, pero además:

- Incrementan la seguridad a partir de que los usuarios se convierten en *testers* a largo plazo. Si un usuario encuentra un problema puede notificarlo a los desarrolladores del *framework* así éstos pueden solucionarlo.
- Reducen el costo del desarrollo dado que generalmente son *Open Source* y ayudan al desarrollador a escribir código más rápidamente.
- Usualmente los *frameworks* también implican la existencia de un grupo de soporte que lo impulsa, documentación y foros donde se pueden conseguir respuestas con relativa rapidez.

Los *frameworks* de desarrollo Web, a su vez, generalmente están construidos siguiendo un **patrón arquitectónico MVC (Model-View-Controller / Modelo-Vista-Controlador)**. Este patrón separa los elementos del sistema en tres componentes lógicos que interactúan entre sí de la manera representada en la Figura 2.6, donde:

- *Model* (Modelo) se encarga del tratamiento de los datos.
- *View* (Vista) maneja como se muestran los datos al usuario
- *Controller* (Controlador) es el componente intermedio que se encarga de recolectar la información desde *Model* y de hacer los datos entendibles antes de enviárselos a *View*.

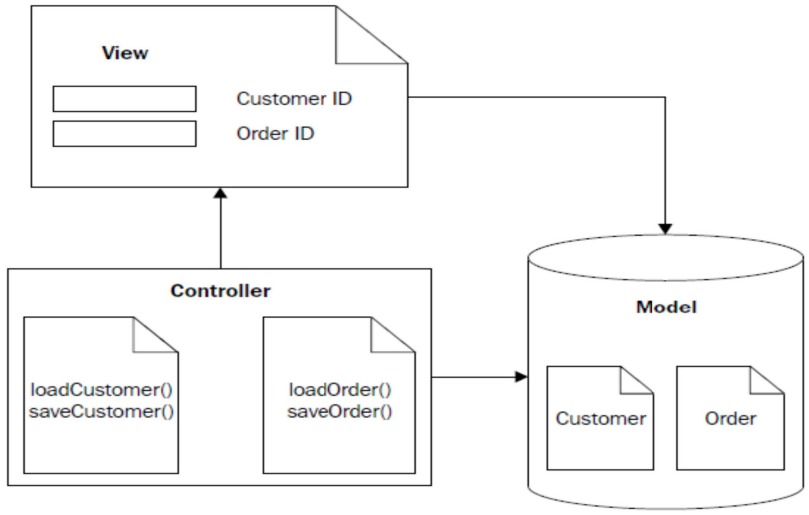

Figura 2.6. Ejemplo de una implementación usando el patrón arquitectónico MVC. [78]

Se han realizado varios estudios sobre el uso de éstos *frameworks* PHP MVC contra PHP plano. Uno de estos es el caso de [55], el cuál está enfocado en el desempeño como atributo de calidad. En él, se consigna que el uso de PHP plano presenta una leve ventaja respecto a los *frameworks* PHP MVC, en tiempo y consumo de memoria, para proyectos pequeños. Sin embargo, para proyectos de mayor envergadura, la diferencia en el uso o no de un *framework* es notable: se inclina a favor del uso de éstos debido a que implican un crecimiento menor en la pila de seguimiento de ejecución.

Además, en cuánto al atributos de mantenibilidad, ordenar las consultas en *Model*, el codigo HTML en *View* y la lógica de negocios en *Controller,* mejoraría este atributo considerablemente. En la Figura 2.7 se presenta una ilustración de ello: a la izquierda, un ejemplo de estructura de código en PHP plano (es decir, codificación sin utilización de *frameworks*) y a la derecha la estructura que tendría ese código si se utilizará dentro de un *framework* PHP MVC Web.

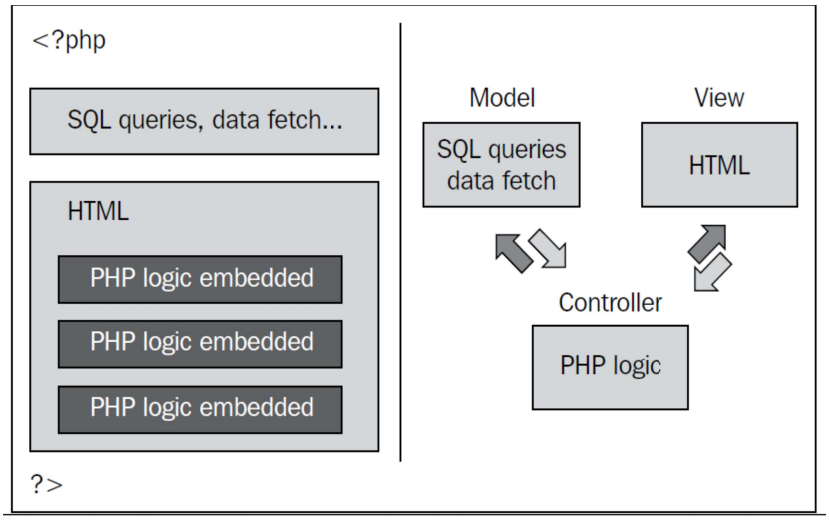

Figura 2.7. Ilustración comparativa de la estructuración del código: con PHP Plano y con framework PHP[75]

La elección de los *frameworks* PHP MVC que se utilizarán responde a su popularidad, es decir, a su grado de aceptación en la comunidad global de desarrolladores.

A partir de lo que se presenta en la Figura 2.8, puede verse que CodeIgniter mostró ser la herramienta favorita entre los años 2011 hasta mitad del 2015 cuando surge Laravel para volverse el *framework* más usado en todo el mundo en poco tiempo. Los *frameworks* mencionados, junto a Yii2, son lo más utilizados en el momento en que este documento está siendo redactado y por lo tanto consideramos que serían los más factibles de ser estudiados.

Teniendo en cuenta, entonces, la relevancia que se desea que tenga el modelo intermedio resultante, las plataformas de destino de la generación de código incluyen pero no se limitan a los *frameworks* Laravel y Yii2.

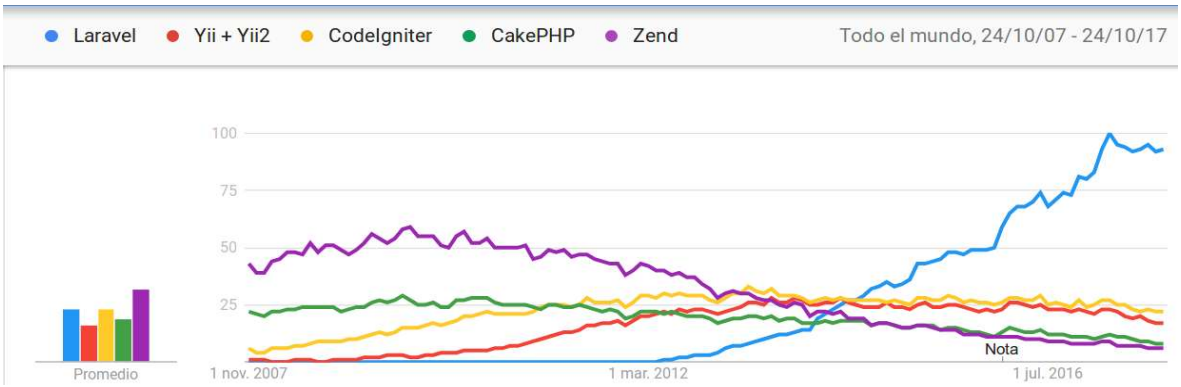

Figura 2.8. Tendencias de búsqueda de frameworks PHP entre 2007-2017 [76]

- Yii2 [56]: Yii2 es un *framework* que puede ser empleado para el desarrollo de todo tipo de aplicaciones Web usando PHP. Por su arquitectura basada en componentes (*component-based*) y soporte de almacenamiento en caché sofisticado, es especialmente adecuado para el desarrollo de aplicaciones de gran escala como

portales, foros, sistemas de administración de contenido (CMS), proyectos *e-commerce*, *RESTful Web Services*, entre otros.

- Laravel [57]: Según su sitio Web oficial, es un *framework* Web PHP con sintaxis elegante y expresiva. Busca mejorar el desarrollo facilitando las tareas comunes utilizadas en la mayoría de los proyectos Web, como son la autenticación, el ruteo, el manejo de sesiones, el caché y el manejo de plantillas, esto último usando un motor llamado Blade.

2.7. CONSIDERACIONES FINALES SOBRE EL ESTADO DEL ARTE

- El desarrollo guiado por modelos constituye un paso importante en la evolución de la Ingeniería del Software, en particular, la Ingeniería Web. Los medios utilizados para describir lo que el software debe hacer: conversaciones, descripciones coloquiales, dibujos en un papel o en un pizarrón son diferentes de los medios en los que se realiza la implementación de esas ideas y por ello el desarrollo guiado por modelos permite una disminución en esta brecha.
- El desarrollo guiado por modelos también permite poner el foco en un concepto a veces olvidado: la trazabilidad. Esta característica es importante en el software, sobre todo en sistemas críticos donde resulta deseable poder mapear cada requerimiento con el código que lo implementa. Con suficiente trazabilidad, es posible demostrar que cuando cierto código cambie -por un cambio en los modelos, para nuestro paradigma- sigue satisfaciendo los requerimientos y que cuando algo no funcione, entender la razón del porqué debería ser.
- El surgimiento del estándar IFML no es el final del camino, pero sí representa un hito importante al ser el primer proceso de desarrollo MDWE estándar de la OMG: queda mucho trabajo por hacer en vistas a asegurar que las aplicaciones Web modernas puedan ser desarrolladas más rápidamente, con menos errores y mayor calidad.
- Aunque los lenguajes de transformación están bastante maduros, necesitan mejoras: depuradores, *frameworks* de *testing* automatizado, *linters, frameworks* que permitan la integración continua de las reglas de transforación -como TravisCI[9]- y herramientas de profiling -en el caso de ATL. También es necesario avanzar sobre teorías que nos permitan razonar sobre transformaciones y reglas, poder comprender como se diferencian los conceptos específicos como modularidad, herencia o extensibilidad del desarrollo de software tradicional.
- EMF provee una de las maneras más sencillas y pragmáticas que, en la actualidad, permiten la definición de metamodelos, generación automática de editores del mismo, y extensiones livianas de UML. Esta propuesta de Eclipse permite integrar fluidamente transformaciones M2M y M2T y se constituye en una base poderosa, aunque aún insuficiente, para la ingeniería dirigida por modelos.

9 https://travis-ci.org/

3. METAMODELO PROPUESTO

En este capítulo se presenta el metamodelo que se propone, incluyendo algunas de las diversas decisiones que se tomaron, como por ejemplo, respecto a las abstracciones a realizar y los *frameworks* de desarrollo que se utilizaron como plataforma tecnológica de destino. Además, se incluye una descripción pormenorizada de cada clase incluida en el metamodelo.

Salvo que se indique lo contrario, todas las clases y enumeraciones referencian a otras clases y/o enumeraciones del mismo paquete. Las restricciones (c*onstraints)* de cada clase están expresadas en lenguaje OCL (*Object Constraint Language*).

3.1. PROPUESTA DE SOLUCIÓN

Se cree que se podría lograr una mayor adopción de metodologías, en el contexto de MDWE, en la medida que se cuente con un modelo intermedio de diseño que abstraiga características comunes entre plataformas y brinde así soporte a plataformas de destino avanzadas y de amplia aplicación en la construcción de sistemas Web actuales.

Asimismo, tal posibilidad de extensión de un enfoque MDWE resalta una de las ventajas de su adopción: el hecho de poder reutilizar modelos del dominio portando semiautomáticamente sistemas Web complejos a otras plataformas, cuando aquella elegida originalmente para su implementación se considere ya obsoleta.

Podría demostrarse la utilidad y factibilidad de realizar las transformaciones que lleven desde los modelos de un proceso MDWE hasta código ejecutable para *frameworks* de desarrollo PHP MVC Web. En el presente TFC (Trabajo Final de Carrera), el proceso de desarrollo MDWE será IFML y los *frameworks* de desarrollo PHP MVC Web Yii2 y Laravel.

En la elaboración de la solución que permita el cumplimiento de los objetivos de este proyecto se seguirá un esquema como el presentado en la Figura 3.1. Donde las flechas entre modelos representan transformaciones entre los mismos.

Dicha solución debe asegurar la correcta generación de:

- Modelos que se puedan crear, leer, actualizar y borrar.
- Vistas que incluyan componentes básicos: botones, anclas, textos, imágenes, formularios y entradas de texto mediante *radiobuttons, textfields y checkboxes*.
- Controladores que aseguren la navegabilidad entre las distintas partes de la aplicación Web.

Todos representados con el nivel de abstracción adecuado como para no restringir la extensibilidad a *frameworks* de otras plataformas o incluso la integración con metamodelos de otros procesos de desarrollo Web. Dicha extensión permitiría no solo lograr un metamodelo más completo, sino también un metamodelo cuyas abstracciones estén extensamente validadas.

Se busca que el prototipo desarrollado como solución, una vez terminado, resulte útil para:

- La verificación del metamodelo propuesto, las transformaciones resultantes y las abstracciones supuestas en dicho metamodelo.

Con el fin de lograr ésta verificación se trabajará sobre un caso de estudio, que será desarrollado en dos fases. La primera fase implicará la creación de un sitio Web informacional y estático (Web 1.0). La segunda, la creación de una aplicación Web interactiva (Web 2.0), basado en la clasificación de sitios web propuestas en: [58], [59].

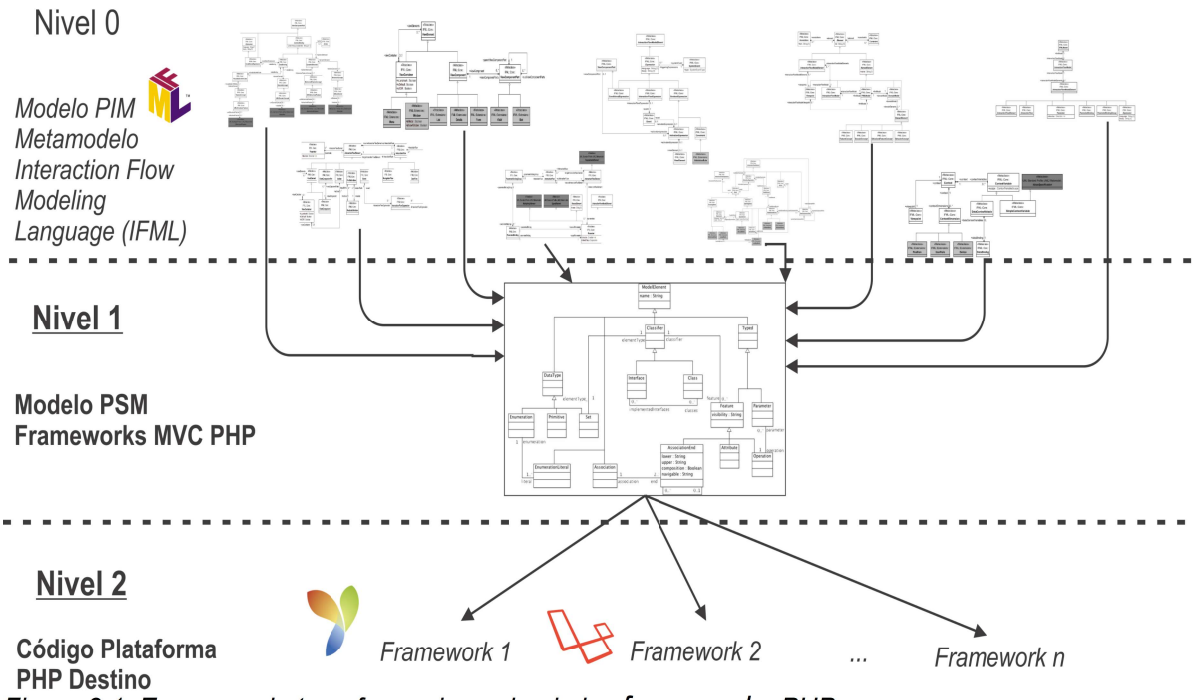

Figura 3.1. Esquema de transformaciones hacia los frameworks PHP.

Fase de creación de un sitio Web estático

El resultado de esta fase será un sitio Web que consistirá en una colección jerárquica de documentos HTML estáticos que ofrecen información de solo-lectura.

El código generado para tal fin deberá incluir:

- Vistas que contengan componentes visuales básicos como botones, anclas, textos e imágenes.
- Controladores que aseguren la navegabilidad entre las distintas secciones del sitio

Los *concerns* propuestos por IFML e involucrados en la elaboración de este nivel de prueba son los de: *IFML Model*, *Interaction Flow Model*, *Interaction Flow Elements*, *View Elements*, *Content Binding*, *Specific View Components* y *Modularization*. [46]

Fase de creación de una aplicación Web orientada a transacciones

El propósito de esta fase es el cubrimiento total de los requerimientos definidos para un sitio Web estático pero, además, permitirá:

- Vistas que incluyan formularios y entradas de texto mediante *radiobuttons, textfields y checkboxes.*
- Modelos del dominio que se puedan crear, leer, actualizar y borrar tanto durante el transcurso de la sesión como en una base de datos.

Los *concerns* propuestos por IFML e involucrados en la elaboración de este nivel de prueba son, no solo los mencionados como necesarios para la generación del sitio Web estático (*IFML Model*, *Interaction Flow Model*, *Interaction Flow Elements*, *View Elements*, *Content Binding*, *Specific View Components* y *Modularization*) sino también los de (*Parameters*, *Events*, *Expressions* y *Context*). De esta manera, se cubrirían todos los *concerns* propuestos por IFML.

Además, con el fin de describir la calidad de la propuesta y considerando que dicha calidad sea la esperable de un futuro producto se evaluarán el caso de estudio terminado considerando medidas y métricas orientadas a metamodelos y objetos.

3.2. PROCESO DE CONSTRUCCIÓN DEL METAMODELO

Con el fin de descubrir abstracciones que constituyan un metamodelo PHP MVC Web se consideró el estudio de la organización arquitectónica subyacente de 3 *frameworks* que cumplan esas características, a saber:

- Laravel, en su versión 5.4.15
- Yii en su versión 2.0.12
- CodeIgniter en su versión 3.1.5

Con el fin de realizar las abstracciones que constituyen el metamodelo, cada uno de los autores de este documento dedicó un mes completo, de forma individual, para el estudio de los 3 *frameworks* en cuestión.

Luego, se pusieron en común cuáles eran las similitudes y diferencias que se encontraron entre cada *framework*, poniendo énfasis en las primeras y comenzando a diagramar esas ideas.

El resultado de este proceso de construcción fue un primer metamodelo que fue presentado como parte del informe de Práctica Profesional Supervisada de los autores y constituye el anexo D de este trabajo. El metamodelo presentado en este documento es un subconjunto de aquél, más abstracto y de mayor pragmatismo y por ende más cerca de su verificación empírica.

Dos hipótesis fueron descartadas durante este tarea:

- La necesidad de proponer un metamodelo por cada *framework* y a partir de allí realizar las abstracciones: no fue necesario proponer un metamodelo de cada *framework* a partir de los cuáles realizar las abstracciones ya que al nivel de abstracción trabajado los 3 *frameworks* terminaban siendo muy similares, a pesar de definir ideas parecidas de maneras distintas.
- El tiempo dedicado al estudio de los *frameworks:* un mes completo para los 3 *frameworks*, debería ser mayor. El tiempo dedicado al estudio de los *frameworks* resultó suficiente ya que como los 3 ***frameworks*** están basados en un mismo patrón arquitectónico y pertenecen a un mismo dominio (Web) las abstracciones fueron surgiendo de manera casi natural.

3.3. DECISIONES DE DISEÑO

3.3.1. Arquitectura Elegida

El modelo intermedio propuesto debería ser construido respetando una arquitectura MVC Web, no solo porque sea la arquitectura subyacente en la mayor parte de los *frameworks* Web, sino que también porque la adopción de este patrón resulta natural cuando se tiene en mente, por ejemplo, que:

- Un controlador que maneje las *requests* y seleccione las páginas apropiadas evitaría un acoplamiento inmediato entre la *request* hecha por el usuario y la página resultante. Lo cuál es útil no solo cuando la navegación se hace compleja sino también para acciones comunes como la autenticación y el manejo de sesiones.
- Las vistas de la aplicación Web es lo que más suele cambiar durante el desarrrollo de ésta, de manera que separarla de la lógica de negocios disminuye la probabilidad de introducir *bugs.*

A pesar de estas ventajas, dicha arquitectura debe lidiar con algunas limitaciones, como:

- A veces se hace particularmente difícil dividir el Modelo del Controlador, es decir, comprender a dónde debe ir una pieza de código en particular.
- Salvo que se tenga consideraciones especiales (como la diferenciación de modelos de Dominio y de Aplicación) el acoplamiento entre la Vista y el Modelo puede no resultar ideal cuando se considera la flexibilidad de la aplicación.
- En algunos tipos de aplicaciones, como aquellas que resulten muy sencillas, el patrón introduce una sobrecarga innecesaria.

3.3.2. Consideraciones Al Metamodelo Propuesto

Dado que el patrón arquitectónico MVC no es particular del dominio Web, en el sentido de que puede ser aplicado en otros dominios, se decidió que sería conveniente distribuir los elementos del metamodelo en 2 paquetes.

- Un paquete MVC compuesto por los elementos que caracterizan este patrón.
- Un paquete HTML que contenga elementos que extiendan los conceptos del paquete MVC.

El detalle acerca de las metaclases y enumeraciones que constituyen el metamodelo puede encontrarse en el Anexo C.

3.3.3. Vista Del Metamodelo

Una representación en diagrama de paquetes del metamodelo puede verse en la Figura 3.2. Mientras que en la Figura 3.3 puede verse una porción en vista de árbol de la representación Ecore de este metamodelo.

Finalmente, las anteriores ilustraciones complementa al diagrama de clases que puede apreciarse en la Figura 3.4.

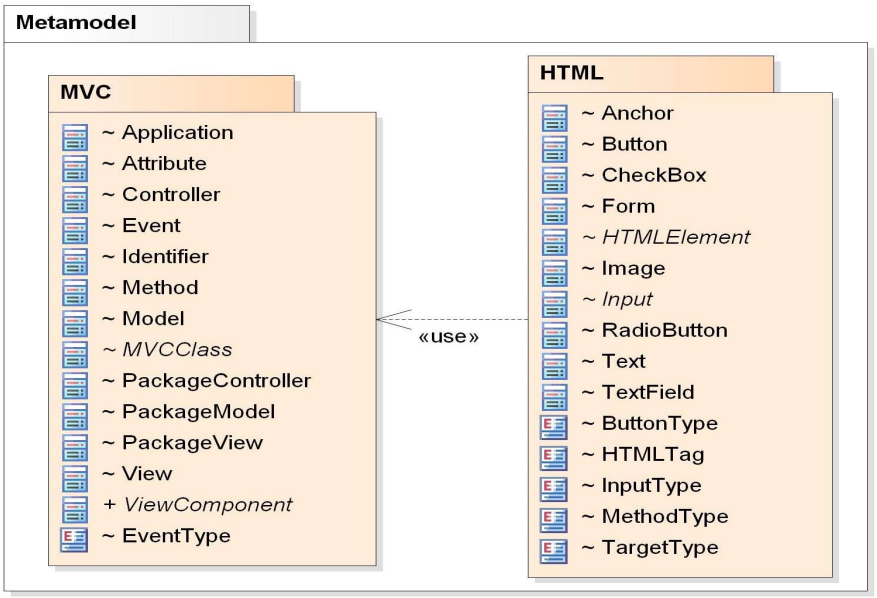

Figura 3.2. Diagrama de Paquetes del metamodelo propuesto

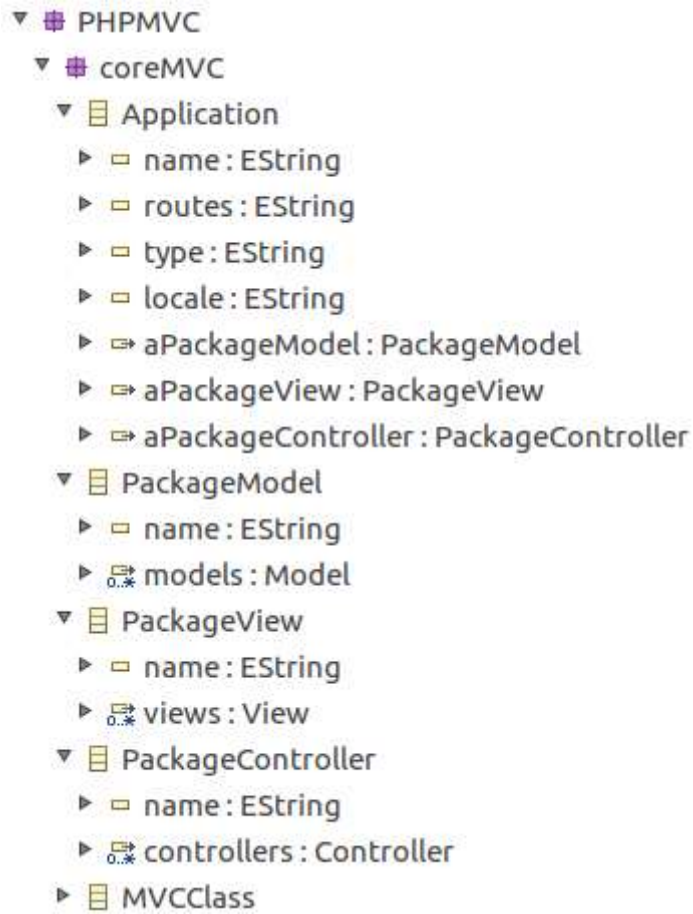

Figura 3.3. Porción de la presentación en vista de árbol del metamodelo en formato Ecore.

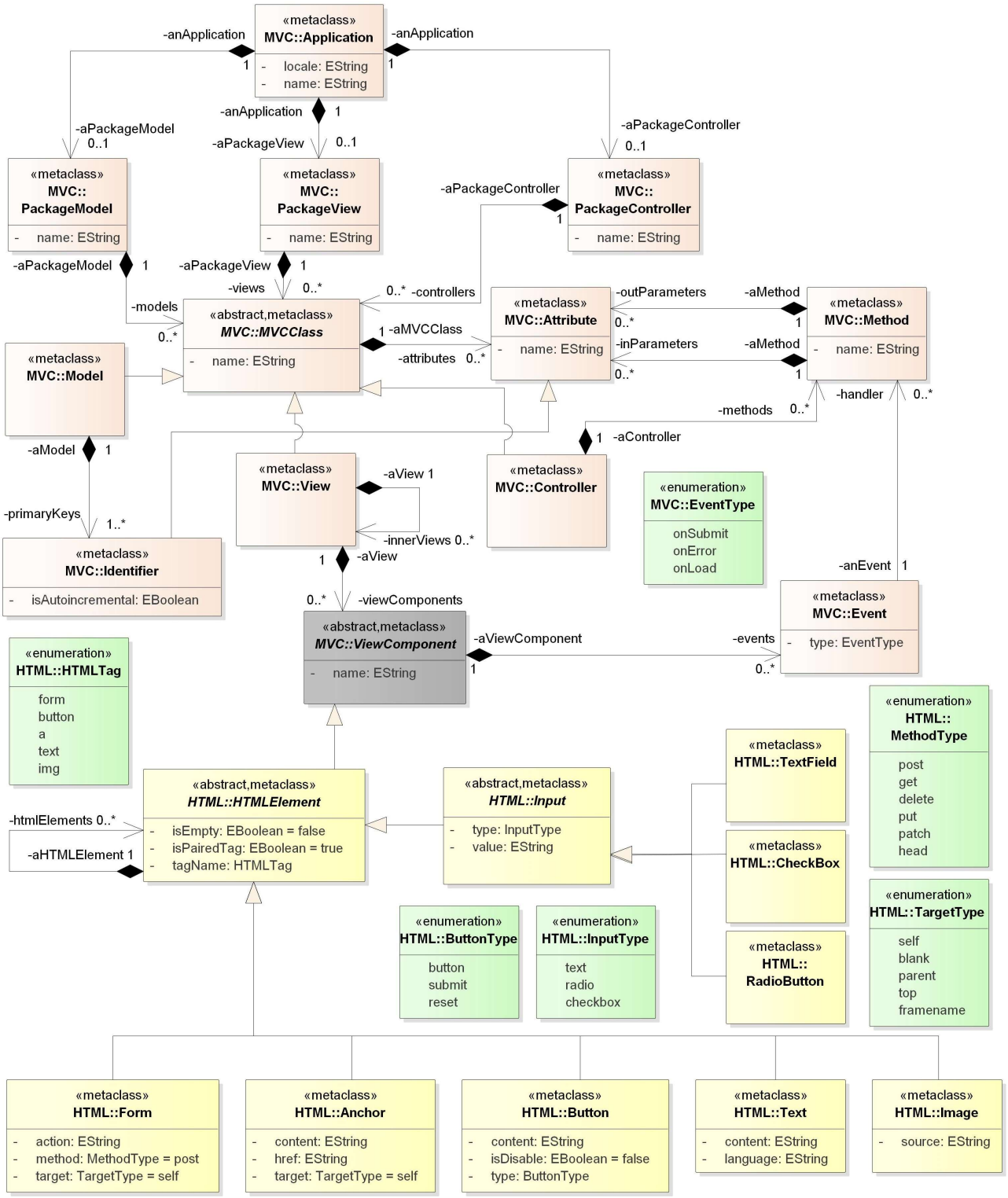

Figura 3.4 Diagrama de Clases del metamodelo propuesto

3.4. PAQUETE MVC

A continuación, en la Tabla 3.1 y en la Tabla 3.2 se presentan las metaclases y las enumeraciones, respectivamente, que pertenecen al Paquete MVC.

Tabla 3.1. Metaclases del paquete MVC del metamodelo

Nombre	Atributos	Descripción
Application	name : EString locale : EString	Una Application encapsula aquellos atributos que son propios de la aplicación. Se pueden tener varias aplicaciones compartiendo la misma instalación del *framework*. Los *frameworks* permiten que los atributos de esta clase sean globalmente accesibles.
Attribute	name : EString	Un Attribute representa tanto las propiedades de Modelos, Vistas y Controladores, como parámetros de los Métodos.
Controller	-	Un Controller agrupa la lógica de manejo de *requests* en clases. El nombre del controlador forma parte de una URI (*Universal Resource Identifier*) que permite tener acceso a ellos, directamente o a través de las rutas definidas en el *framework*.
Event	type : EventType	Un Event es el resultado de un acción del navegador o del usuario.
Identifier	isAutoincremental : EBoolean	Los Identifiers son aquellos atributos que serán claves primarias del modelo de datos.
Method	name : EString	Los Methods son los procedimiento asociado al comportamiento de los objetos de la aplicación.
Model	-	Los Models son las clases base de los modelos de datos. destinadas a trabajar con la información de la base de datos.
MVCClass	name : EString	Una MVCClass encapsula las propiedades y el comportamiento en común que puedan tener sendos componentes Model-View-Controller.
PackageController	name : EString	Un PackageControler reagrupa a todos los controladores. Esto se traduce, por ejemplo, en el nombre de una carpeta que los contenga.

Nombre	Atributos	Descripción
PackageModel	name : EString	Un PackageModel reagrupa a todos los modelos. Esto se traduce, por ejemplo, en el nombre de una carpeta que los contenga.
PackageView	name : EString	Un PackageView reagrupa a todas las vistas. Esto se traduce, por ejemplo, en el nombre de una carpeta que los contenga.
View	-	Las Vistas son las responsables de presentar los datos a los usuarios finales. Por lo general son creadas en función a *view templates.* Nunca se cargan directamente sino que siempre son invocadas a través de un Controller.
ViewComponent	name : EString	Un ViewComponent es todo elemento que se renderiza como un componente visual en la interfaz gráfica. Esta clase, al igual que todas las demás de este paquete, no depende de ninguna tecnología en particular. Un ViewComponent podría, por ejemplo, realizarse en un elemento HTML o un componente Java Swing.

Tabla 3.2: Enumeraciones del paquete MVC del metamodelo.

Nombre	Literales	Descripción
EventType	onError onLoad onSubmit	Se enumera aquí un pequeño subconjunto de los eventos definidos en [60].

3.5. PAQUETE HTML

Se presentan a continuación en la Tabla 3.3 y en la Tabla 3.4 as metaclases y enumeraciones, respectivamente, que se encuentran contenidas en el Paquete HTML.

Tabla 3.3. Metaclases del paquete HTML del metamodelo.

Nombre	Atributos	Descripción
Anchor	content : EString href : EString target : TargetType	Los *Anchor* son porciones de texto que marcan el inicio o el final de un hiperenlace. Ayudan a mantener la organización de una pagina, permitiendo al usuario encontrar mas rápidamente lo que necesita.

Nombre	Atributos	Descripción
Button	content : EString isDisable : EBoolean type : ButtonType	Esta clase define un botón. El comportamiento de dicho botón al ser activado, es controlado por el atributo *type*.
CheckBox	-	Es el componente gráfico HTML que permite al usuario hacer selecciones múltiples sobre un conjunto de opciones.
Form	action : EString method : MethodType target : TargetType	Contenedor que agrupa una serie de datos para su envío a través de *requests*. Soporta algunos atributos específicos para configurar su comportamiento.
HTMLElement	isEmpty : EBoolean isPairedTag : EBoolean tagName : HTMLTag	Un HTMLElement es un componente individual del documento HTML que una vez parseado pasa a formar parte del DOM (*Document Object Model*).
Image	source : EString	Define una imagen en un documento HTML.
Input	type : InputType value : EString	Esta clase especifica los campos de entrada a través de los cuales los usuarios pueden ingresar datos.
RadioButton	-	Elementos renderizados como pequeños iconos circulares que permiten realizar una selección única sobre un conjunto de opciones.
Text	content : EString language : EString	Representa una porción de texto.
TextField	-	Este tipo de Input renderiza en una caja de texto que permite a los usuarios ingresar texto en una sola linea.

Tabla 3.4. Enumeraciones del paquete HTML del metamodelo.

Nombre	Literales	Descripción
ButtonType	button reset submit	Definen el comportamiento del botón una vez que ha sido activado.
HTMLTag	a button form img text	Encapsula las etiquetas que son los bloques del lenguaje HTML.
InputType	checkbox radio text	Especifica el tipo de datos y el control asociado a un elemento input dado.
MethodType	delete get head	Especifica el método HTTP con el cual serán enviados los *request* de una pagina a otra.

Nombre	Literales	Descripción
	patch post put	
TargetType	black framename parent self top	Especifica donde se abrirá un documento HTML.

3.6. ACLARACIONES

Cabe mencionar que en la especificación del metamodelo propuesto en EMF, a todas las enumeraciones presentadas anteriormente se les agregó como primer valor un literal denominado *"workaround"* -palabra utilizada con frecuencia para definir un desvío a un problema reconocido en un programa. Esto se debe a que, EMF no serializa el valor por defecto de la enumeración, es decir, su primer literal.

Es de destacar que dicha situación no se considera como un problema en la documentación de la herramienta, pese a que varios usuarios de la comunidad lo han reportado[10]. En el marco del presente trabajo se desarrolló un *script* que utiliza *bash* y *sed* para solucionar este problema de manera parcialmente automatizada. Se dice parcialmente ya que el usuario necesita descargar y ejecutar el script en lugar de poder integrarlo a su *workflow* de manera transparente. Dicho *script* se encuentra alojado en el repositorio del proyecto.

10 https://www.eclipse.org/forums/index.php/t/26267/

4. DISEÑO E IMPLEMENTACIÓN

El siguiente capítulo se divide en dos partes principales. En la primera, se definen las transformaciones modelo a modelo, para lograrlo, se especifica el algoritmo de transformación entre los elementos del metamodelo de IFML y los elementos del metamodelo propuesto, justificando el porqué de cada transformación realizado, tales transformaciones se llevan a cabo con ATL.

En la segunda parte, se presenta la transformación modelo a código utilizando Acceleo. A su vez, esta se encuentra divida en dos subpartes, donde se detalla la transformación hacia cada uno de los *frameworks* elegidos -Laravel y Yii2-. El anexo E presenta una guía respecto a como replicar todo el entorno de desarrollo utilizado.

4.1. CONVENCIÓN DE NOMBRES UTILIZADOS

Una buena práctica a seguir al trabajar en un proyecto en equipo, actualizar un antiguo proyecto o buscar colaboración poniendo a disposición un proyecto en un repositorio, es seguir convenciones de nombres existentes Seguir con una convención de nombres consistentes a lo largo del proyecto facilita su mantenibilidad. Una convención de nombres inconsistentes inevitablemente causa confusión. Por estas razones, los proyectos M2M y M2T de este trabajo siguen la convención de nombres usada para plugins de Eclipse[11].

La Tabla 4.1 presenta los nombres de cada proyecto siguiendo esta convención. ”lycmm” significa **L**aravel-**Y**ii2-**C**odeIgniter **M**eta-**M**odel.

Tabla 4.1. Nombres de los proyectos construidos

Proyecto	Nombre
M2M: ATL	edu.ifml2php.pim.ifml.gen.lycmm
M2T: Laravel	edu.ifml2php.psm.lycmm.gen.laravel
M2T: Yii2	edu.ifml2php.psm.lycmm.gen.yii2

4.2. TRANSFORMACIONES PIM A PSM

4.2.1. Modulo Principal De Transformación M2M

El código ATL producido se encuentra en el Anexo F y, al igual que todo el resto de la implementación, en el repositorio GitHub[12] del proyecto.

- **Metamodelos de Entrada**

Además del metamodelo que se propone en el presente trabajo (de ahora en más, PHPFW), para esta primera transformación son necesarios los metamodelos que se presentan en la Tabla 4.2.

11 https://www.obeo.fr/fr/acceleo-best-practices/406-acceleo-best-practices
12 https://github.com/Dipiert/ifml2php

Tabla 4.2. Metamodelos necesarios para la transformación M2M.

Nombre del Metamodelo	URI
UMLMM	http://www.eclipse.org/uml2/5.0.0/UML
IFMLMM	http://www.omg.org/spec/20130218/core
EXTMM	http://www.omg.org/spec/20130218/ext

- El metamodelo de UML es necesario para la interpretación de los modelos de UML.
- El metamodelo core de IFML es necesario porque es quien contiene los conceptos sobre los cuáles se construye la infraestructura del lenguaje de modelado en términos de InteractionFlowElements, InteractionFlows y Parameters.
- El metamodelo extension (también llamado ext) de IFML extiende los conceptos del paquete core con conceptos concretos y comportamientos complejos.

- **Modelos de entrada**

Cada modelo de entrada debe estar construido conforme a un metamodelo. En el caso de este trabajo, son necesarios 2: un modelo de dominio -conforme al metamodelo de UML- y un modelo de flujos de interacción -conforme al metamodelo de IFML.

- **Librerías**

Con el fin de aumentar la mantenibilidad del código de las transformaciones ATL se utilizaron librerías. Como quedó dicho brevemente en la Sección 2, ATL no soporta las invocaciones entre módulos ATL, eso hubiera sido deseable para aumentar la modularidad aún más.

Se crearon las librerías **ifmlCoreLibrary**, **ifmlExtLibrary**, **mvcLibrary** y **systemLibrary** compuestas de ATL *helpers*.

- ifmlCoreLibrary

IFMLMM es el contexto de todas las clases. Todas las salidas de los helpers pertenecen a PHPFW, ya sea al paquete MVC o HTML del mismo. En la Tabla 4.3 se muestran las interfaces de los helpers que se encuentran en esta librería.

Tabla 4.3. Interfaces de los helpers de la librería ifmlCoreLibrary.

Contexto	Nombre	Retorno
IFMLAction	getInParameters	Sequence(Attribute)
IFMLAction	getOutParameters	Sequence(Attribute)
ViewContainer	getAnchors	Sequence(Anchor)
ViewContainer	getConditionedViewComponentsParts	Sequence(ViewComponentPart)
ViewContainer	getViewComponents	Sequence(ViewComponent)

Contexto	Nombre	Retorno
ViewContainer	getImages	Sequence(HTLM!Image)
ViewContainer	getForms	Sequence(HTML!Form)
ViewElement	getInputFields	Sequence(Input)
ViewElement	getSelectionField	Sequence(Input)

getInParameters(): Retorna una secuencia de atributos cuya dirección del IFML Parameter asociado, sea “in” o “inout”.

getOutParameters(): Retorna una secuencia de atributos cuya dirección del IFML Parameter asociado, sea “out” o “inout”.

getAnchors(): Retorna una secuencia de Anchor donde cada uno apunta a una de las ViewContainers que haya sido caracterizada como landmark y que no sea el ViewContainer en sí desde donde se está invocando el helper. Se utiliza para poder armar una barra de navegación.

getConditionedViewComponentsParts(): Retorna una lista de ViewComponentsParts que tienen una ConditionalExpression definida.

getViewComponents(): este helper es el encargado de coordinar las actividades de conseguir InputsFields, Images, ViewElements y, ademàs, si es un ViewContainer que no tiene como clase padre otro ViewContainer -es una vista del más alto nivel- se traen también los Anchors utilizados para armar una barra de navegación.

getImages(): Retorna la lista de Images que existen dentro de un ViewContainer obtenidos a partir de los ViewElements.

getForms(): Retorna la lista de Forms que existen dentro de un ViewContainer, obtenidos a partir de los ViewElements.

getInputFields(): Retorna una secuencia de Input realizables en Checkboxes y RadioButtons.

getSelectionFields(): Retorna una lista con todos los Input de selección, a saber, Checkboxes y RadioButtons.

- ifmlExtLibrary

EXTMM es el contexto de todas las clases. Excepto que se indique lo contrario, todas las salidas de los helpers pertenecen a PHPFW. En la Tabla 4.4 se presentan las interfaces de los helpers que se encuentran en esta librería.

Tabla 4.4. Interfaces de los helpers de la librería ifmlExtLibrary.

Contexto	Nombre	Retorno
Field	isCheckBox	Boolean (OCL Primitive Type)
Form	getSimpleFields	TextField
IFMLWindow	getSimpleFields	TextField

isCheckBox(): Retorna *true* si el Field en el contexto del cual se invoca el helper, es multiSelection.

getSimpleFields(): Retorna una lista con todos los SimpleFields definidos en una IFMLWindow o en un Form desde los cuáles se invocare el helper.

- mvcLibrary

PHPFW es el contexto de todas las clases. Excepto que se indique lo contrario, todas las salidas de los helpers pertenecen a PHPFW. En la Tabla 4.5 se introducen las interfaces de los helpers que se encuentran en esta librería.

Tabla 4.5. Interfaces de los helpers de la librería mvcLibrary.

Contexto	Nombre	Retorno
Event	getDefaultEventType	EventType
Event	getHandler	String (OCL Primitive Type)
HTMLElement	getDefaultTarget	TargetType

getDefaultEventType(): Retorna el tipo de evento predeterminado. En este prototipo: “onSubmit”.

getHandler(): Retorna el nombre del controlador que manejará un evento dado. En este prototipo se asume un submit por Form.

getDefaultTarget(): Retorna el tipo de target predeterminado para el Anchor o el Form desde el cual se invoca el helper. En este prototipo: “self”.

- systemLibrary

Todo el contexto de los helpers contenidos en esta librería son clases OCL Primitive Type. En la Tabla 4.6 se menciona la interfaz del helper que se encuentran en esta librería.

Tabla 4.6. Interfaces de los helpers de la librería sistemLibrary.

Contexto	Nombre	Retorno
String	decapitalize	String

decapitalize(): pone en minúsculas la primera letra de un String.

- **Reglas**

En la Tabla 4.7 se listan todas las reglas utilizadas en la transformación M2M por ATL, en ella respecto al tipo de regla, M refiere a reglas Matched y L a Lazy.

Tabla 4.7. Reglas de Transformación ATL.

Regla	From	To	Tipo
ConditionalExpression2Text	IFMLMMM! ConditionalExpression	HTML!Text	M
DomainModel2PackageController	IFMLMM!DomainModel	MVC! PackageController	L
DomainModel2PackageModel	IFMLMM!DomainModel	MVC!PackageModel	M
IFMLAction2Method	IFMLMM!IFMLAction	MVC!Method	M
IFMLForm2Form	ExtMM!Form	HTML!Form	M
IFMLModel2ApplicationClass	IFMLMM!IFMLModel	MVC!Application	M
IFMLSlot2Checkbox	ExtMM!IFMLSlot	HTML!Checkbox	L
IFMLSlot2RadioButton	ExtMM!IFMLSlot	HTML!RadioButton	L
InteractionFlowModel2PackageView	IFMLMM! InteractionFlowModel	MVC!PackageView	M
Landmarks2MenuItems	IFMLMM!ViewContainer	HTML!Anchor	L
OnSubmitEvent2Event	ExtMM!Form	MVC!Event	L
Parameter2Attribute	IFMLMM!IFMLParameter	MVC!Attribute	L
SimpleField2TextField	ExtMM!SimpleField	HTML!TextField	M
UMLClass2Controller	UMLMM!Class	MVC!Controller	L
UMLClass2Model	UMLMM!Class	MVC!Model	M
UMLProperty2Attribute	UMLMM!Property	MVC!Attribute	M
UMLProperty2Identifier	UMLMM!Property	MVC!Identifier	M
ViewContainer2View	IFMLMM!ViewContainer	MVC!View	M
ViewElement2Image	IFMLMM!ViewElement	HTML!Image	L
ViewContainer2InnerView	IFMLMM!ViewContainer	MVC!View	L

Luego de haber introducido las reglas utilizadas en la transformación M2M, se presentan ejemplos de los mapeos -y sus respectivas reglas- entre la vista IFML Model y el paquete MVC del metamodelo propuesto, en la Figura 4.1; como así también de la vista Specific ViewComponent -del metamodelo de IFML- y el paquete HTML del metamodelo resultante del trabajo, en la Figura 4.2.

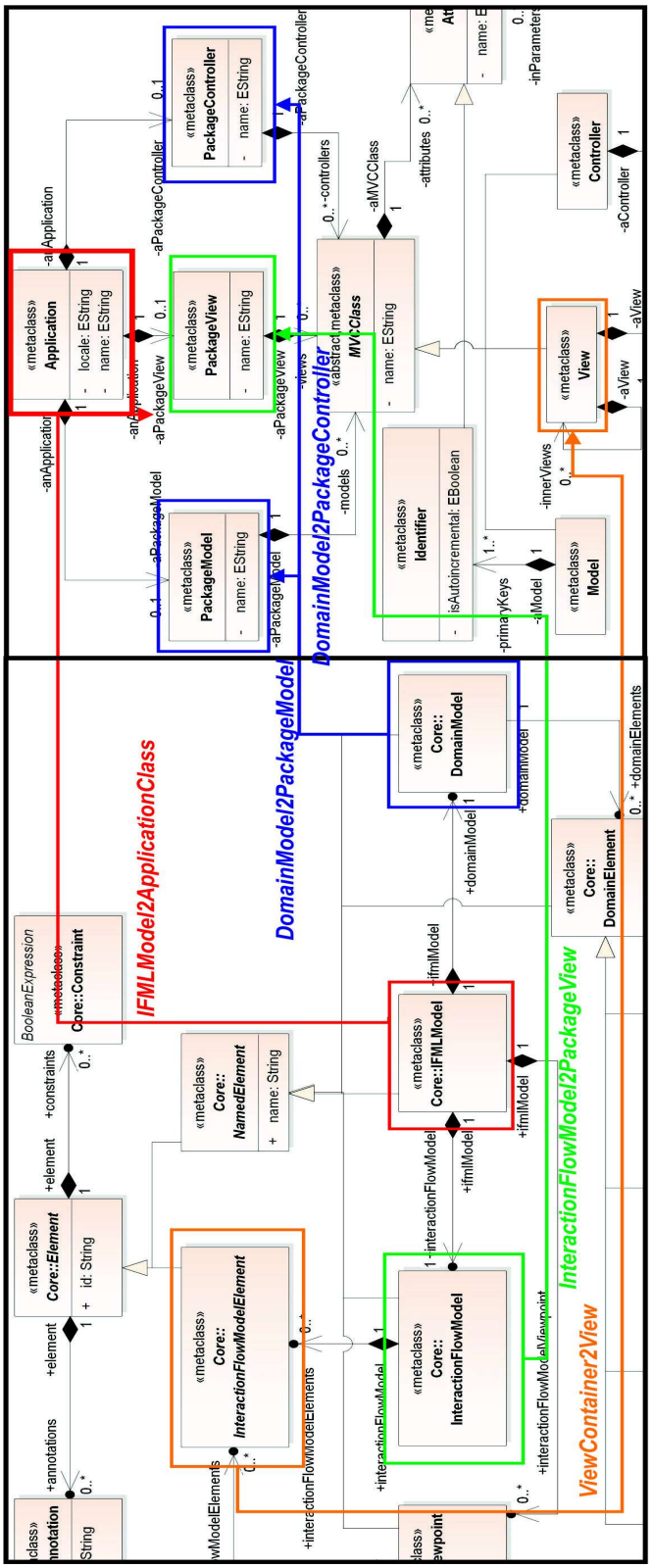

Figura 4.1. Mapeo de metaclases - Vista IFML Model y paquete MVC.

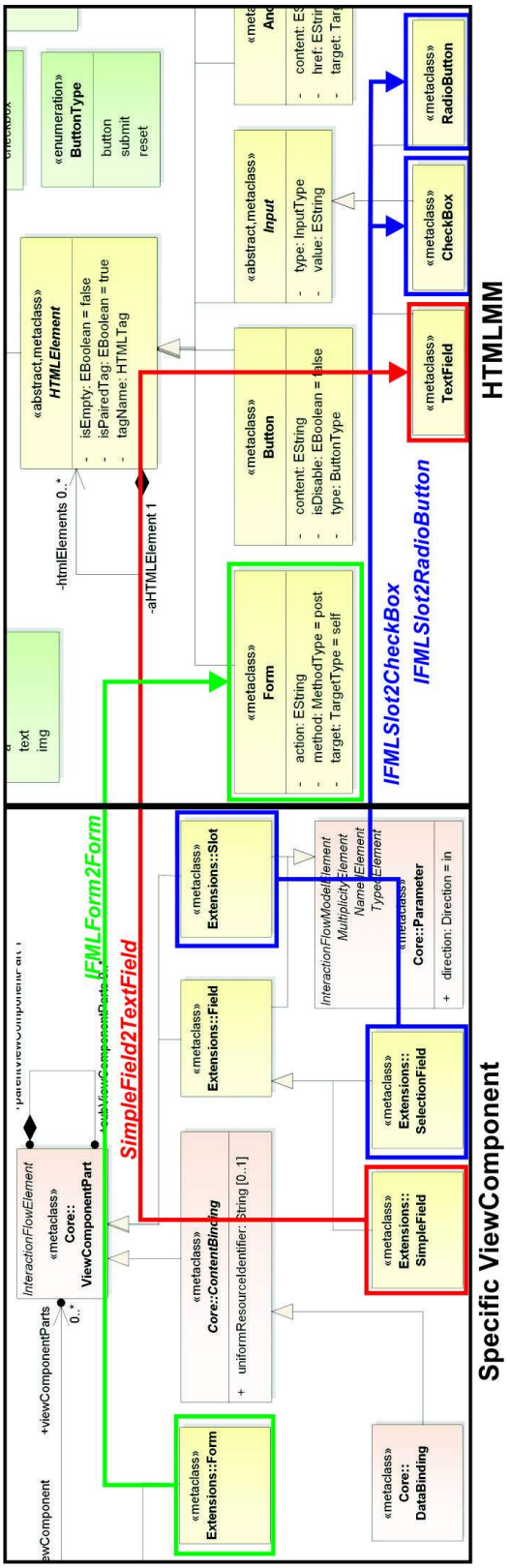

Figura 4.2. Mapeo de metaclases - Vista Specific ViewComponent y paquete HTML.

De estas 19 reglas, la ejecución de algunas depende también de algunas condiciones específicas, las cuales se detallan en la Tabla 4.8.

Tabla 4.8. Detalles de condiciones de ejecución - Reglas ATL.

Rule	Condición de ejecución
Landmarks2MenuItems	El ViewContainer a transformar en un menu item, está seteado como landmark. Es decir, que puede ser accedido desde cualquier parte de la aplicación.
UMLProperty2Attribute	La propiedad a transformar no está marcada como identificador.
UMLProperty2Identifier	La propiedad a transformar está seteada como identificador.
ViewContainer2InnerView	El padre contenedor inmediato de la ViewContainer a transformar es de tipo ViewContainer. Es decir, se trata de una subvista.
ViewContainer2View	El padre contenedor inmediato de la ViewContainer a transformar no es de tipo ViewContainer. Es decir, no se trata de una subvista.

4.2.2. Justificación De Cada Regla

ConditionalExpression2Text: desde que es razonable asumir que el texto presentado en la aplicación esté guardado en fuentes de datos externas, las expresiones condicionales -que son expresiones predefinidas de consulta asociadas con DataBindings- son un buen candidato para representar la consulta de dicho texto. Se utiliza <p> como nombre de tag HTML por defecto en la metaclase Text.

DomainModel2PackageController: desde que los IFML Actions representan porciones de la lógica de negocios pero no las agrupa en conceptos de algún nivel superior, es dificil poder establecer una relación directa entre el metamodelo de IFML y un paquete de controladores PHPFW. Esta regla aprovecha la cardinalidad uno a uno entre modelo de dominio IFML y paquete de controladores PHPFW y asume un controlador por cada clase de dominio UML.

DomainModel2PackageModel: desde que el DomainModel se define como contenedor de todos los elementos de contenido utilizados por ViewComponents. Puede verse una clara relación con el concepto de PackageModel.

IFMLAction2Method: los IFML Action representan porciones de la lógica de negocios de la aplicación. Esa lógica de negocios es implementada, en *frameworks* de desarrollo para lenguajes Orientados a Objetos, utilizando métodos.

IFMLForm2Form: el concepto de Form es el mismo en ambos metamodelos. A saber, la representación de un formulario de entrada donde el usuario puede enviar información entre páginas. Por defecto, se asume POST por método de envío y *self* como target.

IFMLModel2ApplicationClass: puede entenderse la transformación entre estas 2 metaclases desde el punto de vista de que un IFMLModel es el contenedor del resto de los elementos de modelo, y por ello, el de más alto nivel, al igual que una Application en el metamodelo al que conforman los modelos destino.

IFMLSlot2Checkbox: un IFML Slot representa los valores que puede asumir un determinado campo del formulario. Esta regla es invocada en los casos en el que dicho campo pueda ser multivaluado.

IFMLSlot2RadioButton: un IFML Slot representa los valores que puede asumir un determinado campo del formulario. Esta regla es invocada en los casos en el que dicho campo es univaluado.

InteractionFlowModel2PackageView: Esta regla se basa en la idea de que un InteractionFlowModel agrega todos los elementos que modelan la interacción con el usuario y siguiendo un patrón arquitectónico MVC el usuario interactúa con la vista.

Landmarks2MenuItems: esta regla transforma todos los ViewContainers que son landmark a items de un menú. Similar al funcionamiento de la herramienta WebRatio, que incluye la implementación paga pero más extendida de IFML.

OnSubmitEvent2Event: El concepto de evento es el mismo en el metamodelo origen y el destino, por lo cual la transformación resulta natural. El nombre corresponde a que en esta implementación, en vistas de cubrir el caso de estudio y sus criterios de éxito, solo se transforma los eventos OnSubmit, pero nada restringe que no puedan tratarse otros eventos.

Parameter2Attribute: una instancia de un IFML Parameter es un nombre tipado capaz de mantener valores, y por ello, puede entendérsela como atributo.

SimpleField2TextField: un SimpleField de IFML es un un campo que puede capturar entradas textuales del usuario, al igual que un textfield (<*input type="text" />*) de HTML.

UMLClass2Controller: Ver Justificación de la regla “DomainModel2PackageController”. Si el modelador tiene en mente el uso de un patrón arquitectónico MVC y desea que determinados IFML Actions modelados pertenezcan a un mismo controlador, debe terminar el nombre cada IFML Action con el nombre del controlador al que pertenecerán después de la transformación.

UMLClass2Model: desde que una clase puede utilizarse para especificar la estructura que caracteriza un determinado objeto, una clase se puede transformar en una metaclase Model que cumpla ese mismo rol en el metamodelo PHPFW.

UMLProperty2Identifier: algunas de las propiedades de los modelos pueden utilizarse para identificar unívocamente un objeto, esas propiedades de UML pueden transformarse en Identifiers.

UMLProperty2Attribute: a excepción de la regla UMLProperty2Identifier, esta regla se disparar para todas las UML Properties que no son IDs.

ViewContainer2InnerView: ViewContainer (View) refieren al mismo concepto en metamodelo origen (destino). Definimos como Inner View a una vista que está contenida dentro de otra.

ViewContainer2View: ViewContainer (View) refieren al mismo concepto en metamodelo origen (destino).

ViewElement2Image: desde que un ViewElement son elementos de la interfaz de usuario que muestran contenido, contiene la estructura necesaria como para ser transformada en un Image.

- **Consideraciones**
  - Paquete MVC
    - Method: dado que el enfoque de modelado con IFML de tiene como base el uso de un patrón arquitectónico MVC, no resulta trivial conocer que a que controlador debe pertenecer un método dado. Se optó por usar el nombre del método para poder transformar esta pertenencia, a saber: cada método debe terminar con el nombre del controlador del cual se desea que forme parte.
    - Views: con el fin de poder separar vistas contenedoras de vistas contenidas, se usó una función reflexiva de ATL que retorna la meta clase contenedora inmediata de una clase dada.
    - Event: con el fin de simplificar las transformaciones (sin dejar de considerar los casos de estudio referidos) se optó porque el prototipo solo maneja eventos del tipo OnSubmit..
  - Paquete HTML
    - Checkboxes y RadioButtons: ambos fueron generados a partir de la metaclase extMM!SeletionField. Para los primeros, se consideró que el atributo *isMultiSelection* de dicha metaclase tenga un valor *true.* Lo contrario fue considerado para el caso de los botones de radio.
    - Anchors: basados en una idea usada en la herramienta WebRatio, todos los ViewContainers cuyo atributo *isLandmark* tenga un valor *true*, forman parte de una barra de navegación presente en todas las vistas. Los Anchors se usaron para crear este *navbar*.
    - Image: se utiliza el id del ViewElement como atributo src de cada imagen.

- **Valores Predeterminados**

Para la realización de las transformaciones, se asumieron una serie de valores predeterminados para las clases resultantes de la misma. Estos valores se presentan en la Tabla 4.9.

Tabla 4.9. Valores predeterminados en la transformación M2M.

Clase	Atributo	Valor
Anchor, Form	target	self
Event	type	onSubmit
Form	method	post

4.3. TRANSFORMACIONES PSM A CÓDIGO

4.3.1. Estructura Del Proyecto M2T: Laravel

Siguiendo las recomendaciones de las buenas prácticas de Acceleo[13] los paquetes del proyecto se organizaron en:

13 https://www.obeo.fr/fr/acceleo-best-practices/406-acceleo-best-practices

- **Paquete Files**

Todos los módulos que generen archivos. Todos los archivos generados incluyen bloques protegidos para que el usuario pueda escribir código manualmente y no perderlo en las generaciones de código subsiguientes. A continuación, en la Tabla 4.10 se detallan los módulos y *templates* incluidos en este paquete. Un módulo contiene uno o más *templates* que generan texto a partir de un modelo y uno o más módulos de los cuáles depende para funcionar y que por ello se llaman dependencias. Salvo que se indique lo contrario, cada modulo tiene un solo *template* con su mismo nombre.

Tabla 4.10. Laravel M2T: Módulos, dependencias y templates del paquete files.

Módulo	Dependencia	Templates
generateWelcomeView	services::helperLaravelOutputPath	generateWelcomeView
setApplicationConf	services::helperApplication services::helperLaravelInputPaths	setApplicationConf
writeController	services::helperLaravelInputPaths services::helperStringUtil	writeControllerSkeleton registerController writeViewConstructors
writeModelRequest	services::helperFileReader	writeModelRequest
writeModelSkeleton	-	writeModelSkeleton
writeViews	services::helperStringUtil services::helperLaravelInputPaths	writeView registerPostForm
writeDefaultStylesheet	services::helperLaravelOutputPaths	writeDefaultStylesheet

Los valores de entrada, sus respectivos tipos y una descripción de cada uno de estos *templates* puede verse en la Tabla 4.11.

Tabla 4.11. Laravel M2T: Templates, Entradas y Descripciones del paquete files.

Template	Entrada	Descripción
generateWelcomeView	-	Genera una vista predeterminada de la aplicación. Actualmente es un portal con un enlace al repositorio GitHub con el código del proyecto. Se quiere destacar que no es la mejor práctica el uso de módulo como éste, que no reciban parámetros de entrada, para no perder precisión en la trazabilidad de cada región de código generada.
setApplicationConf	anApp : Application	Cambia el nombre y la configuración regional de la aplicación según lo indicado en los modelos y los beans de configuración, respectivamente. Para esto necesita el path del archivo de configuración

Template	Entrada	Descripción
writeControllerSkeleton	path : String controller : Controller appName : String packageModelName : String	Genera: a) El espacio de nombres de la clase PHP considerando el nombre de la aplicación y el del paquete de los modelos. b) Los espacios de nombres de Controllers, *Request* y Models a partir los nombres de la aplicación, del controlador y del paquete de modelos. c) Los métodos del controlador considerando el nombre de cada método, sus parámetros de entrada y salida y el *boilerplate* para los controladores generado por la consola interactiva Artisan integrada a Laravel. Para diferenciarlos de los modelos, los controladores generados terminan con el sufijo "Controller".
registerController	nameController : String namePackage : String pv : PackageView	Se optó por llamar registro de controladores al proceso de inclusión de los mismos en las rutas del *framework*. Este *template* registra los controladores que permiten la navegabilidad por la aplicación a partir de las vistas, los nombres de los controladores y de su paquete. Además se registra una ruta que considere todos los *requests* que producirían un error HTTP 404 (página no encontrada) y devuelva la vista generada por el *template* generateWelcomeView
writeViewConstructors	path : String controllerName : String pv : PackageView	Crea los métodos en los controladores necesarios para la instanciación de las vistas.
writeModelRequest	appName : String path : String pmName : String model : Model	Genera e importa los espacios de nombres necesarios a partir del nombre de la aplicación y del nombre del paquete de modelos. Además, genera reglas de validación por defecto para todos los atributos del modelo

Template	Entrada	Descripción
writeModelSkeleton	appName : String path : String model : Model	El esqueleto de un model se genera a partir del nombre de la aplicación y de su path. Los modelos indican que atributos se esperan recibir mediante *requests* y el nombre de la tabla en la base de datos.
writeView	path : String view : View pcName : String pvName : String	Las vistas generadas usan una declaración de tipo de documento HTML5. Estas vistas incluyen todos los viewComponents (Anchor, Text, Checboxes, Radiobuttons, Images y Forms. Para no definir de manera exhaustiva y con código estático los atributos de cada HTMLElements, podría utilizarselos mediante su representación ECore (EAttributes). El problema que surge es que los EAttributes no permiten por sí mismos la especificación de valores por defecto. Por lo cual, aun necesataríamos castear los HTMLElements a un elemento determinado y de esa manera no se aprovecharía la mayor ventaja de este enfoque: permitir la adición de atributos en el metamodelo a los HTMLElements (por ejemplo, conforme a alguna nueva especificación HTML) sin necesidad de modificar las reglas -al menos- M2T. Nótese que también este enfoque también aumentaría la abstracción del metamodelo de los *frameworks* PHP pero haría perder expresividad a las clases que lo conformen. Se quiere destacar que en esta versión de la transformación no contemplamos formularios que estén compuestos de otros forms, anchors o imágenes.
registerPostForm	actionName : String formHandler : String controllerName : String	Se encarga del registro de la ruta necesaria para generar la respuesta a los envíos de formularios entre páginas

Template	Entrada	Descripción
writeDefaultStylesheet		Este módulo crea clases CSS (*Cascading Style Sheets*) para dar estilo a las páginas web generadas en la transformación.

- **Paquete Main**

Por cambios en la plataforma de Eclipse en la versión Luna (la mínima necesaria para poder manejar correctamente todas las dependencias) la estrategia de usar Acceleo Plugin como lanzador de los módulos Acceleo ejecutables no funciona. La alternativa es usar Java como lanzador, como si se tratara de una aplicación stand alone. Para estos casos EMF no descubrirá ni registrará los metamodelos disponibles y por ello debe hacer esto manualmente.

Este paquete contendrá todos los modulos que tengan un módulo main y por ello tengan asociado un lanzador Java con el registro manual de los metamodelos incluido en el método *registerPackages*. A continuación, en la Tabla 4.12 se muestran las dependencias y los *templates* de cada módulo.

Tabla 4.12. Laravel M2T: Módulos, dependencias y templates del paquete main.

Módulo	Dependencias	Template
application	files::setApplicationConf	application
controller	services::helperFileReader services::helperLaravelPaths services::helperLaravelInputPaths services::helperLaravelOutputPaths services::helperApplication files::writeController	makeControllers
main	main::controller main::model main::view main::application	generateElement
model	services::helperFileReader services::helperApplication services::helperLaravelPaths services::helperLaravelOutputPaths files::writeModelSkeleton files::writeModelRequest	makeModels
view	services::helperFileReader services::helperLaravelPaths files::writeViews files::generateWelcomeView	makeViews makeInnerViews

El detalle de cada uno de estos *templates* puede verse en Tabla 4.13.

Tabla 4.13. Laravel M2T: Templates, entradas y descripciones del paquete main.

Template	Descripción	Entradas
makeControllers	Genera la carpeta que contendrá todos los controladores generados y podrá dentro todos los controladores después de generarlos con *writePHPTag*, *writeControllerSkeleton*, *writeViewConstructors* y *registerControllers*	anApp : Application
generateElement	Se encarga de coordinar la invocación a los *templates* que generan la aplicación, sus modelos, vistas y controladores. Su fin es que toda la salida pueda ser generada con la ejecución de solamente este *template*.	
makeModels	Crea las carpetas donde se generaran los modelos y los ModelRequests y lo puebla con el resultado de la invocación a los módulos *writeModelSkeleton* y *writeModelRequest*	
makeViews	Crea la carpeta que se poblará con las vistas generadas como producto de la invocación al *template writeView* y *makeInnerViews.*	
makeInnerViews	Se invoca recursivamente mientras haya vistas que contentan otras vistas.	path : String view : View controllerName : String packageViewName : String

- **Paquete Services**

Todos los módulos que invocan métodos de Java. El ámbito de todas las clases es público. Todos los módulos de este paquete tienen como prefijo la palabra “helper” y como sufijo el nombre de la clase Java que permite invocar. Además de los helpers, este paquete contiene las clases Java tabuladas en la Tabla 4.14.

Tabla 4.14. Laravel M2T: Clases, descripción y métodos del paquete services.

Clase	Extiende	Atributos	Métodos	Descripción
FileReader	-	-	getFile makeFolder	Encapsula la lógica relacionada con la lectura de archivos del disco.
LaravelInputPaths	LaravelPaths	private String configAppPath: especifica el path del cuál se lee la configuración de Laravel.	Accesores y mutadores de los atributos.	Encapsula rutas de Laravel que brindan información útil para la transformación M2T. Por ello, son consideradas rutas de entrada.

Clase	Extiende	Atributos	Métodos	Descripción
		private String RoutesPath: especifica el path del cuál se lee la configuración de rutas de Laravel		
LaravelOutputPaths	LaravelPaths	private String laravelVersion: especifica la raíz donde se guardará el proyecto generado usando la versión de Laravel. private String modelsHome: indica en dónde se guardan los modelos de Laravel. private String viewsHome: indica dónde se guardan las vistas de Laravel private String controllersHome: indica dónde se guardan los controladores de Laravel. private String stylesheetsPaths: indica dónde se guardan las hojas de estilo de Laravel	Accesores y mutadores de los atributos.	Encapsula rutas de Laravel en dónde deberían guardarse determinados archivos generados. Por ello, son consideradas rutas de salida.
LaravelPaths	-	private String baseDir private String laravelVersion	-	Dado que cada path pertenece a un aspecto particular de la transformación podría encapsularse en la clase relacionada pero dado de que puede ser necesario cambiar los paths debido a un cambio de versión de Laravel, decidimos agruparlas todas juntas, siguiendo un

Clase	Extiende	Atributos	Métodos	Descripción
				principio de responsabilidad simple.
StringUtil	-	private char[] chars	-	-

El detalle de cada clase de este paquete services puede verse en la Tabla 4.15.

Tabla 4.15. Laravel M2T: Clases, métodos y descripción del paquete services.

Clase	Métodos	Parámetros	Retorno	Descripción
FileReader	Public getFile	path : String	file : String	Retorna el contenido de un archivo a partir de su path
	Public getFile	path : String keyword : String	File : String	Retorna las lineas de un archivo que contengan una determinada palabra clave.
	Public makeFolder	path : String	void	Genera una carpeta en el path dado.
StringUtil	Public getSuffix	str : String	suffix : String	Obtiene el sufijo de un string escrito siguiendo una notación CamelCase.
	Public getPrefix		prefix : String	Obtiene el prefijo de un string escrito siguiendo una notación CamelCase.
	Public separatePrefixSuffix		phrase : String	Separa un string en notación CamelCase en prefijos y sufijos

Dado que no solo existen atributos del metamodelo PHPFW -tales como rutas y parámetros regionales de la aplicación o nombre y versión del *framework* destino- que no pueden obtenerse como fruto de las transformaciones, sino que además el metamodelo podría extenderse para dar soporte a algunas características que, aunque no quedaron comprendidas dentro del alcance de esta versión del metamodelo, implicarían el modelado de los aspectos y características presentados en la Tabla 4.16.

Tabla 4.16. Aspectos y características que no se pueden obtener de modelos IFML.

Aspecto	Característica
Aplicación	Tipo (Web, Consola, Api)
Controladores	¿Es *hookeable*?
Librerías utilizadas	Su nombre, path y versión.
Script de entrada	Entorno (desarrollo, producción, *testing*)
Vistas	Versión de HTML.

Se hizo necesario definir un mecanismo que permita este soporte y evolución. Para ello se analizaron las opciones de usar: clases Java, Anotaciones IFML o un diagrama de clases UML adicional. Los pros y contras de las opciones mencionadas se presentan en la Tabla 4.17.

Tabla 4.17. Pros y contras de la adopción de distintas alternativas al problema de información no contenida en los modelos IFML

Solución	Pros	Contras
Clases Java	- Podría aplicarse Ingeniería Inversa para obtener un diagrama de clases UML a partir de ellos en caso de que sea necesario. - Mayor facilidad respecto a la extensibilidad: agregar un nuevo atributo a una clase Java es más fácil que adaptar una regla de transformación M2M que lo soporte. - Existen antecedentes en el uso de clases Java con este fin, como el caso de AndroMDA[14]. - Puede usarse cualquier lanzador Java asociado a un módulo Acceleo con el fin de parametrizar estas clases de forma dinámica.	- Desde que una clase Java no es naturalmente un modelo, no es la solución más cercana a MDD.
IFML Annotations	- Las anotaciones pueden hacerse sobre el modelo, sin necesidad de escribir o modificar código, por lo cuál es una opción más idiomática respecto a MDD. - El metamodelo estaría más en línea con el principio abierto/cerrado, es decir, de ser abierto para extensión , cerrado para modificación. Se usarían mecanimos previstos.	- Se debería definir una sintaxis y usar expresiones regulares para especificar como interpretar las anotaciones ya que no están escritas en ningún lenguaje estructurado particular. - El modelo PIM ya no sería independiente de la plataforma.

14 https://www.andromda.org/

Solución	Pros	Contras
Diagrama de clases UML adicional	- Utilizar un lenguaje estándar y conocido. - UML permite la definición de restricciones OCL.	- El desarrollador debería hacer un modelo más. - Habría un modelo de entrada más para las transformaciones M2M, con sus consecuentes reglas de transformación. - Se debería definir un *profile* para este modelo.

Después de este análisis se decidió por el uso de clases de Java, por lo cuaĺ se añadió a la estructura de paquetes previstas el paquete:

- **Paquete Main.Beans**

  La clase que se presenta tabulada en la Tabla 4.18 es pública.

Tabla 4.18. Laravel M2T: Clases, atributos y métodos del paquete main.beans.

Clase	Atributos	Métodos
Application	private String locale private FileReader fr private LaravelInputPath lip	getDefaultAppName getAppConfigLines getAppConfig getRoutes getLocale setLocale setAppName setPHPEnv

Todos los métodos de la tabla mostrada a continuación, en la Tabla 4.19, son públicos y pertenecen a la clase Application.

Tabla 4.19. Laravel M2T: Detalle de los métodos de la clase Application.

Método	Parámetros	Tipo Retorno	Descripción
getDefaultAppName	framework : String	defaultAppName : String	Este método retorna el nombre por default que tienen las aplicaciones en los *frameworks* considerados.
getAppConfigLines	keyword : String	lines : String	Retorna todas las líneas de un archivo de configuración que contengan una palabra clave dada
getAppConfig	-	config : String	Retorna el archivo de configuración de Laravel.

Método	Parámetros	Tipo Retorno	Descripción
getRoutes		routes : String	Retorna el archivo de rutas Web de Laravel.
getLocale		locale : String	Permite obtener los parámetros regionales de la aplicación en tiempo de ejecución.
setLocale	locale : String	void	Permite obtener los parámetros regionales de la aplicación en tiempo de ejecución.
setAppName	pathFramework : String appName : String	void	Cambia el nombre el nombre de la aplicación mediante el uso, de manera transparente para el usuario, de la consola interactiva Artisan

4.3.2. Estructura Del Proyecto M2T: Yii2

- **Paquete Common**

Este paquete, contiene los métodos y *templates* de utilidades mostrados en la Tabla 4.20.

Tabla 4.20. Yii2 M2T: Módulos y templates del paquete common.

Módulo	Templates
stringUtils	camelCase2CaterpillarCase writeIncludes
writeIncludes	writeHelpers

La entrada y descripción de cada uno de estos *templates* se presenta en la Tabla 4.21.

Tabla 4.21. Yii2 M2T: templates del paquete common.

Template	Entradas	Descripción
camelCase2CaterpillarCase	camelCased : String	Convierte la notación del string dado como parámetro, de notación Camel Case a notación Caterpillar Case.

Template	Entradas	Descripción
writeHelpers	view : View	Escribe los includes necesarios para que los helpers URL y HTML puedan utilizarse en aquellos casos donde existan viewComponents de tipo Anchor y/o Form.

- **Paquete Files**

Este paquete contiene todos los módulos que generan archivos. Todos los archivos generados incluyen bloques protegidos para que el usuario pueda escribir código manualmente y no perderlo en las generaciones de código subsiguientes. A continuación, en la Tabla 4.22, se detallan los módulos y *templates* incluidos en este paquete. Salvo que se indique lo contrario, cada modulo tiene un solo *template* con su mismo nombre.

Tabla 4.22. Yii2 M2T: módulos del paquete files.

Módulo	Dependencias	Templates
generateWelcomeView	services::helperYii2OutputPath	generateWelcomeView
writeControllerSkeleton	services::helperYii2InputPaths services::helperApplication services::helperFileReader	writeControllerSkeleton registerController
writeDefaultStylesheet	services::helperYii2OutputPaths	writeDefaultStylesheet
writeView	services::helperStringUtil services::helperYii2InputPaths services::helperYii2OutputPaths common::stringUtils common::writeIncludes	writeView registerPostForm

Se presentan más detalles acerca de los *templates* mencionados en la tabla anterior en la Tabla 4.23.

Tabla 4.23. Yii2 M2T: templates del paquete files

Template	Entrada	Descripción
generateWelcomeView	-	Genera una vista predeterminada de la aplicación. Actualmente es un portal con un enlace al repositorio GitHub con el código del proyecto. Se quiere destacar que no es la mejor práctica el uso de módulo como éste, que no reciban parámetros de entrada, para no perder precisión en la trazabilidad de cada región de código generada.
writeControllerSkeleton	path : String controller : Controller appName : String packageModelName : String	Este módulo genera: a) El espacio de nombres de la clase PHP considerando el nombre del paquete de los modelos. b) Los métodos del controlador considerando el nombre de cada método y sus

Template	Entrada	Descripción
		parámetros de entrada y salida. Para diferenciarlos de los modelos, los controladores generados terminan con el sufijo "Controller".
registerController	pc : PackageController	Se optó por llamar registro de controladores al proceso de inclusión de los mismos en las rutas del *framework*. Este *template* registras los controladores que permiten la navegabilidad por la aplicación a partir de las vistas, los nombres de los controladores y de su paquete. Además se registra una ruta que considere todos los *requests* que producirían un error HTTP 404 (página no encontrada) y devuelva la vista generada por el *template* generateWelcomeView
writeDefaultStylesheet	-	El *template* del mismo nombre del módulo genera código CSS básico.
writeModelSkeleton	appName : String path : String model : Model	El esqueleto de los modelos se generan a partir del nombre de la aplicación y de su path.
writeView	path : String view : View pcName : String pvName : String	Las vistas generadas usan una declaración de tipo de documento HTML5. Estas vistas incluyen todos los viewComponents (Anchor, Text, Checboxes, Radiobuttons, Images y Forms. Para no definir de manera exhaustiva y con código estático los atributos de cada HTMLElements, podría utilizárselos mediante su representación ECore (EAttributes). El problema que surge es que los EAttributes no permiten por sí mismos la especificación de valores por defecto. Por lo cual, aun necesataríamos castear los HTMLElements a un elemento determinado y de esa manera no se aprovecharía la mayor ventaja de este enfoque: permitir la adición de atributos en el metamodelo a los HTMLElements (por ejemplo, conforme a alguna nueva especificación HTML) sin necesidad de modificar las reglas -al menos- M2T. Nótese que también este enfoque también aumentaría la abstracción del metamodelo

Template	Entrada	Descripción
		de los *frameworks* PHP pero haría perder expresividad a las clases que lo conformen. Se quiere destacar que en esta versión de la transformación no se contempla el uso de formularios que estén compuestos de otros forms, anchors o imágenes.
registerPostForm	actionName : String formHandler : String controllerName : String	Este *template* se encarga del registro de la ruta necesaria para generar la respuesta a los envíos de formularios entre páginas

- **Paquete Main**

Por cambios en la plataforma de Eclipse en la versión Luna (la mínima necesaria para poder manejar correctamente todas las dependencias) la estrategia de usar Acceleo Plugin como lanzador de los módulos Acceleo ejecutables no funciona. La alternativa es usar Java como lanzador, como si se tratara de una aplicación stand alone. Para estos casos EMF no descubrirá ni registrará los metamodelos disponibles y por ello debe hacer esto manualmente.

Este paquete contendrá todos los modulos que tengan un módulo main y por ello tengan asociado un lanzador Java con el registro manual de los metamodelos incluido en el método *registerPackages*. La Tabla 4.24 tabula los módulos contenidos en este paquete, mientras que la Tabla 4.25 presenta los *templates* del paquete en cuestión.

Tabla 4.24. Yii2 M2T: módulos del paquete main

Módulo	Dependencias	Templates
application	services::helperApplication services::helperYii2InputPaths files::writeControllerSkeleton services::helperFileReader	setAppConf setNameApp setLocaleApp
controller	services::helperFileReader services::helperApplication files::writeControllerSkeleton files::writeView services::helperYii2Paths services::helperYii2OutputPaths	makeControllers
main	main::controller main::model main::view main::application	generateElement
model	services::helperFileReader services::helperApplication files::writeModelSkeleton services::helperYii2Paths services::helperYii2OutputPaths	makeModels

Módulo	Dependencias	Templates
view	services::helperFileReader files::writeView files::generateWelcomeView services::helperYii2OutputPaths files::writeDefaultStylesheet services::helperYii2Paths	makeViews makeInnerViews

Tabla 4.25. Templates del paquete main.

Template	Descripción	Entradas
setAppConf	Coordina el llamado de los *templates* encargados de generar la configuración de la aplicación.	anApp : Application
setNameApp	Cambia el nombre de la aplicación según lo contenido en el nombre definido en los modelos o con el nombre predeterminado de las aplicaciones de Yii en caso de que lo anterior no fuese posible.	
setLocaleApp	Configura el locale de la aplicación según lo indicado en los modelos.	
makeControllers	Genera todos los controladores modelados	
generateElement	Se encarga de coordinar la invocación a los *templates* que generan la aplicación, sus modelos, vistas y controladores. Su fin es que toda la salida pueda ser generada con la ejecución de solamente este *template*.	
makeModels	Genera todos los modelos que fueron modelados	
makeViews	Genera todas las vistas de cada PackageView y coordina la generación de una vista de bienvenida para la aplcación y una hoja de estilo predeterminada.	
makeInnerViews	*Template* recursivo que crea y anida todas las vistas internas de una vista	path : String view : View controllerName : String pvName : String

- **Paquete Main.Beans**

La única clase contenida en este paquete, llamada Application contiene los atributos y métodos tabulados en Tabla 4.26.

Tabla 4.26. Yii2 M2T: Clase, atributos y métodos del paquete main.beans

Clase	Ámbito	Atributos	Métodos
Application	public	public String locale private FileReader fr private Yii2InputPaths lip private String defaultAppName	getDefaultAppName getAppConfigLines isSet getAppConfig getLocale setLocale

A continuación se muestra una tabla con el detalle de los métodos del bean Application usado en la transformación M2T al *framework* Yii2. Todos los métodos mencionados en Tabla 4.27 son públicos.

Tabla 4.27. Yii2 M2T: métodos de la clase Application

Método	Parámetros	Retorno	Descripción
getDefaultAppName	String framework	String appName	Los nombres por defecto de la aplicación cambian según el *framework* con el que se esté trabajando. Este método tiene mapeado cada *framework* con el nombre predeterminado de la aplicación y a partir de allí, lo retorna.
getAppConfigLines	String keyword	String lines	Este método retorna todas las líneas del archivo de configuración de Yii que tengan una palabra clave dada.
isSet	boolean	String attribute	Retorna *true* si el atributo pasado como parámetro está seteado como atributo en el archivo de configuración de la aplicación.

- **Paquete Services**

Este paquete contiene las clases que son consultadas por helpers de Acceleo. Todos las clases de este paquete, tabuladas en Tabla 4.28, son públicas.

Tabla 4.28. Yii2 M2T: métodos del paquete services.

Clase	Métodos	Atributos	Extiende
FileReader	getFile makeFolder removeReturnStmt	-	-
StringUtil	getSuffix getPrefix separatePrefixSuffix camelCase2CaterpillarCase	char[] chars;	-

Clase	Métodos	Atributos	Extiende
Yii2InputPaths	Todos los métodos son accesores o mutadores de la clase.	String configAppPath	Yii2Paths
Yii2OutputPaths		private String modelsHome private String viewsHome private String controllersHome private String welcomeView private String stylesheetsPath	
Yii2Paths		private String baseDir private String yii2Version	-

Todos los métodos tabulados a continuación, en la Tabla 4.29, son públicos.

Tabla 4.29. Yii2 M2T: Detalle sobre los métodos de las clases del paquete services

Método	Clase	Parámetros	Retorno	Descripción
getFile	FileReader	String path	file : String	Obtiene un archivo desde un path.
geFile		String path String keyword		Obtiene las líneas de un archivo que contengan una palabra clave dada.
removeReturnStmt		String varReturn String path	void	Método saca la línea que contiene el retorno de una variable dada. Necesario para poder agregar líneas a un archivo de configuración en modo “append”.
makeFolder		String path	void	Este método crea una carpeta en un path dado.
getSuffix	StringUtil	String str	suffix : String	Obtiene el sufjio de un string escrito siguiendo una notación CamelCase.
getPrefix	-	-	prefix: String	Obtiene el prefijo de un string escrito siguiendo una notación CamelCase.
separatePrefixSuffix	-	-	str : String	Separa un string en notación CamelCase en prefijos y sufijos
camelCase2CaterpillarCase	-	-	-	Convierte notación del string dado como entrada, convirtiendo de notación CamelCase a notación CaterpillarCase.

5. VERIFICACIÓN: CASO DE ESTUDIO

Este capítulo presenta el Caso de Estudio que se utilizará para verificar que las abstracciones propuestas en el metamodelo están cubiertas, son válidas y útiles para generar aplicaciones Web utilizando *frameworks* PHP.

Para ello, se expondrán los modelos usados para la ingeniería de requerimientos -como casos de uso y modelos de dominio- y otros más cercanos al diseño pero que sirven para especificar el caso de estudio, como el modelo de flujos de interacción de IFML.

5.1. INTRODUCCIÓN

La verificación del metamodelo se llevó a cabo utilizando un caso de estudio que consistió en la implementación de las reglas de transformación que permitan pasar de modelos de dominio UML y Flujos de Interacción IFML a código para los *frameworks* Laravel y Yii2 que asegure la correcta generación de: modelos que se puedan crear, leer, actualizar y borrar; vistas que incluyan componentes básicos: botones, anclas, textos, imágenes, formularios y entradas de texto mediante *radiobuttons, textfields y checkboxes y* controladores que aseguren la navegabilidad entre las distintas partes de la aplicación Web.

Todos ellos aprovechando las funciones específicas que podría brindar cada *framework* destino, respecto a, por ejemplo, seguridad de la información.

Se ha elegido como caso de estudio del presente trabajo, la realización de una aplicación que permita mantener una base de datos de películas en línea (también llamada *Movies Database,* del inglés Base de Datos de Películas), estos se debe a que, casos similares, se han presentado para ejemplificar la capacidad de IFML para representar un sistema funcional [46] o como iniciación a la herramienta IFML Editor[15].

Por otra parte, trabajos como [61] y otras herramientas MDWE -como en el caso del proceso de desarrollo UWE[16]- se han valido de ejemplos similares para demostrar de manera general la aplicabilidad de estas técnicas a un sistema real y usable por usuarios finales.

5.2. CASOS DE USO

5.2.1. Diagrama De Casos De Uso

En la Figura 5.1 se presenta un diagrama de casos de uso UML de la aplicación, especificando las interacciones posibles entre el actor -en este caso, un usuario cualquiera- y el sistema.

5.2.2. Especificación Textual De Casos De Uso

Tomando como punto inicial el diagrama presentado en la sección 5.2.1, a continuación se presenta la especificación textual de los casos de uso expuestos allí y basados en el esquema presentado en [62].

15 http://ifml.github.io/
16 http://uwe.pst.ifi.lmu.de/exampleIMDB.html

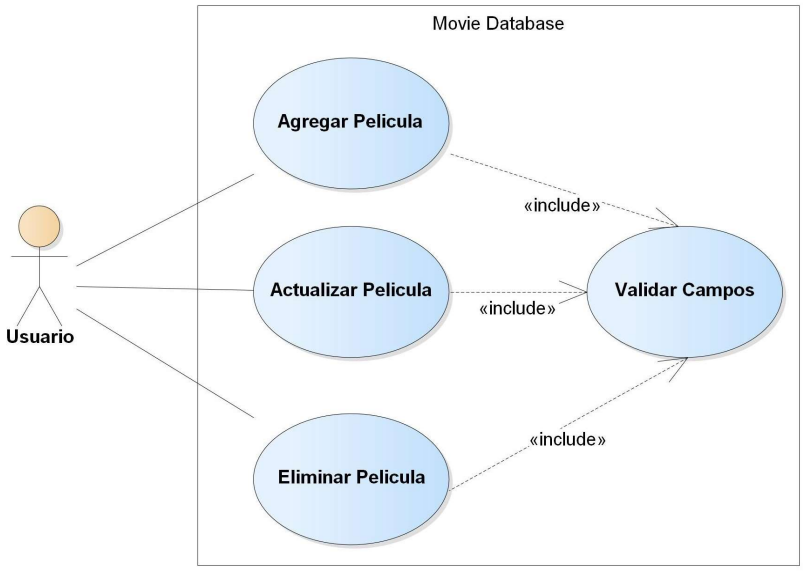

Figura 5.1. Diagrama Caso de Uso - Caso de Estudio

En la Tabla 5.1 se describe los pasos que un usuario debería seguir a la hora de agregar una nueva película, mientras que en la Tabla 5.2 y Tabla 5.3 se presentan los pasos necesarios para actualizar una pelicula y eliminarla, respectivamente. Finalmente, en la Tabla 5.4, se presentan los pasos realizados en la validación de las entradas.

Tabla 5.1. Especificación Caso de Uso - Agregar Película

Nombre	**Agregar Película**	Versión	1.0
Actores	Usuario		
Precondiciones	-		
Postcondiciones	Se registra la nueva película en la base de datos.		
Casos de usos incluidos	Validar campos.		
Escenario Principal	1. El usuario usa un navegador para acceder al sistema.		
	2. El usuario inicia el proceso de alta para una nueva película.		
	3. El usuario ingresa datos válidos (Ver Caso de Uso: “Validar Campos”) para el título de la película, el año de su estreno, su clasificación por edades y si la película forma o no parte de una saga.		
	4. El sistema informa al usuario que la película se ha registrado exitosamente.		

Tabla 5.2. Especificación Caso de Uso - Actualizar Película

Nombre	**Actualizar Película**	Versión	1.0
Actores	Usuario		
Precondiciones	-		
Postcondiciones	La película tiene ahora las entradas ingresadas por el usuario a través del formulario.		
Casos de usos incluídos	Validar campos.		
Escenario Principal	1. El usuario usa un navegador para acceder al sistema.		
	2. El usuario inicia el proceso de modificación para una película.		
	3. El usuario ingresa datos válidos (Ver Caso de Uso: “Validar Campos”) para la película que desea actualizar, que pueden ser: el título, el año de su estreno, su clasificación por edades y/o si la película forma o no parte de una saga.		
	4. El sistema informa al usuario que la película se ha actualizado exitosamente.		

Tabla 5.3. Especificación Caso de Uso - Eliminar Película

Nombre	**Eliminar Película**	Versión	1.0
Actores	Usuario		
Precondiciones	-		
Postcondiciones	No existe más la película en la base de datos.		
Casos de usos incluídos	Validar campos.		
Escenario Principal	1. El usuario usa un navegador para acceder al sistema.		
	2. El usuario inicia el proceso de eliminación de una película.		
	3. El usuario ingresa un título de película válido (Ver Caso de Uso: “Validar Campos”).		
	4. El sistema informa al usuario que la película se ha eliminado de manera exitosa.		

Tabla 5.4. Especificación Caso de Uso - Validar Campos

Nombre	**Validar Campos**	Versión	1.0
Actores	-		
Precondiciones	Se ha enviado un formulario de alta, baja o modificación.		
Postcondiciones	Se confirma que los campos cargados son correctos		
Casos de usos incluídos	-		

Nombre	**Validar Campos**	Versión	1.0
Escenario Principal	1. Los campos ingresados por el usuario son validados.		
	2. El sistema permite continuar con su operación normal.		
Flujos Alternativos	3. El usuario ingresa: 3.1. Un título de película vacío o con caracteres que no son letras mayúsculas, minúsculas, guiones, números o paréntesis.		
	3.2. Un año de estreno no vacío con más o menos de 4 dígitos.		
	3.3. Un valor de audiencia no vacío con valores distintos a G, PG, PG-13 y R.		
	3.4. Un valor de saga no vacío que es distinto a true y false.		
	En cualquier caso, se le notifica al usuario de su error y, permaneciendo en la misma página, se le permite corregirlo.		

5.3. MODELO DE DOMINIO

Habiendo definido las tareas que debe realizar el sistema y como debe responder a las mismas, se presenta el modelo de dominio, usado para identificar los conceptos e ideas relevantes del dominio.

En la Figura 5.2 se presenta la clase que fue identificada, sobre la cual han de interactuar todas las operaciones del sistema. Para la realización del modelo de dominio se usó el *plugin* para Eclipse llamado Papyrus[17].

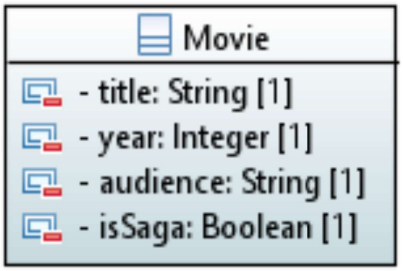

Figura 5.2. Modelo de Dominio

5.4. PROTOTIPADO DE INTERFACES GRÁFICAS DE USUARIO

La presentación de la interfaz gráfica del usuario, su contenido, los mecanismos de interacción que implementa y su aspecto general tiene mucha importancia en la satisfacción del usuario y su posterior aceptabilidad. En [58] se menciona que, crear estos prototipos es una buena idea hacerla durante la etapa de especificación de los requerimientos.

17 https://eclipse.org/papyrus/

Mientras más temprano -dentro del proceso de desarrollo- se pueda validar con el usuario los prototipos de las interfaces, más altas son las probabilidades que el usuario, al final, obtenga lo que espera.

Desde que este trabajo apunta a la generación semiautomática de aplicaciones Web, el enfoque fue puesto en el aspecto funcional, mucho más fácil de obtener de los modelos que los aspectos presentacionales. De cualquier manera, es recomendable que tanto durante la construcción de los prototipos como durante la realización en sí del *front-end* se utilicen las mismas herramientas, ya que suelen ser accesibles (desde un punto de vista económico) y totalmente funcionales.

Considerando ya en mayor detalle la aplicación a obtener, en la Figura 5.3 se muestra la vista principal de la base de datos de películas en línea. En esta vista el usuario puede observar una barra de navegación que le permita llegar a las funcionalidades principales de la aplicación y leer una breve descripción de la aplicación con una imagen.

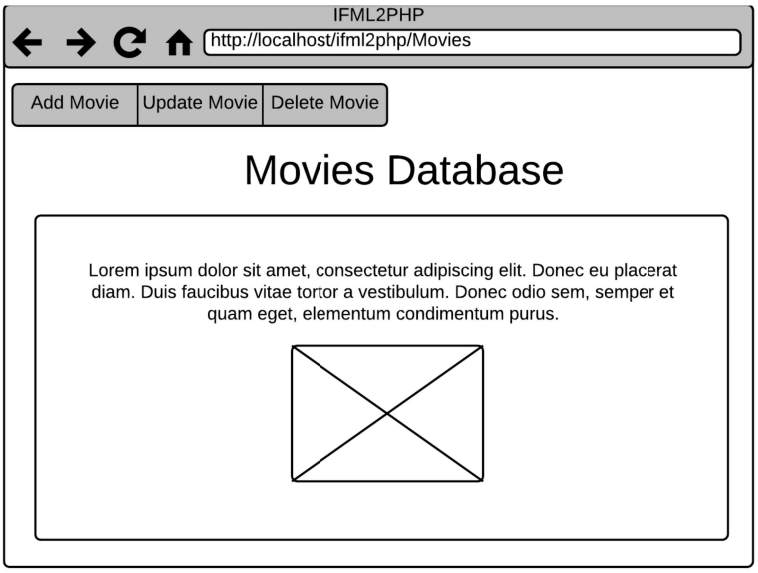

Figura 5.3. Mockup - Menú Principal

Cuando el usuario hace click en el botón “Add Movie” de la barra de navegación, se muestra una figura como la mostrada en Figura 5.4. En esta vista el usuario puede ingresar el título y el año de estreno de la película, la edad pretendida de la audiencia y si la película es o no una saga, para finalmente ingresar una nueva película a la base de datos.

En cambio, si el usuario hace click en el botón “Update Movie” de la barra de navegación del menú principal, se muestra una figura como la mostrada en Figura 5.5. En esta vista el usuario puede ingresar los campos de la película que desea actualizar, y que pueden ser: el título y el año de estreno de la película, la edad pretendida de la audiencia y si la película es o no una saga.

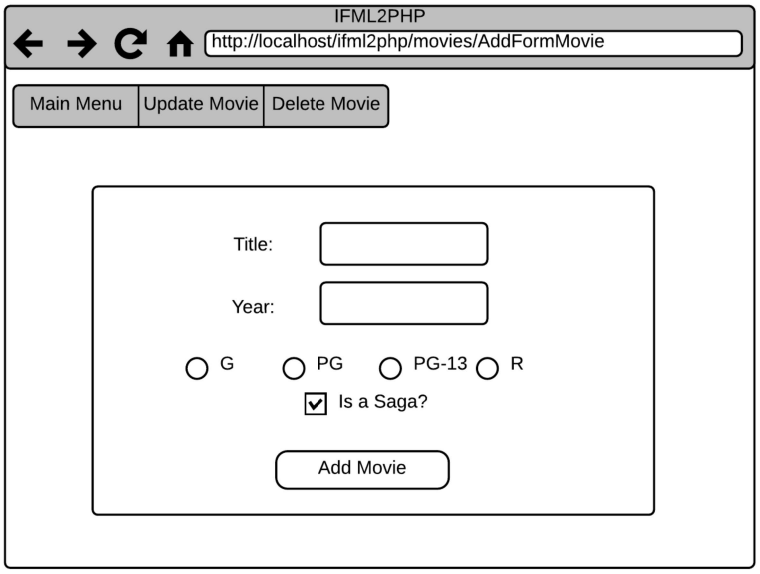

Figura 5.4. Mockup - Add Movie

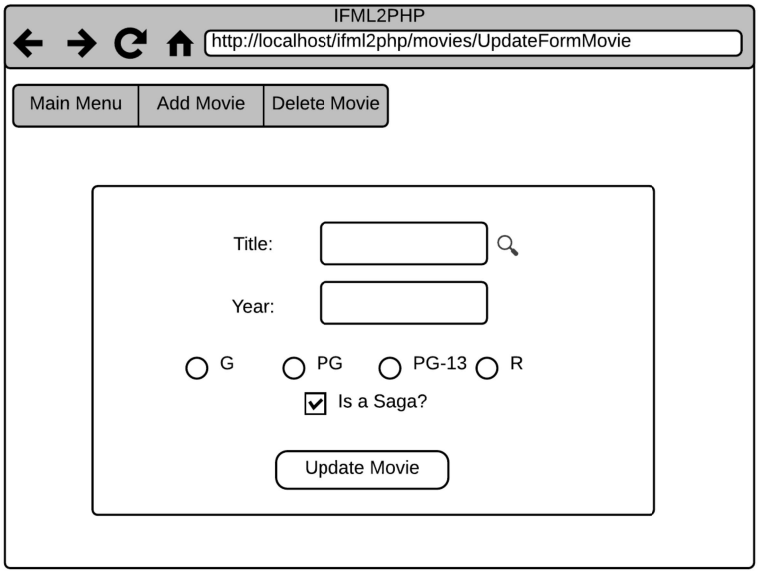

Figura 5.5. Mockup - Update Movie

Por otra parte, si el usuario hace click en el botón “Delete Movie” de la barra de navegación del menú principal, se muestra una figura como la mostrada en Figura 5.6. Es esta vista el usuario puede ingresar el título de la película a eliminar.

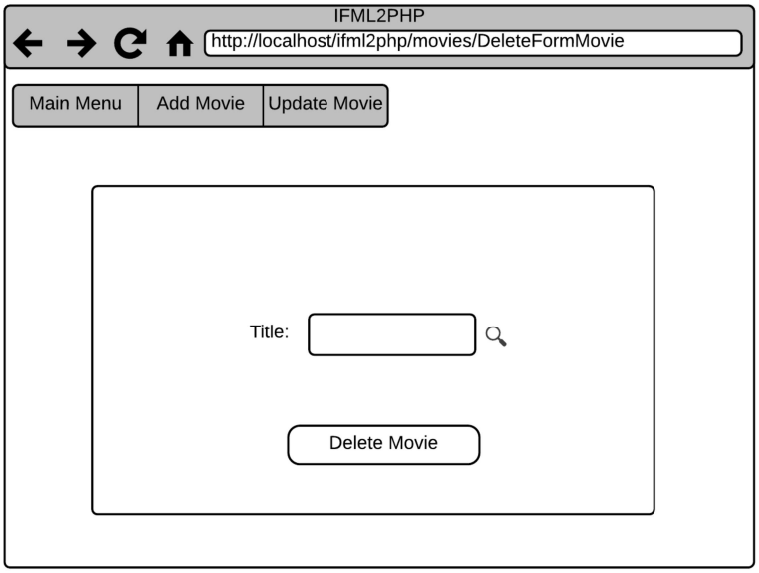

Figura 5.6. Mockup - Delete Movie

5.5. MODELO DE INTERACCIÓN DE FLUJOS DEL USUARIO

Para conformar el caso de estudio especificado, el modelado IFML del mismo se ha realizado con IFML Editor. IFML Editor es una herramienta de código abierto basada en Eclipse y Sirius API. Es importante aclarar que, dicha herramienta funciona bajo Eclipse Luna 4.4[18] y el paquete de Eclipse Modeling Tools[19].

Para la construcción de un modelo IFML con la herramienta utilizada, se debe contar con un Diagrama de Clases UML que describa el modelo de dominio (*Domain Model*) IFML y sobre el cual se va a basar el modelo de flujo de interacción (también conocido como *Interaction Flow Model*).

En la Figura 5.7, se puede observar una representación en vista de árbol (*Tree View*, por su nombre en inglés) del diagrama de clases a partir del modelo de dominio presentado en la sección 5.4.

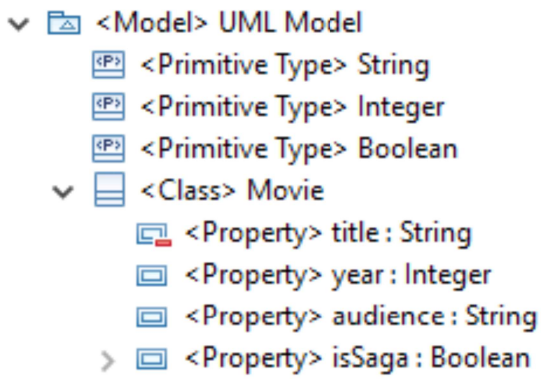

Figura 5.7. Tree View - Diagrama de Clases

18 https://www.eclipse.org/luna/
19 https://www.eclipse.org/downloads/packages/eclipse-modeling-tools/lunasr2

A partir de todas las consideraciones previas, en la Figura 5.8 se presenta el diagrama IFML construido para el caso de estudio.

En este punto, ya se cuenta con una representación completa en IFML del caso de estudio, teniendo en cuenta los caso de uso y *mockups* presentados como así también, el modelo de dominio.

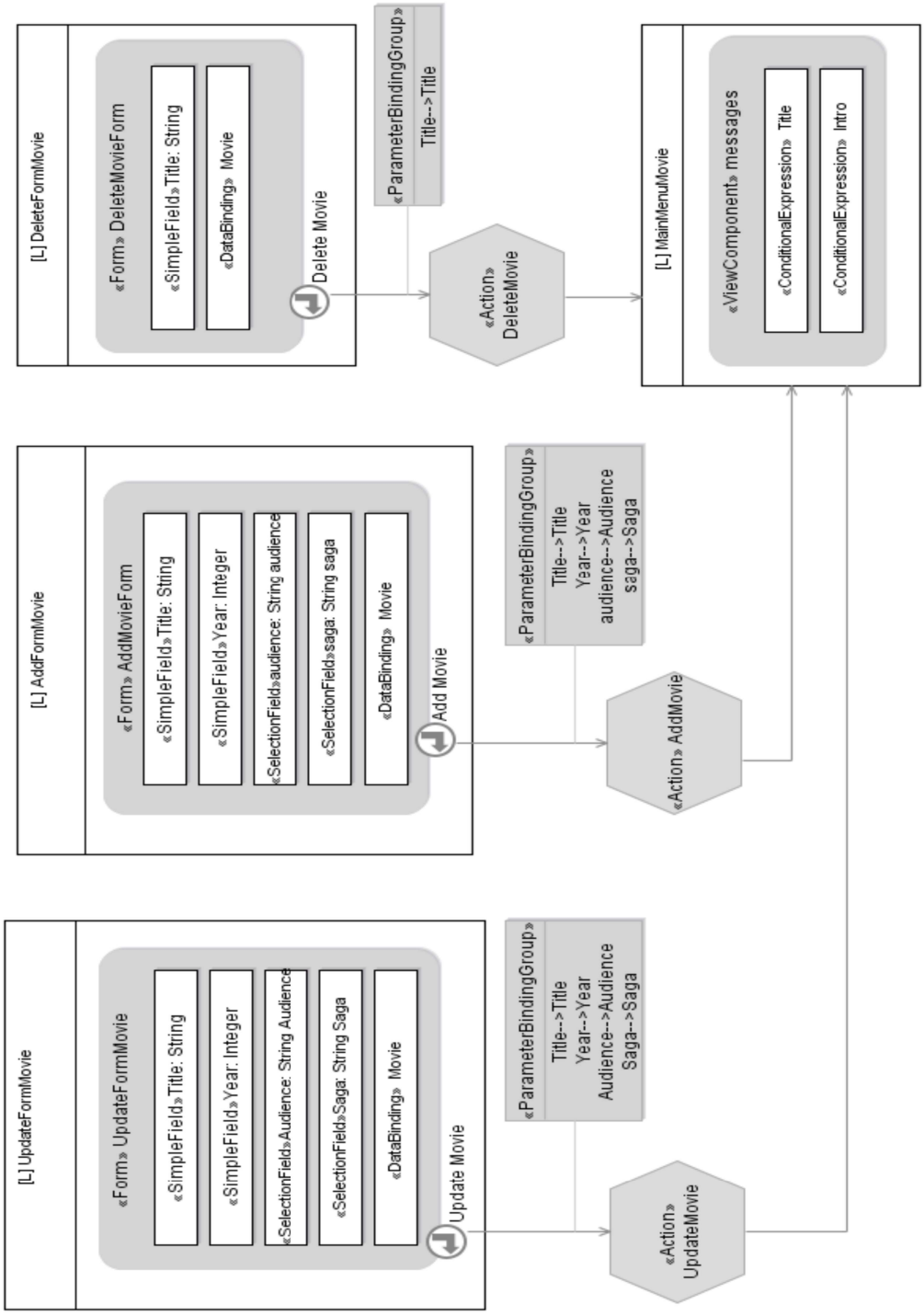

Figura 5.8. Diagrama IFML.

Finalmente, en la Figura 5.9 se puede apreciar -de manera general- mediante una vista de árbol como dentro del *IFML Model* se contiene el *Domain Model* (obtenido a partir del diagrama de clases presentado en la Figura 5.7) y el *Interaction Flow Model* (modelado en la Figura 5.8).

Mientras que en la Figura 5.10 se aprecia en detalle -a modo de ejemplificar- los elementos que contiene la ventana para agregar una nueva película, del tipo *IFML Window* llamado *AddFormMovie*.

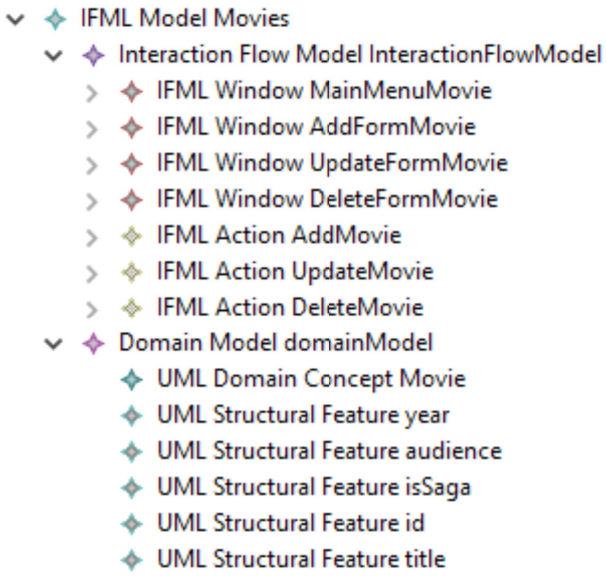

Figura 5.9. Tree View - IFML Model

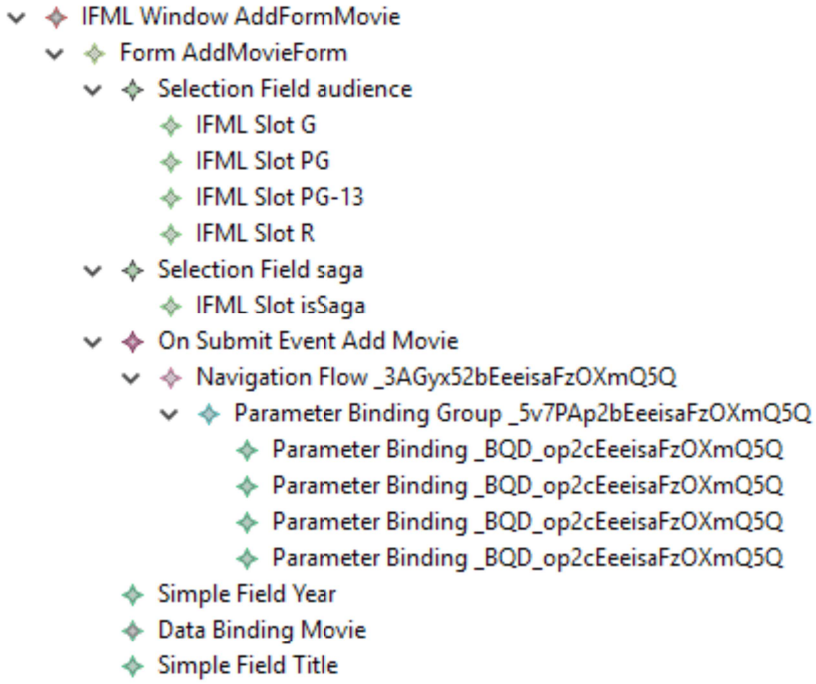

Figura 5.10. Tree View - AddFormMovie (IFML Window)

5.6. VERIFICACIÓN DE LA APLICACIÓN GENERADA

Se consideró necesaria la realización de pruebas unitarias para el presente trabajo, de manera que se pruebe la correcta transformación M2T y que aún cuando conviva código manual con el generado automáticamente, éste no quede invalidado por aquel.

Con el fin de verificar que la aplicación generada estuviera compuesta de los elementos HTML modelados, tales como: formularios, entradas de texto, botones de radio, etc. se utilizó la herramienta de automatización de *tests* Selenium WebDriver[20] en su versión 3.8.1. Hay que tener en cuenta que se debe contar con los *drivers* apropiados para cada browser en donde se quieran correr los tests. La versión de estos drivers a optar depende del sistema operativo, la arquitectura del procesador y, por supuesto, la versión del browser. Los *tests*, escritos en Java, fueron construidos a partir del conjunto de casos de prueba presentados en el Anexo G, los mismos son una simplificación del estándar de la IEEE (*Institute of Electrical and Electronics Engineers*) [63].

Los casos de prueba realizados para la aplicación resultante, se encuentran categorizados en pruebas sobre la página web y sobre los formularios. Los mismos fueron realizados tanto para Laravel como en Yii2.

- *Test* para la Página Web
  - Contengan título.
  - Contengan botón para enviar formularios (*submit*) aquellas que así lo requieran.
  - Contengan anclas *(anchors)*.
  - Contengan imágenes aquellas que así deberían.
- *Test* para los formularios
  - Contengan entradas de texto *(inputs)*.
  - Contengan *labels*.
  - Contengan botones de radio *(radiobuttons)* aquellas que así lo requieran.

20 http://www.seleniumhq.org/projects/webdriver/

6. RESULTADOS

En la siguiente sección se exponen como resultados una serie de medidas realizadas sobre el metamodelo, un análisis de las transformaciones M2M y un análisis de las reglas utilizadas para la obtención del mismo. Finalmente, en cuanto a las transformaciones M2T, se presentan las estructuras de archivos obtenidas y las interfaces de usuarios funcionando de la aplicación generada.

6.1. MEDIDAS DEL METAMODELO

Los metamodelos son un punto clave en MDD, tal es así que cualquier artefacto en el ecosistema de modelado debe ser definido acorde a un metamodelo, lo cual representa una descripción ontológica del dominio de una determinada aplicación -cf.: [6]

Pese a la importancia que tienen los metamodelos, pocas investigaciones se han llevado a cabo analizándolos empíricamente. Es crucial entender las características comunes de los metamodelos, cómo evolucionan a través del tiempo y/o qué impacto tienen los cambios realizados en el ecosistema de modelado. Algunas propuestas para realizar esta tarea se presentan en [64]–[66], donde se definen atributos de calidad como interpretabilidad, reusabilidad y extensibilidad. Mientras que otros estudios como [67], [68] proponen la adopción de métricas usadas en el desarrollo de software orientado a objetos.

Las revisiones bibliográficas realizadas no permitieron encontrar un consenso claro respecto a que métricas usar en vistas a lograr análisis de los metamodelos. Se cree importante presentar una serie de medidas orientadas al tamaño realizadas sobre el metamodelo terminado. Dichas medidas se han obtenido mediante una adaptación del código fuente de la herramienta MDEForge[21]. Los resultados obtenidos se presentan en la Tabla 6.1.

Tabla 6.1. Medidas del metamodelo propuesto.

Medida	Valor
Paquetes	2
Metaclases	23
Enumeraciones	6
Metaclases abstractas	4
Metaclases concretas	19
Metaclases sin atributos propios ni heredados	0
Metaclases sin atributos propios pero si heredados	3
Metaclases abstractas sin atributos propios pero si heredados	0
Metaclases concretas sin atributos propios pero si heredados	3
Total de características sin considerar el árbol de herencia (referencias y atributos)	46
Total de características considerando el árbol de herencia (referencias y atributos)	115

21 http://www.mdeforge.org/

Medida	Valor
Total de referencias	15
Total de Atributos	31
Total de Atributos (considerando herencias)	78
Atributos de tipo primitivo	24
Nivel Máximo de Generalización	3
Promedio de referencias en metaclase (*fan-out*)	0,65
Promedio de atributos en metaclase	1,35
Promedio de metaclases por paquete	11,5
Metaclases hijas	14
Número máximo de clases hermanas	6
Metaclases aisladas	0

Por otra parte, se realizó una serie de medidas pero individualizando los resultados para cada clase con que cuenta el metamodelo. Para la obtención de ellas se ha utilizado la herramienta para Eclipse llamada EMF-Refactor[22]. De un conjunto de 23 medidas que ofrece la herramienta, se han tomado aquellas que se consideraron relevantes en el contexto de este proyecto y, a su vez, desglosen los valores obtenidos con anterioridad. Dichas medidas se presentan en la Tabla 6.2.

Tabla 6.2. Medidas para el conjunto de clases del metamodelo.

Acrónimo	Definición	Acrónimo en la Herramienta
LRL	Longitud del recorrido más largo hacia las hojas del árbol de jerarquías de relaciones de composición.	HAGGEC
TRC	Total de referencias de la clase.	NEREC
NRC	Cantidad de referencias de la clase hacia otras clases.	NEROEC
NAR	Cantidad de autorreferencias de la clase.	NERSEC
CAR	Cantidad de atributos y referencias de la clase.	NFEEC
CCC	Cantidad de clases que pueden componer la clase dada.	NPECEC
CCP	Cantidad de clases padres de la clase.	NSUPEC
TAC	Total de ancestros de la clase.	NSUPEC2

En la Tabla 6.3 se presentan, ordenados alfabéticamente, los valores obtenidos indicando su clase (*EClass*) y a que paquete pertenece (*Epackage*).

22 https://www.eclipse.org/emf-refactor/

Tabla 6.3. Medidas del metamodelo discriminado por clases.

EClass	EPackage	Medidas							
		LRL	TRC	NRC	NAR	CAR	CCC	CCP	TAC
Anchor	HTML	0	0	0	0	3	2	1	2
Application	MVC	4	3	3	0	4	3	0	0
Attribute	MVC	0	0	0	0	1	0	0	0
Button	HTML	0	0	0	0	3	2	1	2
Checkbox	HTML	0	0	0	0	0	2	1	3
Controller	MVC	2	1	1	0	0	2	1	1
Event	MVC	0	0	0	0	2	0	0	0
Form	HTML	0	0	0	0	3	2	1	2
HTMLElement	HTML	0	1	0	1	3	2	1	1
Identifier	MVC	0	0	0	0	1	0	1	1
Image	HTML	0	0	0	0	1	2	1	2
Input	HTML	0	0	0	0	2	2	1	2
MVCClass	MVC	1	1	1	0	1	1	0	0
Method	MVC	1	2	2	0	1	2	0	0
Model	MVC	1	1	1	0	0	2	1	1
PackageController	MVC	3	1	1	0	1	1	0	0
PackageModel	MVC	2	1	1	0	1	1	0	0
PackageView	MVC	3	1	1	0	1	1	0	0
RadioButton	HTML	0	0	0	0	0	2	1	3
Text	HTML	0	0	0	0	2	2	1	2
TextField	HTML	0	0	0	0	0	2	1	3
View	MVC	2	2	1	1	0	3	1	1
ViewComponent	MVC	1	1	1	0	1	1	0	0
Suma		**20**	**15**	**13**	**2**	**31**	**37**	**14**	**26**
Media		**0,87**	**0,65**	**0,57**	**0,09**	**1,35**	**1,61**	**0,61**	**1,13**

Una verificación directa de los valores contenidos en la Tabla 6.1 y Tabla 6.3 puede realizarse contrastando, por ejemplo, las medidas de **TRC (**Total de referencias de la clase**) y CAR (**Cantidad de atributos y referencias de la clase).

6.2. CARACTERIZACIÓN DEL METAMODELO PROPUESTO

Con el fin de entender mejor las principales características del metamodelo presentado se decidió ponerlo en el contexto de medidas realizadas sobre otros metamodelos. Para ello, se usaron como parámetro los valores consignados en [65]. En

dicho artículo, se lleva a cabo un análisis cuantitativo usando distintas medidas sobre un corpus de 537 metamodelos Ecore públicamente disponibles. Como algunas de estas medidas también han sido calculadas para este metamodelo, se presentan una serie de comparativas realizadas.

6.2.1. Comparativa Del Metamodelo

A continuación se listan un conjunto de características comparadas entre el corpus de metamodelos del artículo referenciado y el metamodelo propuesto en el presente trabajo, además dichas comparativas se grafican en la Figura 6.1.

- En el artículo se destaca que la cantidad media de metaclases por metamodelo es de 39,3, con una mediana de 13, máximo de 912 y mínimo de 1. El metamodelo del presente trabajo cuenta con un total de 23 metaclases, siendo inferior a la media.
- La cantidad promedio de características (conocidas como features) en el corpus es de 69,2, con un valor máximo de 2410 y un mínimo de 0. El metamodelo propuesto cuenta con un total de 46 características. La proporción en que la cantidad de features del metamodelo es menor al promedio (0,59) es muy similar a la proporción entre el tamaño en metaclases de éste y, de nuevo, el promedio del corpus (0,62).
- Por otro lado, mayores diferencias se encontraron al considerar que los metamodelos contenidos en el corpus tienen más referencias (43 en promedio) que atributos (26,2 en promedio). Mientras que, de las 46 características del metamodelo, 15 de ellas son referencias y las 31 restantes atributos.

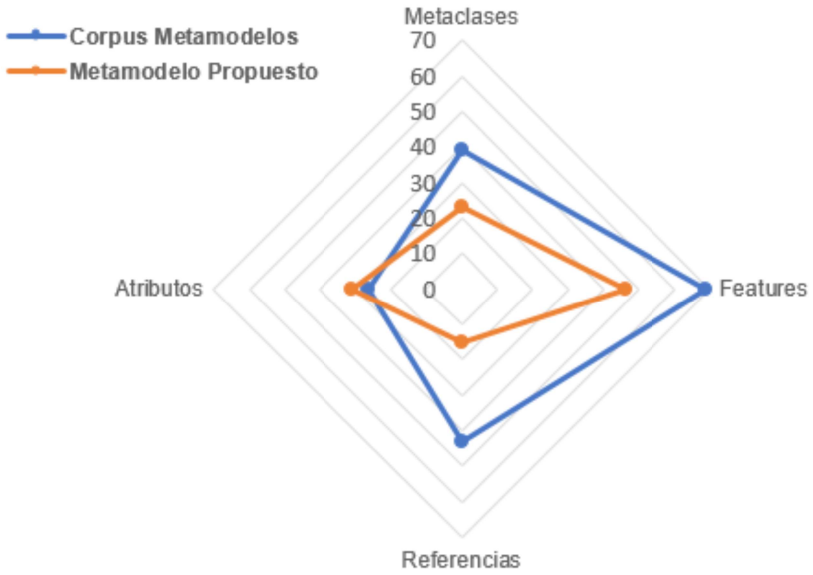

Figura 6.1. Gráfico comparativo de características del metamodelo.

6.2.2. Comparativa De Las Metaclases Del Metamodelo

A continuación se listan un conjunto de características comparadas entre las metaclases del corpus de metamodelos del artículo referenciado y las metaclases del metamodelo propuesto en el presente trabajo, además dichas comparativas se grafican en la Figura 6.2.

- El metamodelo cuenta con 4 metaclases abstractas. Esto está en línea al corpus contra el cuál se equiparó este metamodelo dónde el 56% de los metamodelos contienen entre 1 y 20 metaclases de esta característica.
- El metamodelo cuenta con 3 metaclases sin atributos de manera inmediata, es decir, con metaclases que no tienen atributos por sí mismas pero que heredan atributos de su super clase. El corpus con el que se compara cuenta con un 42% de metamodelos que poseen, al menos una, metaclase de similares características.
- El metamodelo propuesto cuenta con un promedio de 2 features por metaclase siendo 0,65 referencias y 1,35 atributos. Al observar los valores que presenta el corpus de metamodelos estudiado, la cantidad promedio de features por metaclase es de 2,1 lo cual muestra una similitud con el metamodelo de este trabajo. Pero, al ver con más detalle los valores se visualiza la diferencia ya presentada con en el punto anterior: la cantidad media de referencias por metaclase en el corpus es de 1,15 y de atributos es 0,95, mientras que en el metamodelo presentado se ve que de las 2 features por metaclase 0,65 son referencias y el 1,35 restante, atributos.

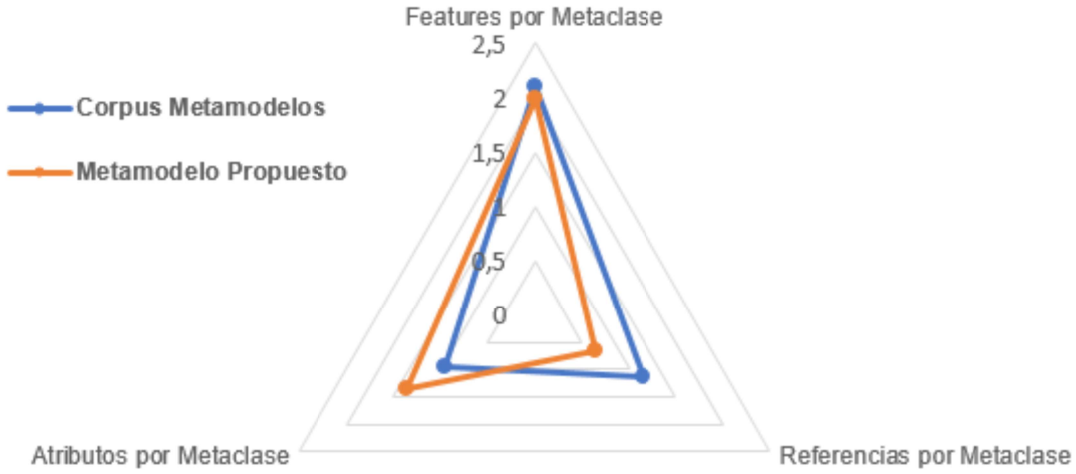

Figura 6.2. Gráfico comparativo de características de las metaclases del metamodelo.

6.3. MEDIDAS DE LAS REGLAS ATL PRODUCIDAS

Las reglas de transformación entre modelos son el núcleo de MDE y requieren ser tratados de una manera similar a los artefactos del desarrollo de software tradicional. Esto es así porque, por ejemplo, las reglas deben ser usadas por distintos desarrolladores, tienen que ser mantenidas de acuerdo a requerimientos cambiantes y, preferiblemente, deberían ser reutilizadas.

Pese a lo dicho y a que en los últimos años ha aumentado la disponibilidad de lenguajes y herramientas para la producción de las reglas, hay muy pocas técnicas disponible para analizar estas transformaciones.

Con el fin de estimar la calidad de las reglas ATL producidas existen distintas propuestas basadas en la idea de medir cuantitativamente determinadas características de las reglas -cf: [69]–[72].

En la realización de este TFC se utilizó la herramienta EMFTVM (*EMF Transformation Virtual Machine*)[23] para transformar el código a un modelo que pudiera servir como entrada a la herramienta MDEForge[24], la cuál ya había sido utilizada con anterioridad a la hora de medir las características del metamodelo. En la Figura 6.3 se puede observar la representación, en vista de árbol, de una parte del modelo obtenido para el código ATL.

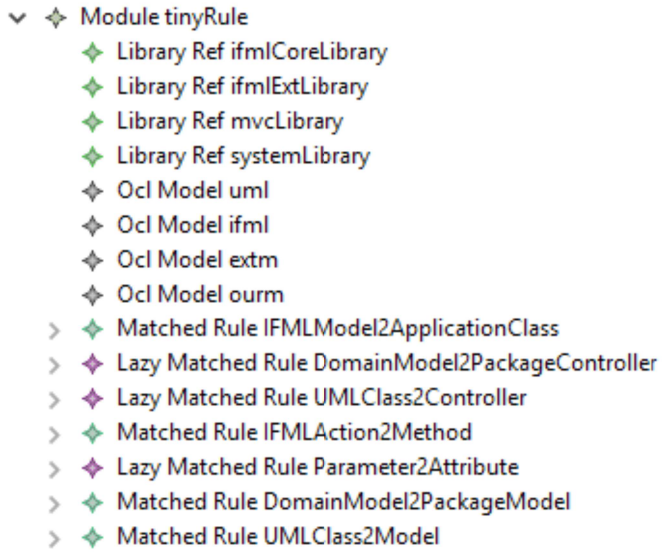

Figura 6.3. Representación parcial del modelo del código ATL en vista de árbol.

Los resultados obtenidos de las medidas se presentan en la Tabla 6.4 y en la Tabla 6.5, donde se ven las medidas globales del código ATL producido y las medidas detalladas sobre el conjunto de helpers y reglas, respectivamente.

Tabla 6.4. Medidas globales del código ATL producido.

Medida	Valor
Bindings	60
Patterns de entrada	22
Patterns de salida	20
Modelos de Entrada	3
Modelos de Salida	1
Reglas de Transformación	20
Lazy Matched Rules	9
Reglas con Condiciones en Input Pattern	5
Rule Inheritance Trees	20
Operation Helpers con contexto	17
Operation Helpers con parámetros	1
Operation Helpers sin parámetros	16

23 https://wiki.eclipse.org/ATL/EMFTVM
24 http://www.mdeforge.org/

Tabla 6.5. Medidas detalladas del código ATL producido.

Medida	Mínimo	Máximo	Mediana	Promedio	Desviación Estándar
Número de elementos del Input Pattern	1	2	1	1,1	0,31
Número de Matched Rules por Input Pattern	1	3	1	1,54	0,66
Número de Bindings por regla	1	6	3	3	1,41
Número de helpers por librería	1	9	3,5	4,25	3,4
Número de Helpers por Context	1	5	1	1,7	1,25
Número de Helpers por nombre del Helper (Sobrecargas)	1	2	1	1,13	0,35
Complejidad ciclomática de los helpers	1	2	1	1,06	0,24
Número de operaciones sobre colleciones por helper.	1	6	2	2,4	1,58
Número de parámetros por helper	0	1	0	0,06	0,24
Número de llamadas a Lazy Matched Rules por Lazy Matched Rule (Fan-In)	1	2	1	1,11	0,33
Número de llamadas a helpers por helper (Fan-In)	1	4	1	1,47	0,83
Número de llamadas desde helpers a helpers por helper (Fan-Out)	0	5	0	0,59	1,33
Número de llamadas a helpers por helper (Fan-Out)	0	5	0	0,59	1,33
Número de llamadas desde reglas a Helpers por regla(Fan-Out)	0	3	0	0,6	0,94
Número de llamadas desde reglas a reglas por regla (Fan-Out)	0	1	0	0,25	0,44
Número de llamadas desde reglas a Lazy Matched Rules por regla (Fan-Out)	0	1	0	0,25	0,44
Incremento de complejidad por regla	0,5	1	1	0,95	0,15

Siguiendo una idea presentada en [67], [70]. Se realizó un análisis de la relación entre las medidas obtenidas y los atributos de calidad de entendibilidad, modifcabilidad, consistencia, complejidad y reusabilidad. Ya que son atributos relevantes en el contexto de la solución que se buscó plantear. Salvo la complejidad, todos las demás son características que se interesa aumentar.

Considerando que:

- Los patterns de entrada y salida y los bindings son lo mismo desde un punto de vista de los atributos de calidad considerados. Todos reducen la entendibilidad, la modifcabilidad y la consistencia y aumentan la complejidad.
- El número de reglas disminuye la entedibilidad y la modificabilidad y aumentan la complejidad.
- El número de llamadas a reglas lazy, ya sea desde otras reglas o helpers, disminuyen la entendibilidad y aumentan la complejidad.
- Al aumentar el número de casi cualquiera de las medidas que fueron presentadas,, se afecta negativamente a los atributos de calidad mencionados. Las únicas excepciones a esto son: los árboles de herencia de reglas (Rule Inheritance Trees) que facilitan la modificabilidad y el número de helpers por nombre de helper (Sobrecargas) que mejoran la entendibilidad.

Se llegó a la conclusión de que la entendibilidad de la solución M2M (y todo lo que ella conlleva) podría mejorarse ajustando la organización arquitectónica de los helpers.

Una representación gráfica de lo mencionado aquí puede verse en la Tabla 6.6

Tabla 6.6. Medidas ATL y su relación con algunos atributos de calidad

Medida	Complejo	Entendible	Modificable	Consistente	Reusable
Número de Helpers por Context	▲				
Modelos de entrada					▼
Modelos de Salida		▼			
Bindings	▲			▼	
Patterns de entrada					
Patterns de salida					
Número de reglas					
Lazy Matched Rules			▼		
Reglas con Condiciones en Input Pattern					
Número de operaciones sobre colecciones por helper.					
Número de llamadas a helpers por helper (Fan-In)					
Complejidad ciclomática de los helpers					
Número de llamadas a Lazy Matched Rules por Lazy Matched Rule (Fan-In)					

Medida	Complejo	Entendible	Modificable	Consistente	Reusable
Número de llamadas desde reglas a Helpers por regla(Fan-Out)					
Número de llamadas desde reglas a Lazy Matched Rules por regla (Fan-Out)					
Rule Inheritance Trees			▲		
Número de Helpers por nombre del Helper (Sobrecargas)		▲			

6.4. TRANSFORMACIÓN M2M

La transformación M2M consistió en la transformación de un modelo de Flujos de Interacción IFML y un modelo de dominio UML a un modelo que conforme al metamodelo propuesto en este TFC. Los modelos de entrada tomaron como guía de su realización al Caso de Estudio. En la Figura 6.4 puede verse una porción del resultado de la ejecución de la transformación M2M utilizando ATL.

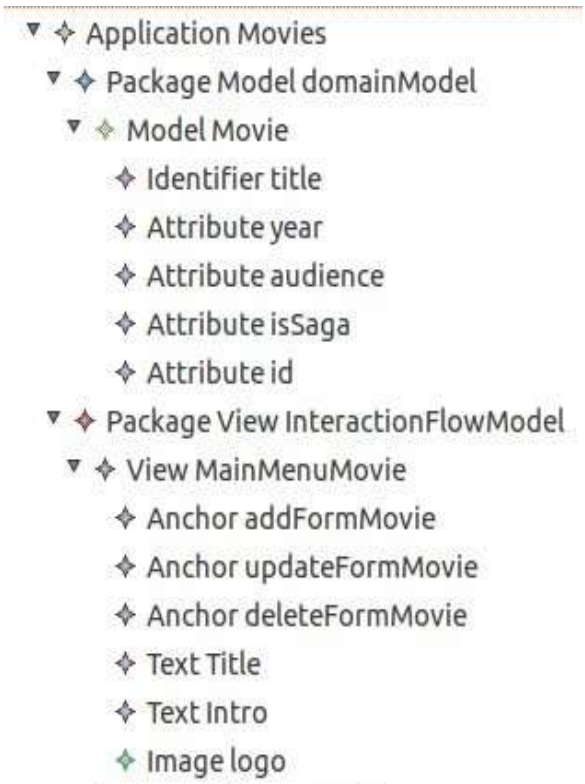

Figura 6.4. Porción del modelo transformado en vista de árbol

A su vez, cada metaclase generada cuenta con las propiedades que han sido consignadas en la sección 3. Un ejemplo, de la metaclase generada Text puede verse en la Figura 6.5.

Property	Value
Content	Lorem ipsum
Is Empty	false
Is Paired Tag	true
Language	es
Name	Intro
Tag Name	p

Figura 6.5.Propiedades de la metaclase Text

A modo de resumen del proceso de transformación M2M, se presenta en la Figura 6.6 un diagrama que ejemplifica el proceso completo. A partir del modelo IFML -en este caso un *ViewContainer* llamado *AddFormMovie*- pasa por una serie de reglas ATL, entre ellas *ViewContainer2View* y finalmente, como resultado de esta transformación, se obtiene el modelo intermedio obtenido a partir del metamodelo propuesto.

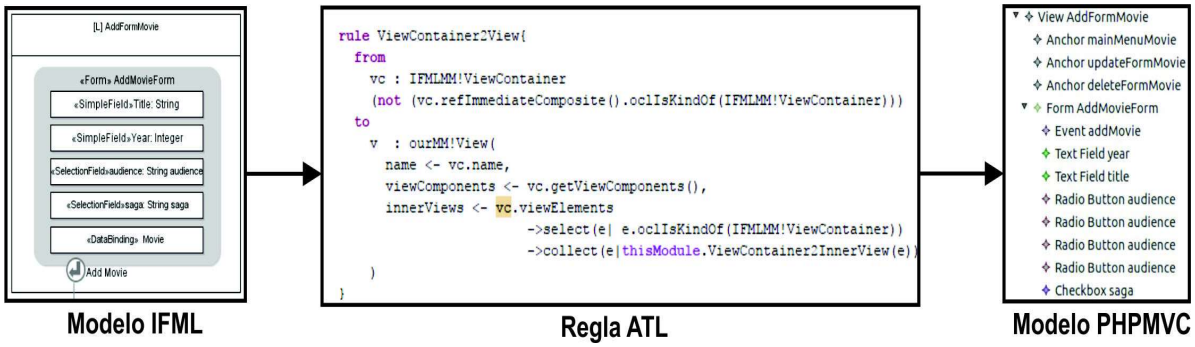

Figura 6.6. Proceso de Transformación M2M.

6.5. TRANSFORMACIÓN M2T

La salida del transformación anterior -Modelo a Modelo- se utiliza como entrada para esta transformación. En las sub-secciones 6.5.1 y 6.5.2 se muestran las vistas que resultan de la renderización del código obtenido como resultado de esta transformación.

6.5.1. Laravel

En la Figura 6.7 se muestra la estructura de directorios de los archivos generados y listo para ser integrados con el *framework* Laravel.

```
└── var
    └── www
        └── html
            └── laravel5.4.15
                ├── app
                │   ├── domainModelModels
                │   │   └── Movie.php
                │   └── Http
                │       ├── Controllers
                │       │   └── domainModelControllers
                │       │       └── MovieController.php
                │       └── Requests
                │           └── domainModelRequests
                │               └── MovieRequest.php
                ├── config
                │   └── app.php
                ├── resources
                │   └── views
                │       ├── InteractionFlowModelViews
                │       │   ├── AddFormMovie.blade.php
                │       │   ├── DeleteFormMovie.blade.php
                │       │   ├── MainMenuMovie.blade.php
                │       │   └── UpdateFormMovie.blade.php
                │       └── welcome.blade.php
                └── routes
                    └── web.php

16 directories, 10 files
```

Figura 6.7. Estructura de directorios del resultado de la transformación para Laravel.

En la Figura 6.8 se muestra la pantalla de bienvenida de la aplicación generada para Laravel. En la Figura 6.9, se muestra la vista principal de la aplicación generada para Laravel.

Figura 6.8. Vista de bienvenida de la aplicación generada para Laravel.

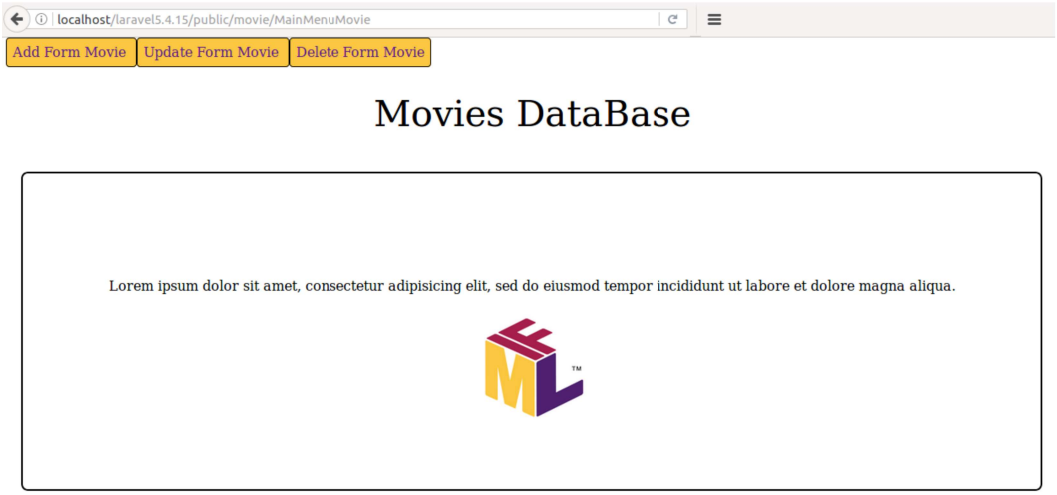

Figura 6.9. Menú Principal de la aplicación generada en Laravel

En la Figura 6.10 puede verse una captura del formulario generado para agregar una nueva película. En la Figura 6.11 puede verse el formulario generado para actualizar los datos de alguna película almacenada.

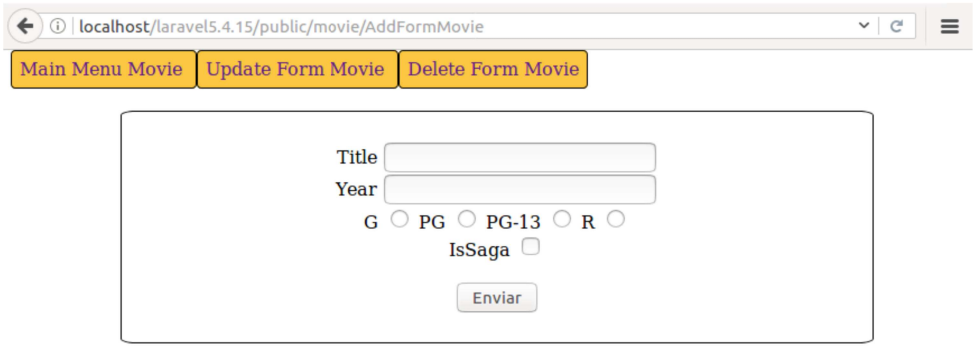

Figura 6.10. Formulario para agregar una nueva película en Laravel.

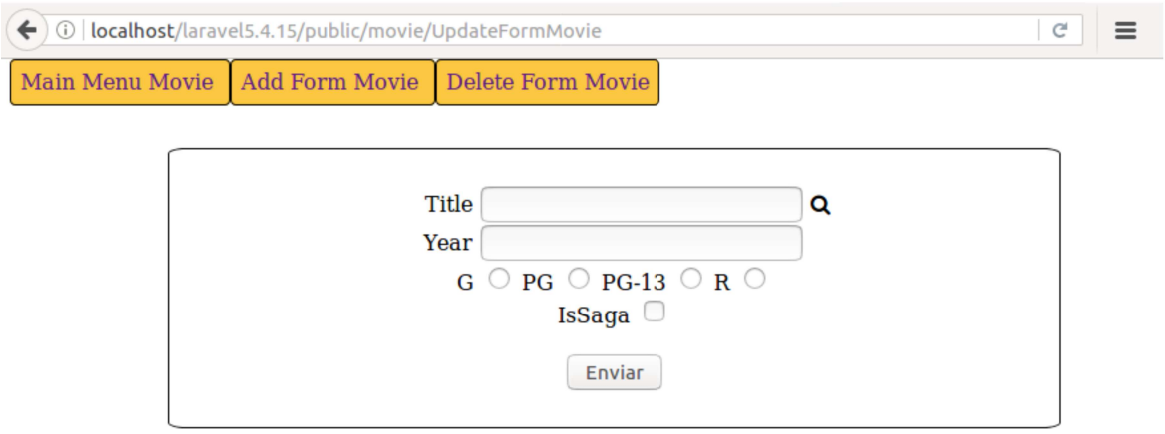

Figura 6.11. Formulario para actualizar una película en Laravel.

La Figura 6.12 muestra el formulario generado para eliminar una película.

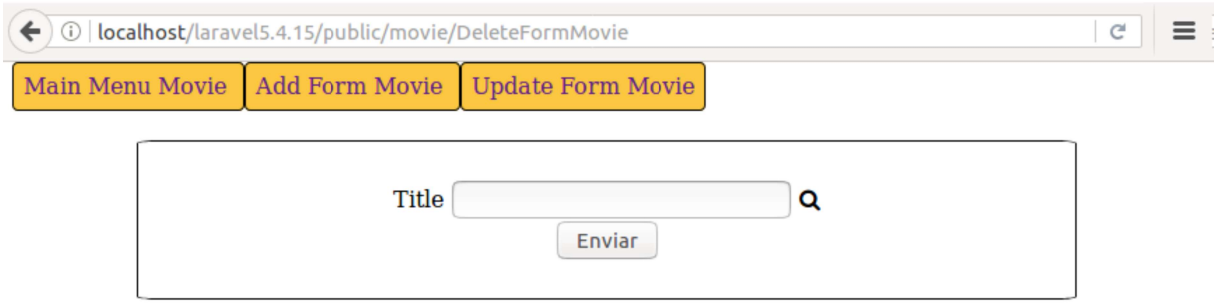

Figura 6.12. Formulario para eliminar una película en Laravel.

6.5.2. Yii2

La estructura directorios de los archivos generados y listo para ser integrados con el *framework* Yii2 puede verse en la Figura 6.13. En la Figura 6.14 puede verse la vista de bienvenida de la aplicación generada, mientra que en la Figura 6.15 se puede ver la vista principal de la aplicación Movies Database.

```
└── var
    └── www
        └── html
            └── yii2.0.12
                ├── config
                │   └── web.php
                ├── controllers
                │   └── domainModelControllers
                │       └── MovieController.php
                ├── models
                │   └── domainModelModels
                │       └── Movie.php
                └── views
                    ├── InteractionFlowModelViews
                    │   ├── AddFormMovie.php
                    │   ├── DeleteFormMovie.php
                    │   ├── MainMenuMovie.php
                    │   └── UpdateFormMovie.php
                    └── layouts
                        └── main.php

12 directories, 8 files
```

Figura 6.13. Estructura de directorios del resultado de la transformación para Yii2.

Figura 6.14. Vista de Bienvenida de la aplicación generada para Yii2.

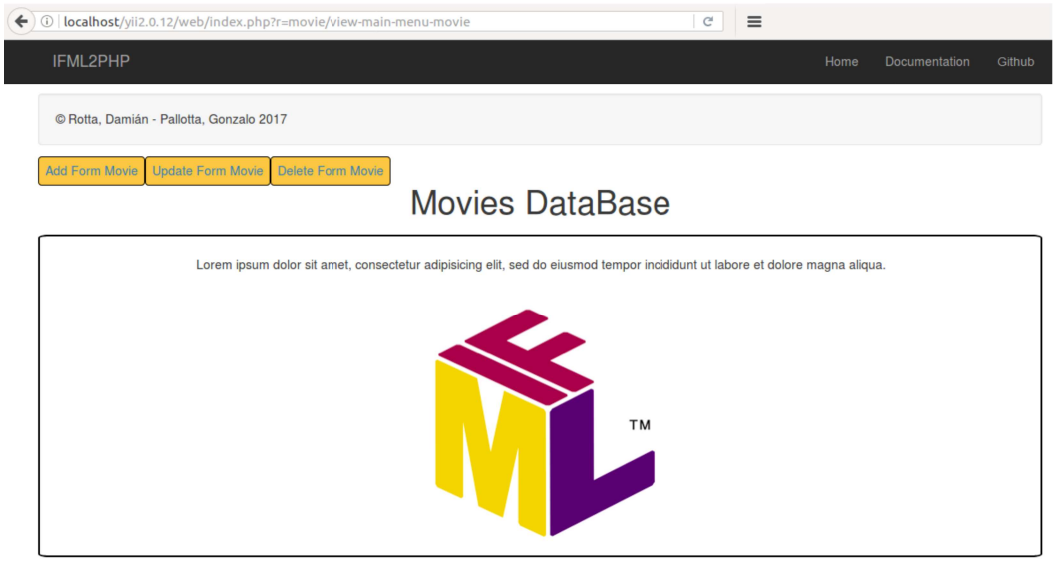

Figura 6.15. Vista principal de la aplicación generada en Yii2.

En la Figura 6.16 puede verse el formulario utilizado para agregar una nueva película. Mientras que en la Figura 6.17 puede verse el formulario utilizado para actualizar una película.

Figura 6.16. Vista del formulario para agregar una nueva película en Yii2.

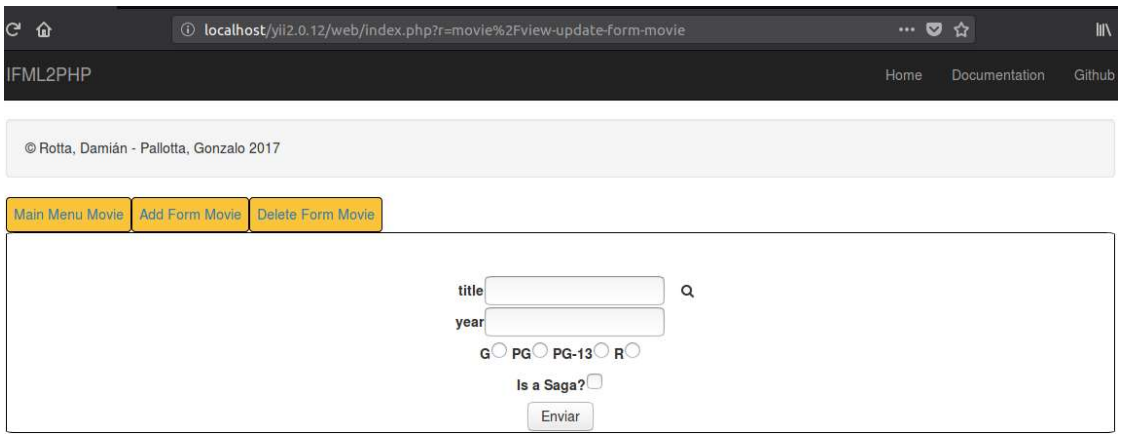

Figura 6.17. Vista del formulario para actualizar una película en Yii2.

En la Figura 6.18 se ve una vista del formulario generado para borrar una película.

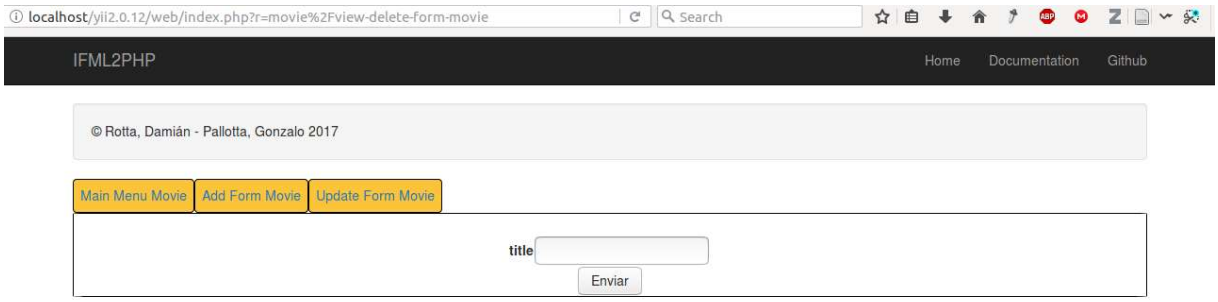

Figura 6.18. Vista del formulario para borrar una película en Yii2.

Continuando con el resumen iniciado Figura 6.6, en la Figura 6.19 se presenta la transformación M2T con un diagrama que ejemplifica el proceso completo. Tomando como entrada el modelo intermedio -en este caso una *View* llamada *AddFormMovie*- debe pasar por el código Acceleo tanto para Laravel como Yii2, teniendo como punto inicial el *template* llamado *makeView*. Como resultado final se presentan las páginas web que se obtienen como resultado.

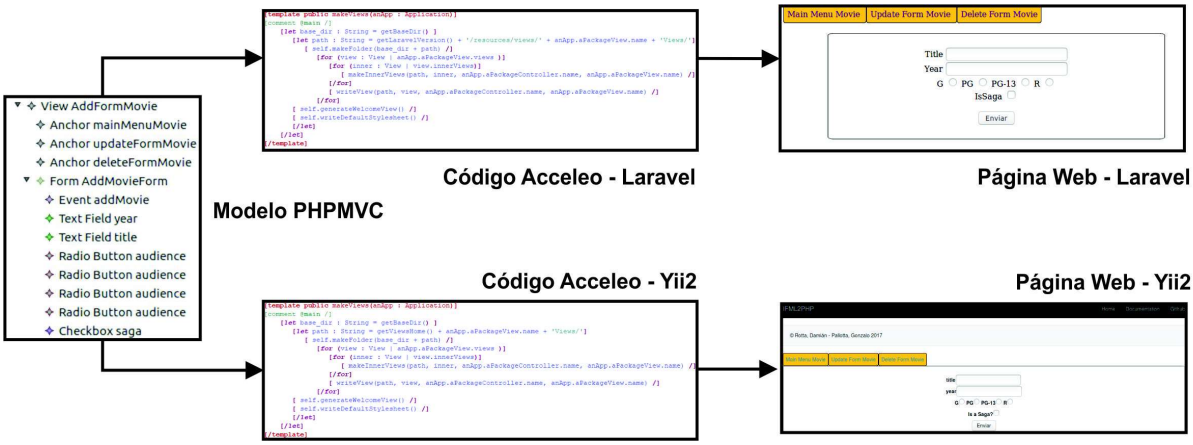

Figura 6.19. Proceso de Transformación M2T.

7. CONCLUSIÓN

El objetivo general de este trabajo fue plantear un metamodelo que constituya un modelo MVC Web para *frameworks* PHP que pueda integrarse a la metodología IFML. También se ha considerado apropiado incluir también las reglas de transformación M2M y M2T con el fin de verificar dicho metamodelo.

Durante las construcción y ejecución de las reglas de transformación se ha puesto en evidencia (tal como se menciona en [28]) la escasez de herramientas apropiadas y/o no discontinuadas en términos de soporte. Estos inconvenientes dieron pie a la idea de que sería importante llevar los proyectos de las transformaciones M2M, M2T y el metamodelo realizados en componentes software autónomos (como *plugins* para Eclipse) que puedan ser descargados y usados de manera independiente. Esto permitiría una reducción de esfuerzo en las primeras etapas del desarrollo tecnológico, en la medida en que facilite el uso de mecanismos de importación y resolución de URIs previstos y, por tanto, simplificados. Este es uno de los avances en la propuesta que facilitará en gran manera el abordaje de la cuestión de si es factible la integración de dos o más de metodologías de Ingeniería Web Dirigida por Modelos.

Se intuye esta idea de integración al considerar que los enfoques metodológicos existentes son múltiples y variados en cuanto a su nivel de alcance y abstracción y en cuanto al número de plataformas para las cuáles se genera el código. Este trabajo es parte de una línea de trabajo enfocada en la realización de dicha integración y muestra de ello es que el metamodelo propuesto aquí, al momento en que se está escribiendo este TFC, forma parte también del trabajo: *"Generación semiautomática de aplicaciones Web. De modelos en UWE a aplicaciones basadas en frameworks PHP CodeIgniter y Laravel"*, que persigue objetivos similares partiendo de un proceso de desarrollo conocido como UWE y generando código para otros *frameworks* PHP.

Por otra parte, cualquiera sea el enfoque metodológico, paradigma y/o técnicas utilizadas durante la construcción de un producto software y aún salvando todos los obstáculos conceptuales y tecnológicos que puedan presentarse se considera que los *tests* son necesarios. Contar con *tests* proveerá a los desarrolladores de ejemplos de como usar las reglas de transformación y/o el metamodelo provisto pero también de una manera de poder verificar que los futuros cambios no invaliden el marco de trabajo preexistente. Aunque éstas hayan sido las razones por las cuáles se escribieron un conjunto de *tests* del código HTML generado usando Selenium, estos *tests* actualmente no están siendo generados directamente desde los modelos.

7.1. TRABAJOS FUTUROS

Dado que la herramienta IFML2PHP es un producto que puede tener existencia independiente de este trabajo. Se consideró conveniente diferenciar trabajos futuros relacionados a mejoras específicas de la propuesta tecnológica y la continuidad conceptual de este TFC. A continuación, se listan dos propuestas para cada uno de estos enfoques, respectivamente.

- **Integración de Metodologías de Ingeniería Web Dirigida por Modelos existentes**

Actualmente los desarrolladores Web no pueden componer productos software en desarrollo que se encuentran en distintas fases y/o provengan de metodologías distintas. Se considera que lograr esta integración de metodologías podría significar un avance

importante para la disciplina MDWE en vistas a lograr dar soporte a todo el Ciclo de Vida de Desarrollo de Aplicaciones Web.

- **Utilización de métricas para la evaluación de calidad del metamodelo y de las reglas de transformación**

Las medidas presentadas en este TFC, tanto del metamodelo como de las reglas de transformación, pueden verse como una base sobre la cuál pueden calcularse métricas de calidad de que evalúen la propuesta de solución descripta en este documento.

- **Elaboración de una herramienta gráfica que integre los artefactos producidos**

Un *framework* con una interfaz gráfica puede facilitar en gran manera la usabilidad, adopción y extensibilidad de esta herramienta. El proyecto Eclipse GMP[25] y Apache ANT[26] pueden resultar de gran ayuda en la construcción de una interfaz gráfica y la automatización del proceso de *building*.

- **Generación semiautomática de tests**

Brambilla indica en [21] que, asumiendo la completitud de la generación de código y una confianza en el generador, el software derivado de los modelos no necesita ser probado. Sin embargo, *s*e considera que no se puede probar la completitud de la generación ni la tener confianza en el generador de código sin la ejecución de *tests*. Los *test* no asegurarán la correctitud pero minimizarán el riesgo de que el sistema generado no haga lo que debe hacer y/o haga lo que no debe. Respecto a la generación automática de *tests* a partir de los modelos, puede referirse a [73].

Para finalizar, se considera que cualquiera sea el *framework* construido que genere los *tests* semiautomáticamente debe considerar permitir:

- El análisis de los resultados obtenidos de la ejecución de los *tests* y la obtención de un reporte estandarizado a partir de ellos.
- Establecer un mecanismo claro de pre y postcondiciones para cada suite de *tests*.
- Permitir que los tests pueden ser compuestos a partir de otros *tests*.

25 http://www.eclipse.org/modeling/gmp/
26 http://ant.apache.org/

BIBLIOGRAFÍA

[1] S. Casteleyn, F. Daniel, P. Dolog, y M. Matera, *Engineering web applications*, vol. 30. Springer, 2009.

[2] G. Rossi, «Web Modeling Languages Strike Back», *IEEE Internet Comput.*, vol. 17, n.º 4, pp. 4-6, ago. 2013.

[3] A. E. Bell, «Death by UML Fever», *Queue*, vol. 2, n.º 1, pp. 72–80, mar. 2004.

[4] A. E. Bell, «UML Fever: Diagnosis and Recovery», *Queue*, vol. 3, n.º 2, pp. 48–56, mar. 2005.

[5] D. Rotta, G. Pallotta, H. Klikailo, y E. A. Belloni, «Un caso de estudio sobre la aplicación de UWE para la generación de Sistemas Web», presentado en XIX Concurso de Trabajos Estudiantiles (EST 2016)-JAIIO 45 (Tres de Febrero, 2016)., 2016, pp. 252-267.

[6] O. M. F. De Troyer y C. J. Leune, «WSDM: a user centered design method for Web sites», *Comput. Netw. ISDN Syst.*, vol. 30, n.º 1-7, pp. 85-94, 1998.

[7] «Metodología NDT», *Grupo IWT2*. [En línea]. Disponible en: http://iwt2.org/actividad-grupo/investigacion/resultados/ndt/. [Accedido: 28-mar-2017].

[8] «UWE - UML-based web engineering». [En línea]. Disponible en: http://uwe.pst.ifi.lmu.de/. [Accedido: 28-mar-2017].

[9] «IFML: The Interaction Flow Modeling Language | The OMG standard for front-end design». .

[10] «JavaServer Pages Technology». [En línea]. Disponible en: http://www.oracle.com/technetwork/java/javaee/jsp/index.html. [Accedido: 14-may-2017].

[11] «Entorno de desarrollo rápido para aplicaciones Móviles y Web», *WebRatio*. [En línea]. Disponible en: https://www.webratio.com/site/content/es/home. [Accedido: 14-may-2017].

[12] «UML2PHP5», *UML2PHP5*. [En línea]. Disponible en: http://uml2php5.zpmag.com/en/index.php. [Accedido: 07-jun-2017].

[13] «Enterprise Architect - Herramienta de modelado UML», *Enterprise Architect*. [En línea]. Disponible en: http://www.sparxsystems.com.ar/index.html. [Accedido: 07-jun-2017].

[14] O. Betari, M. Erramdani, S. Roubi, K. Arrhioui, y S. Mbarki, «Model Transformations in the MOF Meta-Modeling Architecture: From UML to CodeIgniter PHP Framework», en *Europe and MENA Cooperation Advances in Information and Communication Technologies*, Á. Rocha, M. Serrhini, y C. Felgueiras, Eds. Cham: Springer International Publishing, 2017, pp. 227-234.

[15] J. Cabot, *UMLtoX: Code generation scripts from UML (class diagrams) to SQL, PHP / Symfony and Python / Django (so far)*. 2017.

[16] M. Brambilla y P. Fraternali, *Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML*. Morgan Kaufmann, 2014.

[17] S. Murugesan, Y. Deshpande, S. Hansen, y A. Ginige, «Web Engineering: a New Discipline for Development of Web-Based Systems», en *Web Engineering: Managing Diversity and Complexity of Web Application Development*, S. Murugesan y Y. Deshpande, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 3-13.

[18] Prof. Dr.-Ing. Martin Gaedke, «Web Engineering Community Portal». [En línea]. Disponible en: http://www.webengineering.org/. [Accedido: 03-abr-2017].

[19] A. Ginige y S. Murugesan, «Web engineering: An introduction», *IEEE Multimed.*, vol. 8, n.º 1, pp. 14-18, 2001.

[20] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.
[21] M. Brambilla, J. Cabot, y M. Wimmer, «Model-driven software engineering in practice», *Synth. Lect. Softw. Eng.*, vol. 1, n.º 1, pp. 1-182, 2012.
[22] C. Pons, R. S. Giandini, y G. Pérez, «Desarrollo de software dirigido por modelos», 2010.
[23] M. Lang, «An analysis of model-driven web engineering methodologies», *Int. J. Innov. Comput. Inf. Control*, 2012.
[24] B. Marín, C. Gallardo, D. Quiroga, G. Giachetti, y E. Serral, «Testing of model-driven development applications», *Softw. Qual. J.*, vol. 25, n.º 2, pp. 407-435, jun. 2017.
[25] M. Botteck y T. Deiß, «Introduction of TTCN-3 into the product development process: considerations from an electronic devices developer point of view», *Int. J. Softw. Tools Technol. Transf. STTT*, vol. 10, n.º 4, pp. 285-289, 2008.
[26] S. A. Slaughter, D. E. Harter, y M. S. Krishnan, «Evaluating the cost of software quality», *Commun. ACM*, vol. 41, n.º 8, pp. 67-73, 1998.
[27] R. H. Thayer, J. B. Slaughter, B. W. Boehm, J. A. Clapp, J. H. Manley, y J. H. Burrows, «The high cost of software: causes and corrections», en *Proceedings of the May 6-10, 1974, national computer conference and exposition*, 1974, pp. 1009-1009.
[28] J. Miller y J. Mukerji, «Model driven architecture (MDA)», *Object Manag. Group Draft Specif. Ormsc2001-07-01*, 2001.
[29] Sommerville, Ian, *Software Engineering*, 9th ed. Pearson.
[30] «UML Web Site». [En línea]. Disponible en: http://www.uml.org/. [Accedido: 28-mar-2017].
[31] N. Moreno, S. Meliá, N. Koch, y A. Vallecillo, «Addressing New Concerns in Model-Driven Web Engineering Approaches», en *Web Information Systems Engineering - WISE 2008: 9th International Conference, Auckland, New Zealand, September 1-3, 2008. Proceedings*, J. Bailey, D. Maier, K.-D. Schewe, B. Thalheim, y X. S. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 426-442.
[32] F. Domínguez-Mayo, M. J. Escalona, M. Mejías, M. Ross, y G. Staples, «Quality evaluation for model-driven web engineering methodologies», *Inf. Softw. Technol.*, vol. 54, n.º 11, pp. 1265-1282, 2012.
[33] M. A. O. Mukhtar, M. F. B. Hassan, J. B. Jaafar, y L. A. Rahim, «WSDMDA: An Enhanced Model Driven Web Engineering Methodology», presentado en Control System, Computing and Engineering (ICCSCE), 2014 IEEE International Conference on, 2014, pp. 484-489.
[34] A. Fatolahi, S. S. Some, y T. C. Lethbridge, «Model-driven web development for multiple platforms», *J. Web Eng.*, vol. 10, n.º 2, p. 109, 2011.
[35] J. A. Aguilar *et al.*, «Techniques and Tools for Web Requirements in NDT, UWE and WebML».
[36] M. J. Escalona y N. Koch, «Requirements engineering for web applications-a comparative study», *J Web Eng*, vol. 2, n.º 3, pp. 193-212, 2004.
[37] M. Escalona, J. Torres, M. Mejías, M. Jurado, y L. Fillerat, «NDT: Navigational Development Techniques», *Publ. En Línea Dispon. Desde Internet Httplsi Ugr Es GedesactividadesDolmen4a11 Pdf Con Acceso En Oct. 2006*.
[38] J. A. O. Ramírez, M. J. E. Cuaresma, J. J. G. Rodríguez, M. P. Pérez, J. Ponce, y G. Aragón, «NDT-Suite, Una Solución Práctica Para El Uso de NDT», presentado en X Jornadas de ARCA. Sistemas Cualitativos y Diagnosis, Robótica, Sistemas Domóticos y Computación Ubicua, 2008, pp. 73-80.
[39] N. Koch, A. Knapp, G. Zhang, y H. Baumeister, «UML-based Web Engineering: An Approach based on Standards (book chapter)», *Web Eng. Model. Implement. Web Appl.*, pp. 157-191.

[40] M. Brambilla, S. Comai, P. Fraternali, y M. Matera, «Designing web applications with WebML and WebRatio», en *Web Engineering: Modelling and Implementing Web Applications*, Springer, 2008, pp. 221-261.
[41] S. Ceri, P. Fraternali, y A. Bongio, «Web Modeling Language (WebML): a modeling language for designing Web sites», *Comput. Netw.*, vol. 33, n.º 1, pp. 137-157, 2000.
[42] N. Moreno, P. Fraternali, y A. Vallecillo, «WebML modelling in UML», *IET Softw.*, vol. 1, n.º 3, pp. 67-80, 2007.
[43] I. Jacobson, G. Booch, J. Rumbaugh, J. Rumbaugh, y G. Booch, *The unified software development process*, vol. 1. Addison-wesley Reading, 1999.
[44] N. Koch, A. Kraus, y R. Hennicker, «The authoring process of the uml-based web engineering approach», presentado en First International Workshop on Web-Oriented Software Technology, 2001, pp. 1-29.
[45] N. Koch y A. Kraus, «The expressive power of uml-based web engineering», presentado en Second International Workshop on Web-oriented Software Technology (IWWOST02), 2002, vol. 16.
[46] M. Brambilla y P. Fraternali, «Interaction Flow Modeling Language (IFML) 1.0. OMG Standard Spec.», feb. 2015.
[47] G. Rossi, M. Urbieta, D. Distante, J. M. Rivero, y S. Firmenich, «25 Years of Model-Driven Web Engineering: What we achieved, What is missing», *CLEI Electron. J.*, vol. 19, n.º 3, p. 1:1-1:29, dic. 2016.
[48] Obeo, «ATL». [En línea]. Disponible en: https://eclipse.org/atl/. [Accedido: 04-abr-2017].
[49] J. Siedersleben y E. Denert, «Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur», *Inform.-Spektrum*, vol. 23, n.º 4, pp. 247-257, 2000.
[50] K. Tatroe, P. MacIntyre, y R. Lerdorf, *Programming Php*. O'Reilly Media, Inc., 2013.
[51] W. Cui, L. Huang, L. Liang, y J. Li, «The research of PHP development framework based on MVC pattern», presentado en Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on, 2009, pp. 947-949.
[52] K. Henney, Ed., *97 things every programmer should know: collective wisdom from the experts*, 1st ed. Sebastopol, Calif: O'Reilly, 2010.
[53] J. Longuski, «Keep It Simple, Stupid», *Seven Secrets Think Rocket Sci.*, pp. 85-86, 2007.
[54] J. Miller, «Patterns in Practice - Convention Over Configuration», *MSDN Mag.*, vol. 24, n.º 02, feb. 2009.
[55] R. Das y L. P. Saikia, «Comparison of Procedural PHP with Codeigniter and Laravel Framework», *Int. J. Curr. Trends Eng. Res.*, vol. 02, n.º 06, pp. 42-48, jun. 2016.
[56] Yii Software LLC, «Yii PHP Framework», *Yii PHP Framework: Best for Web 2.0 Development*. [En línea]. Disponible en: http://www.yiiframework.com/. [Accedido: 03-abr-2017].
[57] T. Otwell, «Laravel», *Laravel - The PHP Framework For Web Artisans*. [En línea]. Disponible en: https://laravel.com/. [Accedido: 03-abr-2017].
[58] R. S. Pressman y D. B. Lowe, *Web engineering: a practitioner's approach*. Boston: McGraw-Hill Higher Education, 2009.
[59] S. Dart, «Content Change Management: Problems for Web Systems», en *System Configuration Management: 19th International Symposium, SCM-9 Toulouse, France, September 5–7, 1999 Proceedings*, J. Estublier, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 1-16.
[60] G. Kacmarcik y T. Leithead, «UI Events», *World Wide Web Consortium*. [En línea]. Disponible en: https://www.w3.org/TR/uievents/. [Accedido: 18-sep-2017].

[61] V. A. Bollati, «MeTAGeM: Entorno de Desarrollo de Transformaciones de Modelos Dirigido por Modelos», Universidad Rey Juan Carlos, 2011.
[62] C. Larman, *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Pearson Education India, 2012.
[63] Software & Systems Engineering Committee, «IEEE standard for software and system test documentation», *Fredericksburg VA USA IEEE Comput. Soc.*, 2008.
[64] M. Monperrus, J.-M. Jézéquel, J. Champeau, y B. Hoeltzener, «Measuring models», *Model-Driven Softw. Dev. Integrating Qual. Assur.*, pp. 147-168, 2008.
[65] W. James *et al.*, «What do metamodels really look like?», *Front. Comput. Sci.*, 2013.
[66] J. Di Rocco, D. Di Ruscio, L. Iovino, y A. Pierantonio, «Mining Metrics for Understanding Metamodel Characteristics», en *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, New York, NY, USA, 2014, pp. 55–60.
[67] H. Ma, W. Shao, L. Zhang, Z. Ma, y Y. Jiang, «Applying OO Metrics to Assess UML Meta-models», en *«UML» 2004 — The Unified Modeling Language. Modeling Languages and Applications: 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings*, T. Baar, A. Strohmeier, A. Moreira, y S. J. Mellor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 12-26.
[68] Z. Ma, X. He, y C. Liu, «Assessing the quality of metamodels», *Front. Comput. Sci.*, vol. 7, n.º 4, pp. 558-570, ago. 2013.
[69] A. Vignaga, «Metrics for measuring ATL model transformations», *Univ. Chile Tech. Rep. TRDCC-2009-6*, 2009.
[70] J. Di Rocco, D. Di Ruscio, L. Iovino, y A. Pierantonio, «Mining Correlations of ATL Model Transformation and Metamodel Metrics», en *Proceedings of the Seventh International Workshop on Modeling in Software Engineering*, Piscataway, NJ, USA, 2015, pp. 54–59.
[71] M. F. van Amstel y M. Van Den Brand, «Quality assessment of ATL model transformations using metrics», en *Proceedings of the 2nd International Workshop on Model Transformation with ATL (MtATL 2010), Malaga, Spain (June 2010)*, 2010, p. 115.
[72] M. F. van Amstel y M. van den Brand, «Using metrics for assessing the quality of ATL model transformations», en *Proceedings of the Third International Workshop on Model Transformation with ATL (MtATL 2011)*, 2011, vol. 742, pp. 20-34.
[73] B. Marín, C. Gallardo, D. Quiroga, G. Giachetti, y E. Serral, «Testing of model-driven development applications», *Softw. Qual. J.*, vol. 25, n.º 2, pp. 407-435, jun. 2017.
[74] «BuildWith - Framework usage in Argentina», *BuildWith*. [En línea]. Disponible en: https://trends.builtwith.com/framework/country/Argentina. [Accedido: 24-oct-2017].
[75] J. A. Blanco y D. Upton, *CodeIgniter 1.7*. Birmingham, U.K.: Packt Pub., 2009.
[76] «Google Trends», *Google Trends*. [En línea]. Disponible en: https://trends.google.com/trends/explore?date=today%205-y&q=codeigniter,laravel,yii. [Accedido: 02-feb-2018].
[77] Karzan Wakil and Dayang N.A. Jawawi, «Analyzing Interaction Flow Modeling Language in Web Development Lifecycle», presentado en International Journal of Advanced Computer Science and Applications, 2017, vol. 8, p. 8.
[78] S. Abeysinghe, *PHP Team Development*. Birmingham, UK; Mumbai: Packt Pub., 2009.
[79] «BuildWith - Framework technologies Web Usage Statistics», *BuildWith*. [En línea]. Disponible en: https://trends.builtwith.com/framework. [Accedido: 24-oct-2017].

A. IFML: NOTACIÓN

En este Anexo se presentaran las distintas vistas que componen al metamodelo de IFML. Describiendo cada una de ellas, las clases que la componen y las relaciones existente entre ellas.

A.1. IFML MODEL

En la Figura A.1 se presenta el metamodelo de IFML Model.

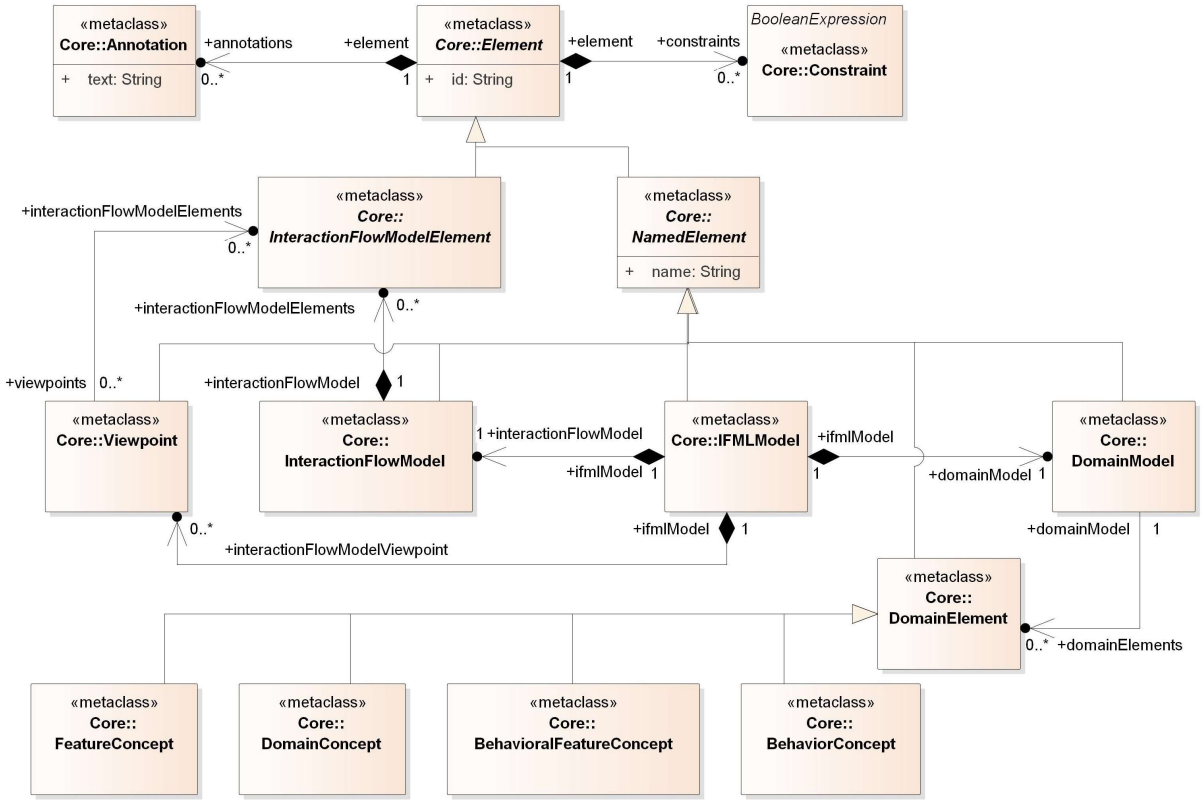

Figura A.1. Metamodelo - IFML Model.

Es el contenedor de todo el resto de elementos de modelado. Contiene un *InteractionFlowModel*, un *DomainModel* y, opcionalmente, *ViewPoints*.

InteractionFlowModel es la vista del usuario de la aplicación como un todo, mientras que los ViewPoints presentan solo aspectos específicos pero complementamente funcionales.

El propósito de un *ViewPoint* es facilitar la comprensión de un sistema complejo, permitir o prohibir el acceso al sistema a un *UserRole* específico o mostrar una pieza del sistema adaptada a un cambio específico de contexto.

Un *DomainModel* representa una vista de lógica de negocios de la aplicación. Es decir, el contenido y el comportamiento que es manejado y referenciado en el *InteractionFlowModel*. El *DomainModel* está compuesto de *DomainElements*, que se

especializan como conceptos (*DomainConcept*), propiedades (FeatureConcept) , comportamientos (*BehaviorConcept*) y métodos (*BehavioralFeatureConcept*).

NamedElement es una clase abstracta que especializa la clase *Element* (la clase más general del modelo) denotando los elementos que tienen un nombre. *NamedElement* también tiene sus subclases, pero serán descriptas en contextos donde juegan un rol mayor.

Para cualquier *Element*, pueden especificarse *Constraints* y *Annotations.*

Las *Constraints* son un mecanismo de extensión de IFML en el sentido que pueden enriquecer, para un modelo específico, las reglas sintácticas existentes en IFML.

A.2. INTERACTION FLOW MODEL

En la Figura A.2 se presenta el metamodelo de Interaction Flow Model.

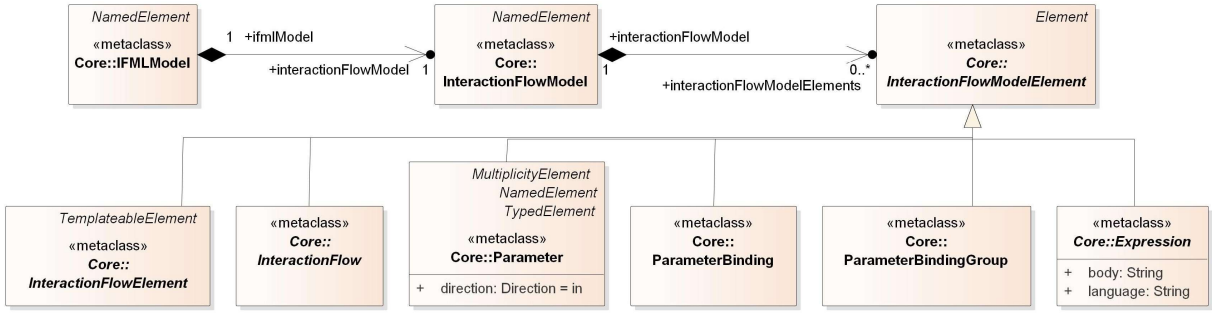

Figura A.2. Metamodelo - Interaction Flow Model

Contiene todos los elementos de la vista del usuario de la aplicación representados por *InteractionFlowModelElement*s. Los *InteractionFlowModelElements* tienen 7 subtipos directos: *InteractionFlowElement*, *InteractionFlow*, *ParameterBindingGroup*, *ParameterBinding*, *Parameter*, *Expression* y *Module*.

Los *InteractionFlowElement*s son los bloques de construcción de las interacciones. Representan piezas del sistema que participan en flujos de interacción a través de *InteractionFlows.*

Un *InteractionFlow* es una conexión dirigida entre dos *InteractionFlowElements*. Los *InteractionFlows* pueden implicar navegación a través de la interfaz del usuario o solo una transferencia de información llevando parámetros de un *InteractionFlowElement* a otro.

Un *Parameter* es un nombre tipado, cuyas instancias guardan valores. Los parámetros se realizan mediante *InteractionFlowElements*, es decir, *ViewElements*, *ViewComponentParts*, *Ports* y *Actions*. Los *Parameters* fluyen entre *InteractionFlowElements* cuando se disparan los *Events.*

Los *ParameterBindings* determinan a qué *Parameter* de entrada de un *InteractionFlowElement* destino está ligado un *Parameter* de salida de un *InteractionFlowElement* origen. Los *ParameterBindings* se agrupan en *ParameterBindingGroups*.

Un *Module* es un colección completamente funcional de *InteractionFlowModelElements* que pueden ser reutilizados en vistas de mejorar la

mantenibilidad de IFML. Los *Modules* pueden ser reemplazados por otros *Modules* o *InteractionFlowElements* con los mismos parámetros de entrada y salida.

Una *Expression* define un enunciado que será evaluado, en un contexto dado, produciendo una instacia, un conjunto de instacias o un resultado vacío. Una *Expression* está libre de efectos colaterales. Los tipos específicos de expresiones, tales como expresiones booleanas, son representadas como especializaciones de *Expression*.

A.3. INTERACTION FLOW ELEMENTS

En la Figura A.3 se presenta el metamodelo de Interaction Flow Elements.

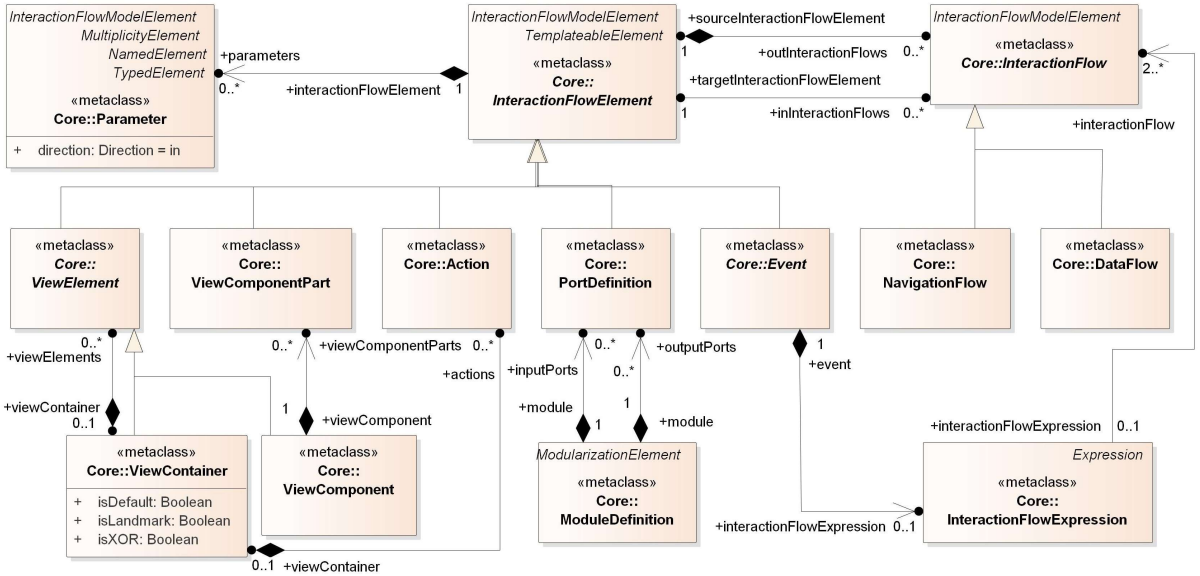

Figura A.3. Metamodelo - Interaction Flow Elements

Cuando se navega a través de un *NavigationFlow* se pueden pasar parámetros desde un *InteractionFlowElement* origen a un *InteractionFlowElement* destino a través de *ParameterBindings*. Un *DataFlow* es un tipo de *InteractionFlow* utilizado para pasar información contextual entre *InteractionFlowElements*. Los *DataFlows* son disparados por *NavigationFlows*, causando paso de parámetros pero no navegación.

Los *Events* puede estar asociados con una *InteractionFlowExpression* tienen más de un *NavigationFlow* saliente.

Las *InteractionFlowExpressions* se utilizan para determinar que *InteractionFlows* se seguirán como consecuencia de la ocurrencia de un *Event*. Cuando ocurre un *Event* y no tiene una *InteractionFlowExpression* relacionada, se siguen todos los *InteractionFlows* asociados con el evento.

Los *ViewContainers* pueden contener *ViewElements* (*ViewContainers* o *ViewComponents*) o *Actions*.

A.4. VIEW ELEMENTS

En la Figura A.4 se presenta el metamodelo de View Elements.

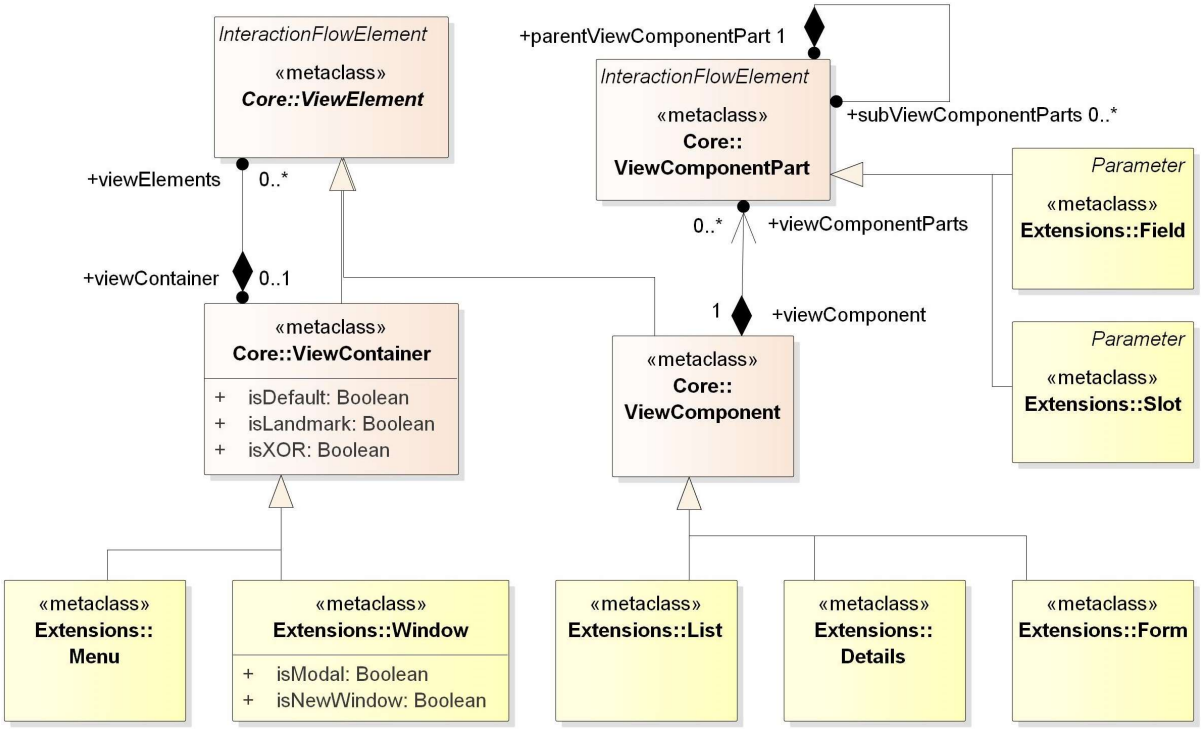

Figura A.4. Metamodelo - View Elements

Los elementos de un modelo IFML que son visibles a nivel de la intefaz del usuario se llaman *ViewElements*, los cuales se especializan en *ViewContainers* y *ViewComponents*. Los *ViewContainers*, como páginas HTML o ventanas, son contenedores de otros *ViewContainers* o *ViewComponents*, mientras que los *ViewComponents* son elementos de la interfaz que muestran contenido o aceptan datos desde el usuario. Un *ViewContainer* puede ser *landmark*, XOR y/o *default*. Un *ViewContainer landmark* puede ser alcanzado desde culaquier otro *ViewElement* sin necesidad de *InteractionFlows* explícitos. Los *ViewContainers* que no son *landmark* pueden ser alcanzados solo a través de un *InteractionFlow*.

En el caso de un *ViewContainer* (el *ViewContainer* contenido) que es parte de otro *ViewContainer* (el *ViewContainer* contenedor), como un *frame* en una página HTML, si está marcado como *default*, será presentado al usuario en el momento en el que accede al *ViewContainer* contenedor. Los *ViewContainers* contenidos pueden ser marcados como XOR. En este caso, los *ViewElements* contenidos en ese *ViewContainer* serán presentados al usuario solo uno a la vez, a medida que el usuario interactúa con el sistema. Un *ViewContainer* puede ser también abierto como una ventana nueva. Esta ventana nueva puede ser *modal.* Las ventanas modales bloquean toda otra interacción del usuario en los contenedores activos previos, hasta que se cierra la ventana nueva. Otro tipo especial de *ViewContainer* es el *Menu*. Los *Menus* representan conjuntos de botones interacivos o enlaces que llevan a contenedores destino. Los *Menus* no pueden contener subcontenedores o *ViewComponents*.

Los *ViewComponents* solo existen dentro de *ViewContainers*. Un *ViewComponent* es un elemento de la interfaz que puede tener comportamiento dinámica, mostrar contenido o aceptar entradas. Puede corresponder, por ejemplo, a un formulario, una grulla o una galería de imágenes.

Un *ViewComponent* puede estar construido a partir de *ViewComponentParts*. Una *ViewComponenentPart* es una parte del *ViewComponent* que no existe fuera del contexto de éste pero puede tener *Events* e *InteractionFlows* entrantes y salientes. Las *ViewComponentParts* pueden contener jerárquicamente otras *ViewComponentParts*.

Una *ViewComponentPart* puede ser visible o no al nivel de la interfaz de usuario dependiendo del tipo de *ViewComponentPart*. Por ejemplo, un *RichTextField* es una *ViewComponentPart* que es visible al usuario, puede disparar eventos y puede recibir valores a través del paso de parámetros, mientras que un *Slot* es un *placeholder* de valores que no es visible al usuario.

El paquete *Extension* incluye ejemplos concretos de *ViewComponents* -tales como *List*, *Details* y *Form*- y *ViewComponentParts* tales como *Fields* y *Slots*. Una *List* permite mostrar, seleccionar y capturar listas de items del mismo tipo. Los *Details* son componentes que permiten mostrar información detallada del elemento contenido. Un *Form* permite capturar entradas del usuario a través de formularios.

A.5. PARAMETERS

En la Figura A.5 se presenta el metamodelo de Parameters.

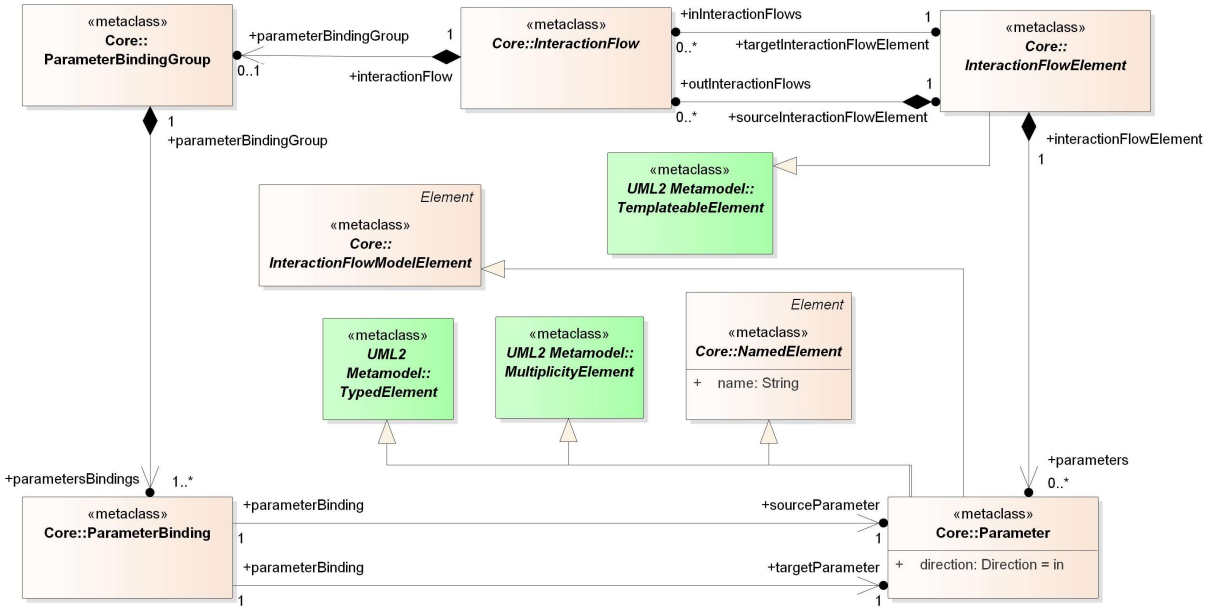

Figura A.5. Metamodelo - Parameters

Un *Parameter* es un elemento tipado y con multiplicidad, cuyas instancias mantienen valores. Un *Parameter* puede ser de tipo primitivo o complejo, tales como objetos o colecciones de objetos. Los *Parameters* son realizados por InteractionFlowElements y fluyen a través de ellos cuando se disparan *Events*. Los *Parameters* pueden ser mapeados a un

solo elemento de la interfaz de usuario, es decir, a un *ViewComponentPart* o a un complejo conjunto jerárquico de *ViewComponentParts*.

Los *Parameters* tienen una propiedad de dirección, que puede ser de entrada (in), salida (*out*) o entrada-salida (*inout*). La dirección por defecto es de entrada. Un *Parameter* de entrada permite a un *InteractionFlowElement* recibir uno o más valores a través de *NavigationFlows* o *DataFlows* entrantes. Un *Parameter* saliente permite a un *InteractionFlowElement* exponer uno o más valores a través de un *NaveigationFlow* o *DataFlow* saliente. Un *Parameter* entrada-salida permite ambos comportamientos.

Un *ParameterBinding* determina a que *Parameter* de entrada de un *InteractionFlowElement* destino está conectada un *Parameter* de salida de un *InteractionFlowElement* origen y como el valor de éste parámetro fluye cuando se dispara un *Event* y se sigue un *InteractionFlow*. Los *ParameterBinding* que fluyen juntos con un *InteractionFlow* están agrupados en *ParameterBindingGroups*, relacionados a ese *InteractionFlow*.

A.6. EVENTS

En la Figura A.6 se presenta el metamodelo de Events.

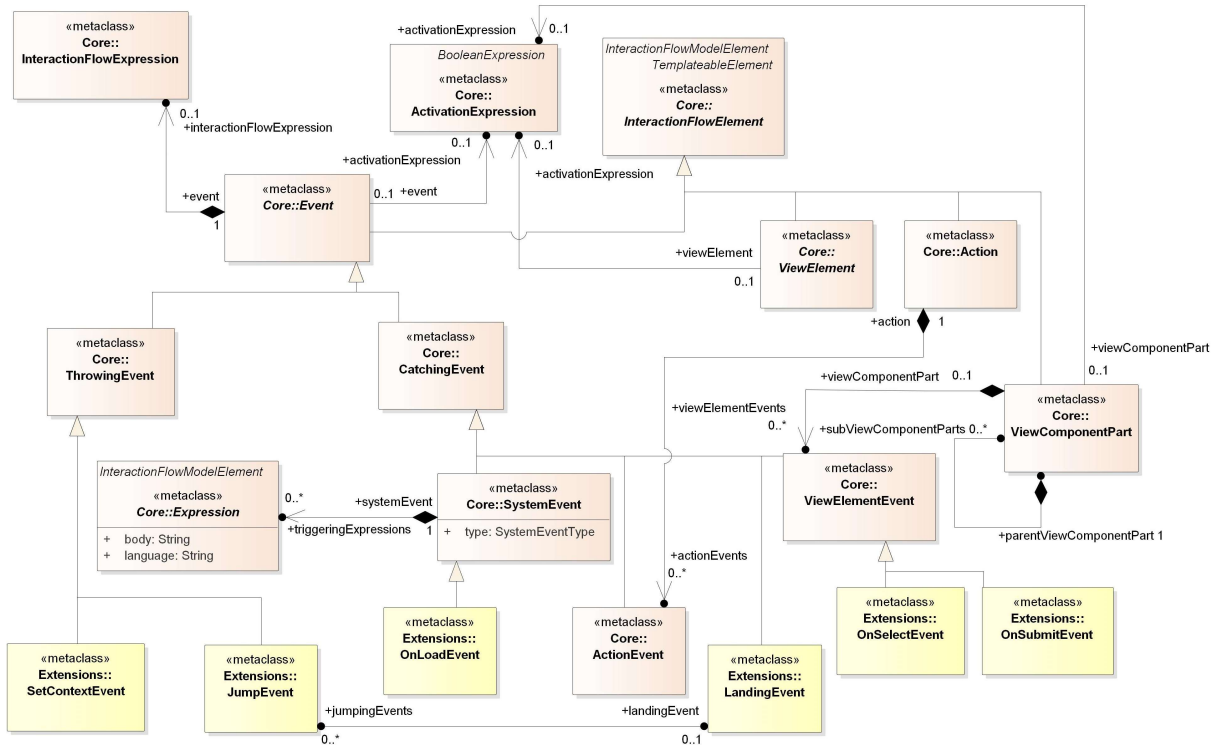

Figura A.6. Metamodelo - Events

Los *Events* son ocurrencias que pueden afectar el estado de la aplicación y son un subtipo de un *InteractionFlowElement*. Los *Events* se clasifican en dos categorías principales: *CatchingEvents* (eventos que son capturados en la IGU y que disparan un cambio de interfaz subsequente) y *ThrowingEvents* (eventos que son generados por la IGU). Hay 3 tipos de *CatchingEvents*: *ViewElementsEvents*, que son el resultado de una

interacción del usuario (con subtipos específicos *OnSelectEvent* y *OnSubmitEvent*), *ActionEvents* y *SystemEvents* (como *OnLoadEvent*).

Los *ViewElementEvents* pertenecen a sus *ViewElements* relacionados. Esto significa que los *ViewElements* contienen *Events* que le permiten al usuario activar una interaccion en la aplicación. Por ejemplo, con el click en un hipervínculo o un botón. Los *ActionEvents* pertenecen a sus *Actions* relacionadas. Una *Action* puede disparar *ActionEvents* durante su ejecución o cuando termina, normalmente o con una excepción.

Los *SystemEvents* son eventos autónomos, que están al nivel del *InteractionFlowModel*. Los *SystemEvents* resultan del evento de terminación de la ejecución de una *Action* o una *triggeringExpression* tales como el arribo de un determinado momento en el tiempo, o eventos especiales como un problema en la conexión de la red.

Los *CatchingEvents* poseen un conjunto de *NavigationFlows*. Un *InteractionFlowExpression* se utiliza para determina que *NavigationFlows* son seguidos como consecuencia de la ocurrencia de un *Event*. Cuando ocurre un *Event* y no tiene una *InteractionFlowExpression*, se siguen todos los *NavigationFlows* asociados con el evento.

Un *Event* puede tener una *ActivationExpression* que determina si un *Event* está habilitado o deshabilitado. En términos prácticos, deshabilitar un *ViewElementEvent* significa, por ejemplo, que el elemento IGU (por decir, un botón) que dispara un *InteractionFlow* está deshabilitado.

A.7. EXPRESSIONS

En la Figura A.7 se presenta el metamodelo de Expressions.

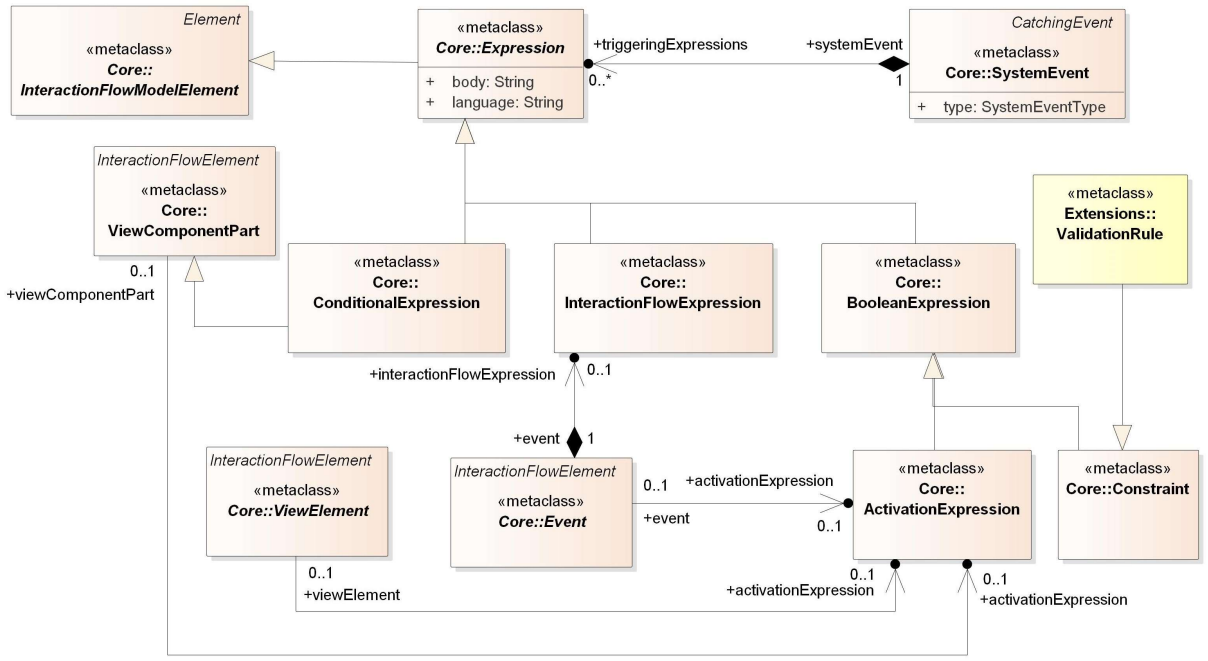

Figura A.7. Metamodelo - Expressions

Una *Expression* define un enunciado libre de efectos colaterales que será evaluado en un contexto dado de una sola instancia, un conjunto de instancias o un resultado vacío.

Los subtipos de *Expression* son *InteractionFlowExpressions*, *BooleanExpressions* y *ConditionalExpressions*.

Una *InteractionFlowExpression*, determina que *NavigationFlow* debería ser seguido cuando más de un *NavigationFlow* que sale de un *Event*.

Una *ConditionalExpression* es una *ViewComponentPart* que representa consultas predefinidas contenidas en *DataBindings* que pueden ejecutados en ellas para obtener información específica desde el *DomainModel*. Las *ConditionalExpression* puede ser definidas solo dentro de un *DataBinding ViewComponentPart*.

Una *BooleanExpression* es una expresión que puede ser verdadera o falsa. Las *BooleanExpression* tienen como especializaciones a *ActivationExpression* y *Constraint*. Una *ActivationExpression* determina si un *ViewElement*, un *ViewComponentPart* o un *Event* está habilitado y disponible para la interacción con el usuario, mientras que una *Constraint* restringe el comportamiento de cualquier elemento.

Los valores utilizados para evaluar las expresiones están definidos dependiendo del tipo específico de la *Expression*. Por ejemplo, las expresiones *SystemEvent* pueden tener como ámbito valores de condición específicos del sistema, la fecha y tiempo actual, etc. Esto no se modela en IFML.

A.8. CONTENT BINDING

En la Figura A.8 se presenta el metamodelo de Content Binding.

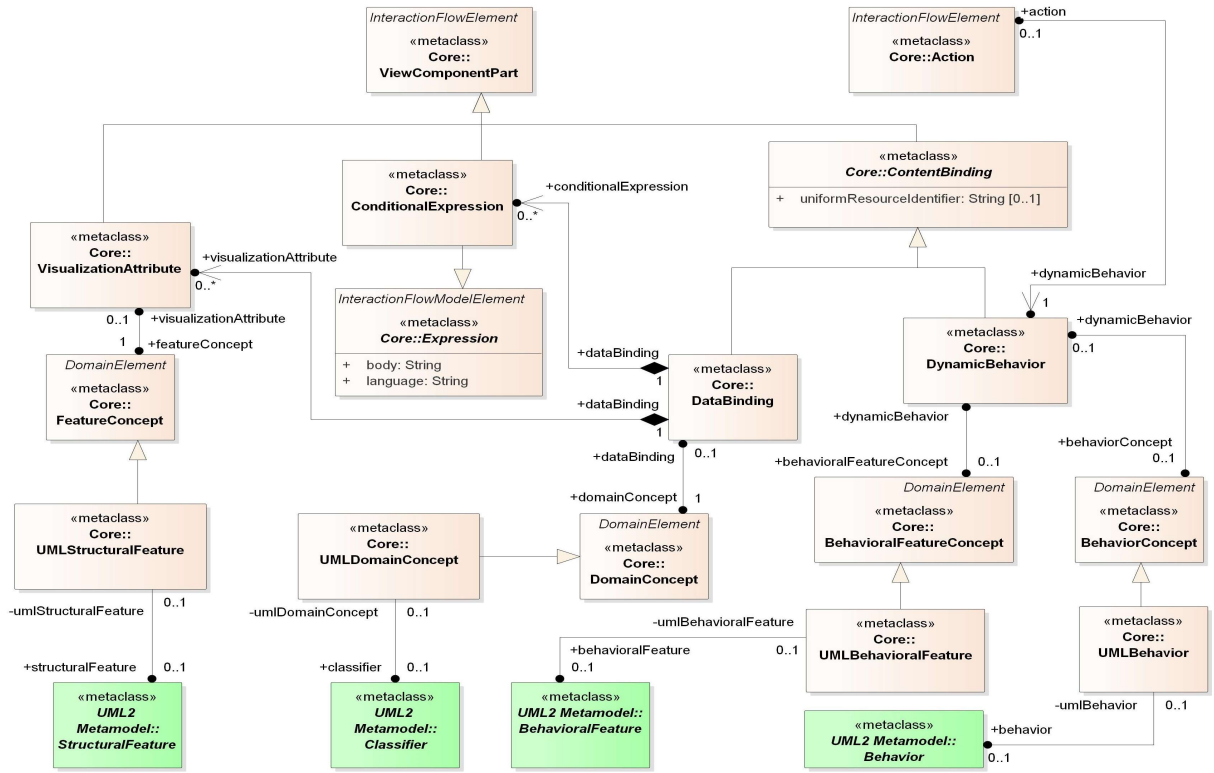

Figura A.8. Metamodelo - Content Binding

Los *ViewComponents* pueden devolver contenido mediante *ContentBindings*. Los *ContentBinding* representen cualquier fuente de contenido. Los *ContentBinding* tienen como atributo opcional la URI del recurso desde el cual se obtiene el contenido.

Los *ContentBindings* se especializan en dos conceptos: *DataBindings* y *DynamicBehavior*. Un *DataBinding* referencia un *DomainConcept* (por ejemplo, un *Classifier* en UML) que puede representar un objeto, un archivo XML, una tabla en la base de datos, etc.

Un *DataBinding* está asociado con una *ConditionalExpression*, que determina el contenido específico que debe ser obtenido de una fuente de contenido. Un *DynamicBehavior* representa un acceso al contenido o lógica de negocios tales como un servicio o un método que retorna un resultado después de una invocación, como lo representa *BehavioralFeature* o *Behavior* en UML.

Un *DataBinding* contiene *VisualizationAttributes* usados por los *ViewComponents* para determinar los atributos accedidos desde el *DataBinding* que pueden ser mostrados al usuario tales como una columna de una base de datos o un elemento XML, como lo representan los *StructuralFeatures* de UML.

A.9. CONTEXT

En la Figura A.9 se presenta el metamodelo de Context.

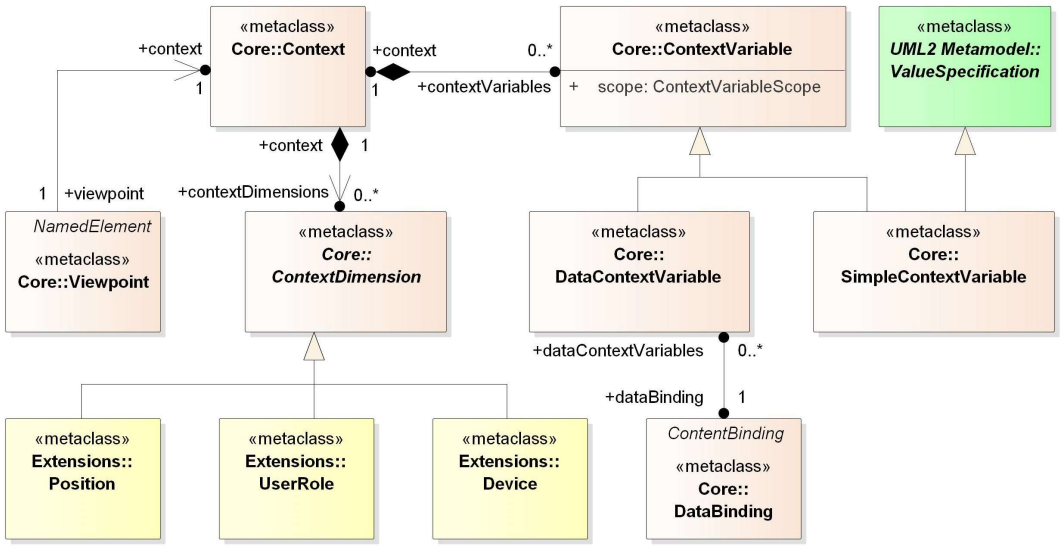

Figura A.9. Metamodelo - Context

El *Context* es un aspecto de *runtime* del sistema que determina como debería estar configurada la interfaz de usuario y como puede mostrarse el contenido. La configuración y el contenido de la interfaz de usuario están determinados por un ViewPoint. Luego, un *Context* está relacionado a un *ViewPoint*.

Un *Context* tiene varias dimensiones llamadas *ContextDimension*, que representan no solo el id del usuario y sus preferencias si no también la interacción con el entorno del sistema. Una *ContextDimension* se especializa en *UserRole*, *Device* y *Position*. Cuando el

contexto del usuario satisface todas las *ContextDimensions*, se le permite al usuario el acceso a los *ViewElements* del *ViewPoint* y a los *Events* que son disparados en ellos.

El *UserRole* representa el perfil que un usuario debería tener para satisfacer la dimensión *UserRole*.

Un *Device* representa un tipo específico de dispositivo para el cuál está configurado el *ViewPoint*. Cuando un usuario accede a la aplicación mediante este dispositivo, se satisface la dimensión *Device*.

Una *Position* representa la localización y orientación del dispositivo para el cual la *ViewPoint* fue configurada. Cuando el dispositivo utilizado por el usuario para acceder a la aplicación alcanza dada posición u orientación, se satisface la dimensión *Position*.

Una *ContextDimension* puede ser especializada en otras dimensiones, como preferencias del usuario, etc.

Las *ContextVariables* pueden estar asociadas al *Context* para guardar valores primitivos (*SimpleContextVariable*) u objetos (*DataContextVariable*) que guarden el estado del sistema en el contexto actual.

A.10. SPECIFIC VIEWCOMPONENT

En la Figura A.10 se presenta el metamodelo de Specific ViewComponent.

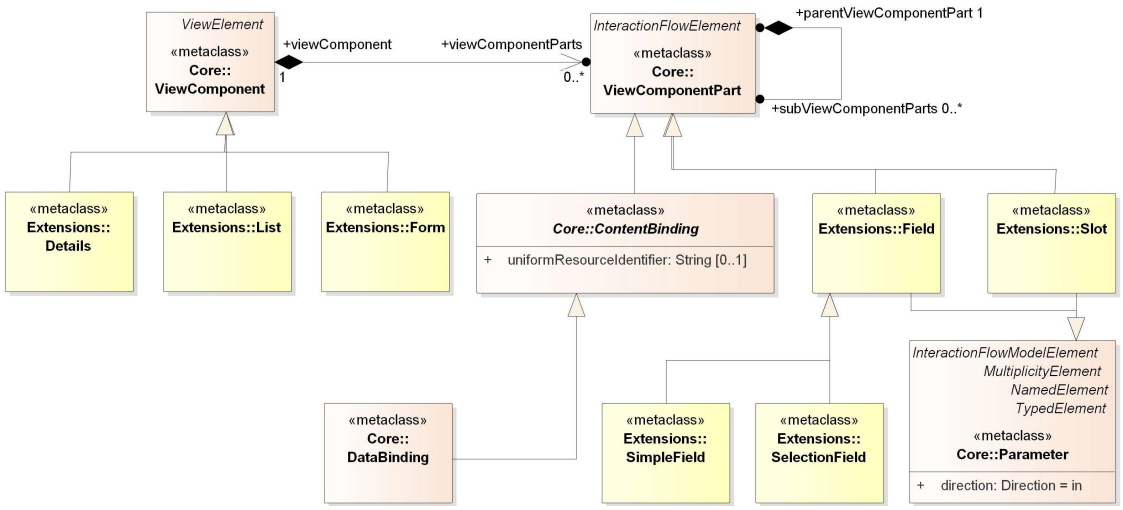

Figura A.10. Metamodelo - Specific ViewComponent

IFML incluye un conjunto básico de extensiones a los elementos del Core que ejemplifican como puede ser extendido IFML.

Las *List*, *Details* y *Forms* son especializaciones de *ViewComponent*. El *ViewComponent List* es utilizado para mostrar una lista de instancias de *DataBindings*. Cuando un *ViewComponent List* está asociado con un *Event*, significa que cada instancia *DataBinding* mostrada por el componente puede disparar ese *Event*. El *Event* a su vez causa el pasaje de parámetros mapeado desde una instancia *DataBinding* a un *InteractionFlowElement* destino. El *ViewComponent Details* es utilizado para mostrar información detallada de una instancia *DataBinding*. Cuando el *ViewComponent Details* está asociado con un *Event*, el disparo de dicho *Event* causa el pasaje de parámetros mapeado desde una instancia *DataBinding* a un *InteractionFlowElement* destino.

El *ViewComponent Form* es utilizado para mostrar un formulario, el cuál está compuesto de *Fields* que pueden mostrar o capturar contenido del usuario. Los *Fields* tienen *Slots* que guardan su valor. Cuando el *Field* es un *SelectionField*, sus *Slots* asociados contienen las opciones de selección disponibles y la seleccionada. Cuando un *Field* es un *SimpleField*, el *Slot* contiene el valor del *Field*. El valor de un *Slot* de un *SimpleField* y los *Slots* correspondientes a las opciones seleccionadas de los *SelectionFields* también se comportan como *Parameters* con el fin de ser pasado a otros *ViewElements* o *Action* cuando se dispara un *Event*. Los *ViewComponents Form* tienen *ValidationRules*, que determinan si el valor del *Field* es válido o no.

B. IFML: NOTACIÓN

A continuación se presenta la notación empleada por IFML para la construcción de los diagramas. Se muestran los símbolos empleados, su correspondiente uso y casos que ejemplifiquen su aplicación. Tanto la notación como sus descripción se basa en la información que aporta WebRatio como referencia para IFML[27].

B.1. VIEW CONTAINERS

B.1.1. View Container

Es un elemento de la interfaz que comprende elementos que muestran contenido y son soporte en la interacción y/o de otros View Containers. Puede representar páginas web, ventanas y paneles. En la Figura B.1 se presenta un ejemplo de un View Container.

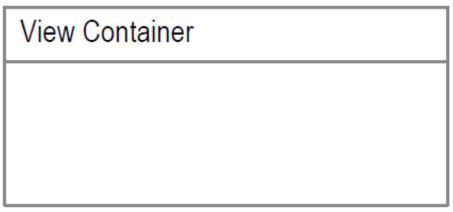

Figura B.1. View Container

B.1.2. XOR View Container

Un View Container que comprende View Containers secundarios que son mostrados de manera alternativa. Por ejemplo, Paneles con pestañas Java Frames en HTML. En la Figura B.2 se muestra este elemento.

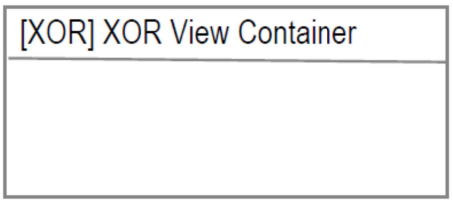

Figura B.2. XOR View Container

B.1.3. Landmark View Container

Es un View Container que es alcanzable desde cualquier otro elemento de la interfaz de usuario sin la necesidad que tenga flujos de interacción (Interaction Flows) de entrada de manera explicita, como ser el caso de un link a la página de inicio de un sitio web. En la Figura B.3 se muestra un caso.

27 http://www.webratio.com/learn/learningobject/ifml-quick-reference-card

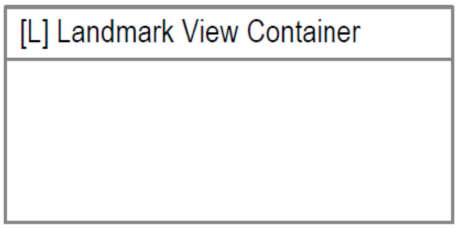

Figura B.3. Landmark View Container

B.1.4. Default View Container

Un View Container que será presentado de forma predeterminada al usuario cuando este acceda al contenedor que lo encierra, como en el caso de las páginas de bienvenida a un sitio web. En la Figura B.4 se representa el elemento mencionado.

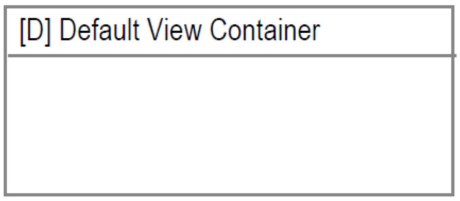

Figura B.4. Default View Container

B.2. VIEW COMPONENTS

B.2.1. View Component

Un elemento de la interfaz que muestra contenido al usuario o que acepta entradas del mismo. Estos elementos pueden ser una lista HTML, una galería de imágenes en JavaScript o un formulario. Como se representa el elemento se observa en la Figura B.5.

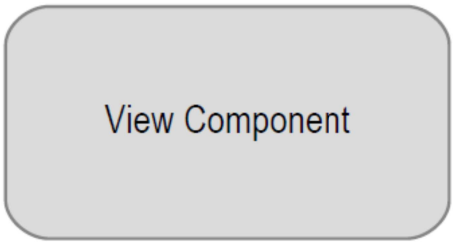

Figura B.5. View Component

B.2.2. View Component Part

Una parte de un View Component que no puede existir por cuenta propia. Puede accionar eventos y tener flujos de interacción (Interaction Flows) tanto de entrada como de salidas. Un View Component Part puede contener otras View Components Parts. Un representación genérica de estos puede verse en la Figura B.6, mientras que un ejemplo de campos en un formulario se puede apreciar en la Figura B.7.

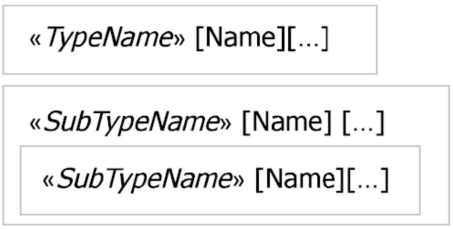

Figura B.6. View Component Part

Figura B.7. Campos en un formulario

B.3. EVENTS

B.3.1. Event

Un evento es un hecho que su ocurrencia afecta el estado de la aplicación, estos pueden ser del tipo Catching Event o Throwing Event, la notación utilizado para ambos se presenta en la Figura B.8.

La ocurrencia de Catching Event afecta el estado de la aplicación, causando el cambio de la navegación o el pasaje de parámetros entre elementos. Puede ser producido por la interacción con el usuario, cuando se termina la ejecución de una acción o por el sistema en forma de notificación. Es decir, ejemplos del mismo puede ser seleccionar un item de una lista o enviar un formulario.

Mientras que un Throwing Event es la ocurrencia de un evento generado por la aplicación modelada.

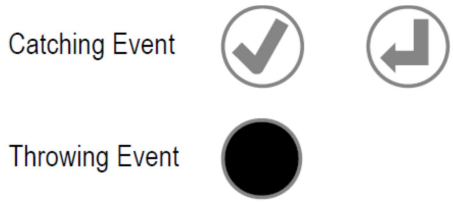

Figura B.8. Event

B.4. ACTIONS

B.4.1. Action

Es una porción de la lógica de negocio accionada por un evento; puede ser del lado del servidor (por defecto) o del lado del cliente, denominado como [Client]. En la Figura B.9 se visualiza su representación, que puede ser usada en el caso de que se quiera cargar una actualización en la base de datos, se quiera enviar un mail desde la aplicación o un verificador de ortografía integrado en el sistema.

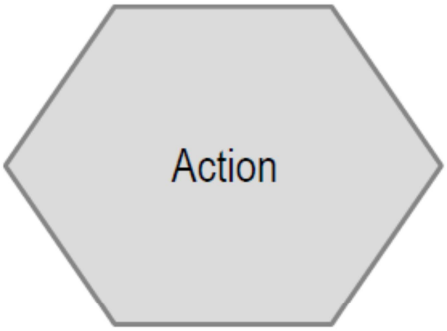

Figura B.9. Action

B.5. ACTIVATION EXPRESSIONS

B.5.1. Activation Expression

Es una expresión booleana asociada a un View element, View Component o Evento. En el caso de ser verdadero el elemento se habilita, puede ser en el caso que un usuario inserta contenido en un área de texto y de esta manera, el botón de envío se habilita. En la Figura B.10 se muestra la notación correspondiente.

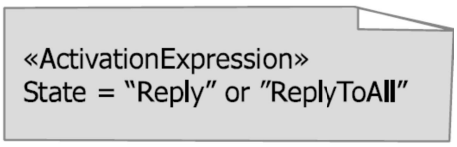

Figura B.10. Activation Expression

B.6. FLOWS

B.6.1. Navigation Flow

Una dependencia de entrada-salida. El origen del la asociación tiene alguna salida que está asociada con una entrada del destino de la asociación. Esto se da en los casos que se envía y se recibe parámetros en una petición HTTP. Su símbolo se muestra en la Figura B.11.

Figura B.11. Navigation Flow

B.6.2. Data Flow

Representa el pasaje de datos entre View Components o Actions como consecuencia de una interacción del usuario previa. Por ejemplo, en un carrito de compras, pasar la

información de la cantidad de dinero a cobrar al seleccionar la acción de pago. En la Figura B.12 se lo simboliza.

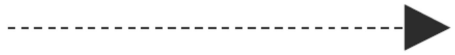

Figura B.12. Data Flow

B.7. PARAMETERS

B.7.1. Parameter

Representa un valor tipado y con un nombre establecido, pudiendo ser un texto correspondiente a los parámetros de una consulta HTTP o parámetros de una función o variables JavaScript. Su representación es opcional, pero de ser necesaria se lo realiza como se presenta en la Figura B.13.

Figura B.13. Parameter

B.7.2. Parameter Binding

Se utiliza para especificar que un parámetro de entrada del origen está asociado a un parámetro de salida del destino. Se utiliza, por ejemplo, en el caso que se quiera conectar el título de un álbum en un componente de entrada a un parámetro de otro componente. Su notación se presenta en la Figura B.14.

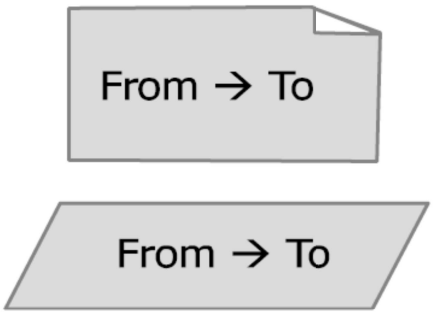

Figura B.14. Parameter Binding

B.7.3. Parameter Binding Group

Representa un conjunto de Parameter Bindings asociado a un flujo de interacción (Interaction Flow), por ende, se lo simboliza asociado a un Navigation Flow o Data Flow, según corresponda. Continuando con el ejemplo anterior, se utiliza para conectar el título del álbum y su año de lanzamiento, que son entrada de un componente, como parámetros de entrada de otro componente. Su notación se presenta en la Figura B.15.

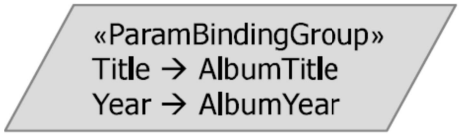

Figura B.15. Parameter Binding Group

B.8. MODULES

B.8.1. Module

Representa una porción de la interfaz del usuario y sus correspondientes acciones, las cuales pueden ser usadas para mejorar la mantenibilidad de los modelos IFML, como ser el caso de los procesos que se ejecutan para el pago en la compra de productos en línea. Su representación se presenta en la Figura B.16.

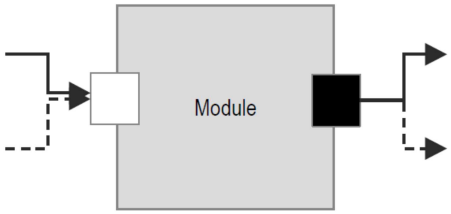

Figura B.16. Module

B.8.2. Input Port

Un Input Port representa un punto de interacción entre un Module y su entorno que recolecta los flujos de interacción y los parámetros que arriban al módulo. Siguiendo el ejemplo anterior, puede representar un punto de interacción para el proceso de ejecución del pago que recolecta la información de los productos. Sus representaciones se observan en la Figura B.17.

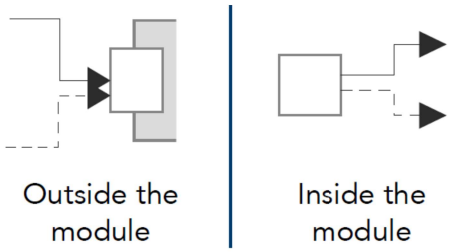

Figura B.17. Input Port

B.8.3. Output Port

Representa un punto de interacción entre un Module y su entorno que recolecta los flujos de interacción y parámetros que salen del módulo. Puede representar un punto de interacción que provee la ejecución de la confirmación del pago con un entorno externo (es decir, un sistema externo al modelo IFML). Su representación se aprecia en la Figura B.18.

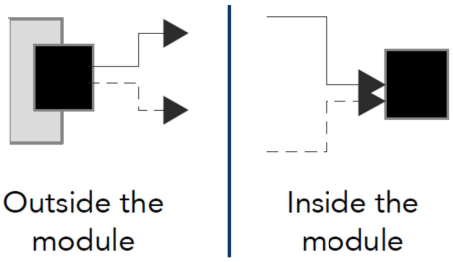

Figura B.18. Output Port

B.9. EJEMPLO: APLICACIÓN DE SERVICIO DE CORREO ELECTRÓNICO

B.9.1. Vista De Alto Nivel

Una representación alto nivel de la estructura de una interfaz para la aplicación de un servicio de emails se presenta en la Figura B.19.

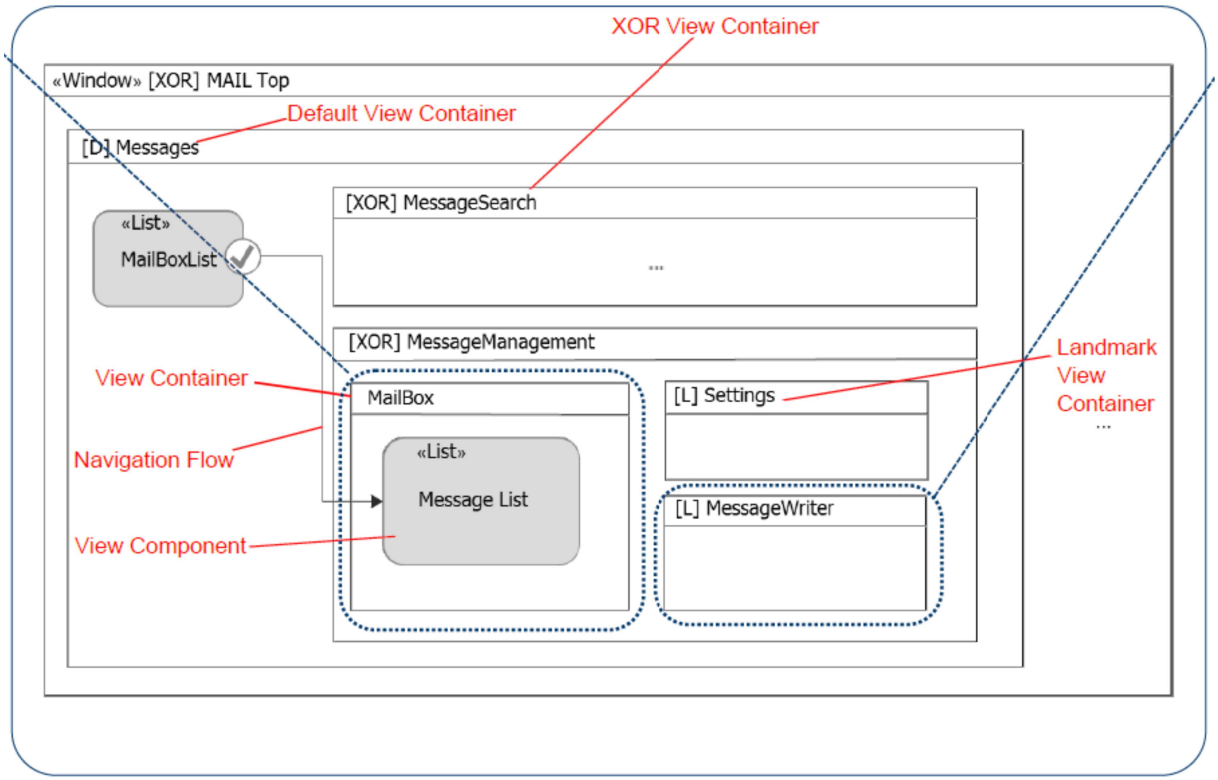

Figura B.19. Vista de Alto Nivel de la Aplicación.

En ella, el View Componente *MailBoxList* muestra el contenido de la casilla. Al seleccionar un email, el usuario puede ver el contenido del mensaje. De manera alternativa, el usuario puede escribir un nuevo mensaje (al seleccionar el contenedor *MessageWriter*) o puede editar la configuración *(Settings)*. Estas dos opciones están identificadas como Landmark ([L]) y son alcanzables desde el contenedor *MessageManagement*. Los contenedores identificados como XOR se muestran de manera alternativa. El contenedor *MessageSearch* puede tener varias interfaces alternativas para buscar los mensajes.

B.9.2. Vista De La Casilla De Emails

En la Figura B.20 se presenta una descripción de la página web que muestra el contenido de la casilla de correos.

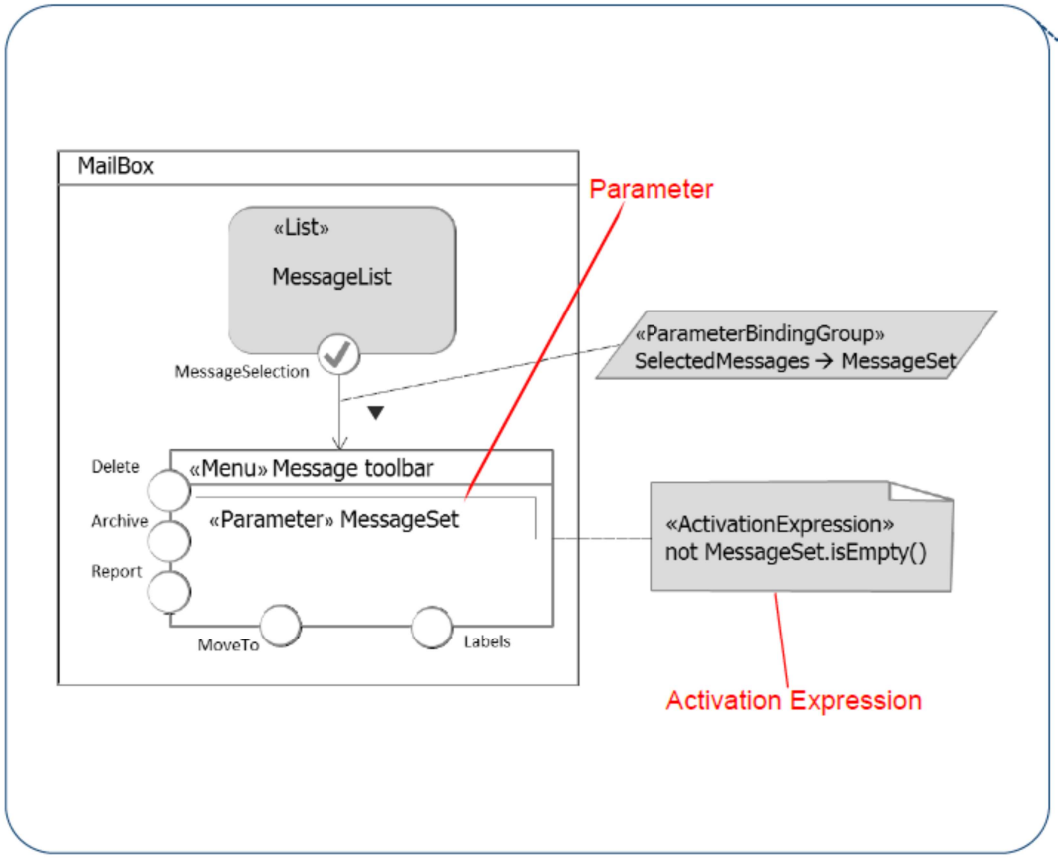

Figura B.20. Vista casilla de emails

El componente *MessageList* muestra la lista de mensajes recibidos y permite al usuario seleccionar los mensajes. Si uno o más mensajes son selecionados, el menú *Message toolbar* se muestra, consigo habilita varios eventos que pueden ser accionados, como pueden ser borrar (*Delete*), archivar (*Archive*), reportar (*Report*), mover (*Move*) y etiquetar (*Label*). Los mensajes seleccionados son pasados al contenedor como parámetros.

B.9.3. Vista De Envíos De Emails

Para esta ejemplificación, se presenta un modelo detallado de la página utilizada para el envío de emails en la Figura B.21.

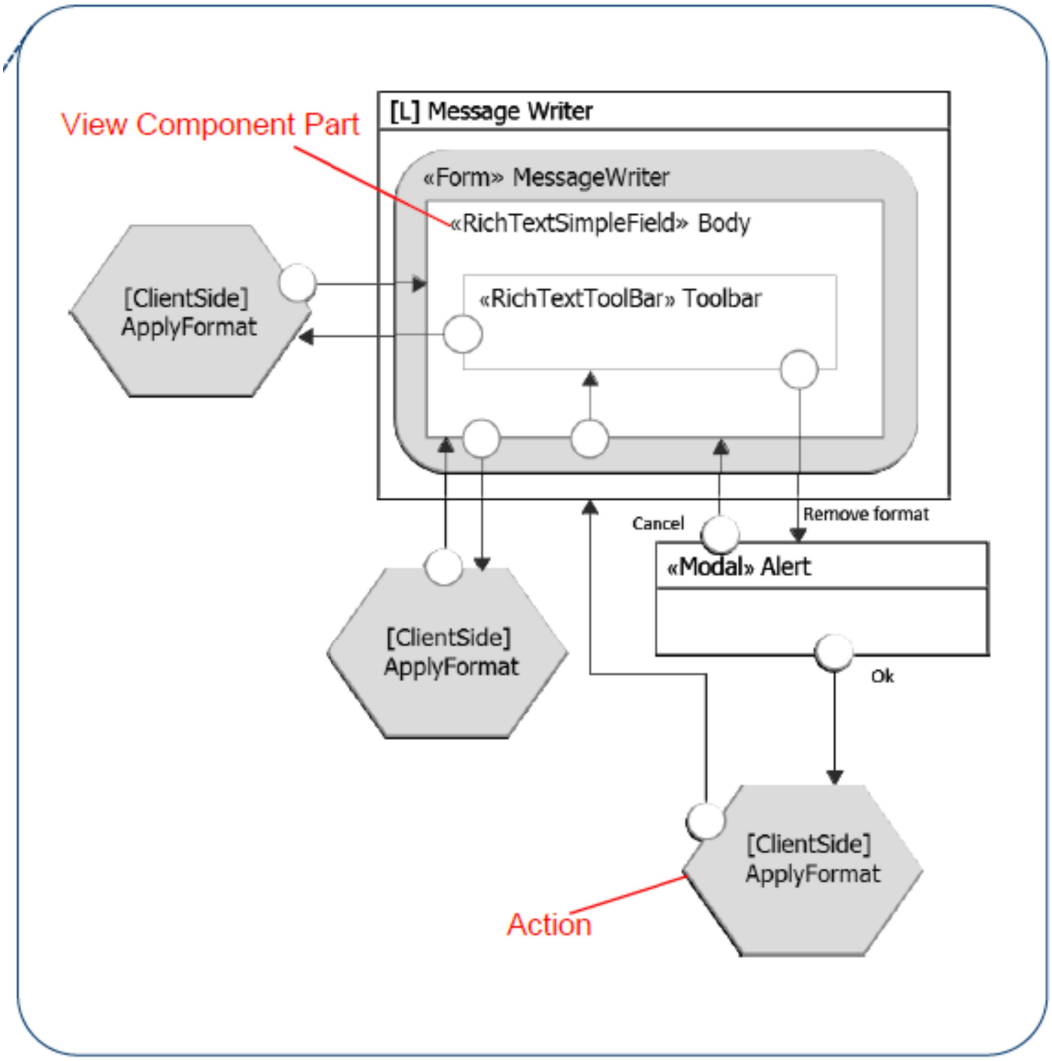

Figura B.21. Vista envíos de emails

El contenedor *MessageWriter* posee un formulario para el ingreso del mensaje, incluyendo un editor de texto enriquecido que permite al usuario aplicar formato desde el lado del cliente y removerlo. En caso de removerlo, un mensaje de confirmación (una ventana modal) se muestra antes de aplicar dicha acción.

C. METAMODELO: DOCUMENTACIÓN

En este anexo se realiza un análisis detallado del metamodelo propuesto, caracterizando cada paquete por el que está compuesto, con sus metaclases y enumeraciones, atributos y literales, relaciones, dependencias y restricciones de las mismas.

C.1. PAQUETE MVC

A continuación se presentan, en orden alfabético, las metaclases que se encuentran dentro del Paquete MVC.

C.1.1. Clase Application

Abstract: No.

Generalizaciones: No.

Especializaciones: No.

Descripción:

Una Application encapsula aquellos atributos que son propios de la aplicación. Se pueden tener varias aplicaciones compartiendo la misma instalación del *framework*. Los *frameworks* permiten que los atributos de esta clase sean globalmente accesibles.

Atributos:

- name : EString – El nombre que tendrá la aplicación. Las aplicaciones generadas por los *frameworks* suelen tener nombre predeterminados. Por ejemplo, en Laravel: “App”. En Yii2 y CodeIgniter: “app”. Algunos *frameworks*, como Laravel, usan el nombre de la aplicación para crear los espacios de nombre del sistema.
- *locale* : EString – Identificadores del lenguaje del usuario, su región y cualquier preferencia particular que el usuario quiera ver en su interfaz gráfica.

Asociaciones:

- aPackageController[1..*] : PackageController - Contenedor de todos los controladores de la aplicación.
- aPackageModel[1..*] : PackageModel – Contenedor de todos los modelos de la aplicación.
- aPackageView[1..*] : PackageView - Contenedor de todos las vistas de la aplicación.

C.1.2. Clase Attribute

Abstract: No.

Generalizaciones: No.

Especializaciones: Identifier.

Descripción:

Un Attribute representa tanto las propiedades de Modelos, Vistas y Controladores, como parámetros de los Métodos.

Atributos:

- name[1] : EString – El nombre asociado al atributo.

C.1.3. Clase Controller

Abstract: No.

Generalizaciones: MVCClass.

Especializaciones: No.

Descripción:

Un Controller agrupa la lógica de manejo de *requests* en clases. El nombre del controlador forma parte de una URI (*Universal Resource Identifier*) que permite tener acceso a ellos, directamente o a través de las rutas definidas en el *framework*.

En la Tabla C.1 se muestra el directorio predeterminado por cada *framework* para los controladores y las superclases que las componen.

Tabla C.1. Descripción de Controllers en los frameworks analizados.

Framework	Default Path	Controller SuperClass
Laravel	app/Http/Controllers	App\Http\Controllers\Controller
Yii2	app/Controllers	yii\web\Controller
CodeIgniter	application/controllers/	CI_Controller

Asociaciones:

- methods[0..*] : Method – Contenedor de los métodos asociados a la clase.

C.1.4. Clase Event

Abstract: No.

Generalizaciones: No.

Especializaciones: No.

Descripción:

Un Event es el resultado de un acción del navegador o del usuario.

Atributos:

- type : EventType – El tipo de evento asociado al mismo.

Asociaciones:

- handler[0..*] : Method – Vinculo con los métodos que manejan el evento en cuestión.

C.1.5. Clase Identifier

Abstract: No

Generalizaciones: Attribute.

Especializaciones: No.

Descripción:

Los Identifiers son aquellos atributos que serán claves primarias del modelo de datos.

Atributos:

- isAutoincremental : EBoolean – Si el valor que tendrá la clave será autoincremental para cada nuevo registro.

C.1.6. Clase Method

Abstract: No

Generalizaciones: No.

Especializaciones: No.

Descripción:

Los Methods son los procedimiento asociado al comportamiento de los objetos de la aplicación.

Atributos:

- name : EString – Nombre que tendrá el método.

Constraints:

- methodsBelongsToControllers
  self.name.endsWith(OurMM::MVC::Controller->select(controller|controller.name))

Asociaciones:

- inParameters[0..*] : Attribute – Contenedor de los atributos que recibe el método.
- outParameters[0..*] : Attribute – Contenedor de los atributos que retorna el método.

C.1.7. Clase Model

Abstract: No.

Generalizaciones: MVCClass

Especializaciones: No.

Descripción:

Los Models son las clases base de los modelos de datos. destinadas a trabajar con la información de la base de datos.

En la Tabla C.2 se presenta el directorio por defecto por cada *framework* para los modelos, las superclases que las componen y recomendaciones de estilo para su nomenclatura.

Tabla C.2. Descripción de Models en los frameworks analizados.

Framework	Default Path	Model SuperClass	Requerimientos de Estilo
Laravel	app/	Illuminate\Database\Eloquent\Model	Se recomienda que la primera letra del nombre del modelo esté en mayúscula.
Yii2	app/models/	yii\base\Model	
CodeIgniter	application/models/	CI_Model	

C.1.8. Clase MVCClass

Abstract: Sí.

Generalizaciones: No.

Especializaciones: Model, View, Controller.

Descripción:

Una MVCClass encapsula las propiedades y el comportamiento en común que puedan tener sendos componentes Model-View-Controller.

Atributos:

- name: EString – Nombre de la Clase.

Asociaciones:

- attributes[0..*] : Attribute – Tanto las clases de modelos, vistas y controladores pueden tener atributos.

C.1.9. Clase PackageController

Abstract: No.

Generalizaciones: No.

Especializaciones: No.

Descripción:

Un PackageControler reagrupa a todos los controladores. Esto se traduce, por ejemplo, en el nombre de una carpeta que los contenga.

Atributos:

- name : EString – Nombre del paquete al que pertenece un conjunto de controladores.

Asociaciones:

- controllers[0..*] : Controller – Representa el conjunto de controladores que pertenecen al paquete.

C.1.10. Clase PackageModel

Abstract: No.

Generalizaciones: No.

Especializaciones: No.

Descripción:

Un PackageModel reagrupa a todos los modelos. Esto se traduce, por ejemplo, en el nombre de una carpeta que los contenga.

Atributos:

- name : EString – Nombre del paquete al que pertenece un conjunto de modelos.

Asociaciones:

- models[0..*] : Model – Representa el conjunto de modelos que pertenecen al paquete.

C.1.11. Clase PackageView

Abstract: No.

Generalizaciones: No.

Especializaciones: No.

Descripción:

Un PackageView reagrupa a todas las vistas. Esto se traduce, por ejemplo, en el nombre de una carpeta que los contenga.

Atributos:

- name : EString – Nombre del paquete al que pertenece un conjunto de vistas.

Asociaciones:

- views[0..*] : View – Representa el conjunto de modelos que pertenecen al paquete.

C.1.12. Clase View

Abstract: No.

Generalizaciones: MVCClass.

Especializaciones: No.

Descripción:

Las Vistas son las responsables de presentar los datos a los usuarios finales. Por lo general son creadas en función a *view templates.* Nunca se cargan directamente sino que siempre son invocadas a través de un Controller.

Una vista:

- Podría leer las propiedades de un modelo pero no modificarlas.
- Podría estar construida en función a un determinado layout.
- Debería contener principalmente código presentacional, HTML.
- Debería evitar el acceso directo a los datos de una *request*, esto debería ser tarea del Controller.
- No debería contener consultas a la base de datos, este código debería estar en un modelo.

En la Tabla C.3 se presenta el directorio predeterminado por cada *framework* para las vistas.

Tabla C.3. Descripción de Views en los frameworks analizados.

Framework	Default Path
Laravel	app/resources/views
Yii2	app/views
CodeIgniter	Application/views

Constraints:

- innerViewsAreViews
  self.refImmediateComposite().oclIsKindOf(OurMM::MVC::View)

Asociaciones:

- innerViews[0..*] : View – Representa las vistas internas que pueden, a su vez, componer una vista.
- viewComponents[0..*] : ViewComponent – Los componentes visuales que componen una vista.

C.1.13. Clase ViewComponent

Abstracta: Sí.

Generalizaciones: No.

Especializaciones: MVC::HTMLElement.

Descripción:

Un ViewComponent es todo elemento que se renderiza como un componente visual en la interfaz gráfica. Esta clase, al igual que todas las demás de este paquete, no depende de ninguna tecnología en particular.

Un ViewComponent podría, por ejemplo, realizarse en un elemento HTML o un componente Java Swing.

Atributos:

- name : EString – Nombre del componente de la vista.

Asociaciones:

- events[0..*] : Event – Un componente visual puede o no tener eventos asociados.

C.1.14. Enumeración EventType

Descripción:

Se enumera aquí un pequeño subconjunto de los eventos definidos.

Literales:

- onError
- onLoad
- onSubmit

C.2. PAQUETE HTML

A continuación se presentan, en orden alfabético, las metaclases que se encuentran dentro del Paquete HTML.

C.2.1. Clase Anchor

Abstract: No.

Generalizaciones: HTMLElement

Especializaciones: No.

Descripción:

Los *Anchor* son porciones de texto que marcan el inicio o el final de un hiperenlace. Ayudan a mantener la organización de una página, permitiendo al usuario encontrar más rápidamente lo que necesita.

Atributos:

- content : EString – El texto visible del ancla que es en el navegador.
- href : EString – El atributo más importante
- target : TargetType – Especifica donde se abrirá el documento enlazado.

C.2.2. Clase Button

Abstract: No.

Generalizaciones: HTMLElement.

Especializaciones: No.

Descripción:

Esta clase define un botón. El comportamiento de dicho botón al ser activado, es controlado por el atributo *type*.

Atributos:

- content: EString - El texto del botón que es visible en el navegador
- isDisable : EBoolean – Default: false.
- type : ButtonType – Tipo de botón.

C.2.3. Clase CheckBox

Abstract: No.

Generalizaciones: Input

Especializaciones: No.

Descripción:

Es el componente gráfico HTML que permite al usuario hacer selecciones múltiples sobre un conjunto de opciones.

C.2.4. Clase Form

Abstract: No.

Generalizaciones: HTMLElement

Especializaciones: No.

Descripción:

Contenedor que agrupa una serie de datos para su envio a través de *requests*. Soporta algunos atributos específicos para configurar su comportamiento.

Atributos:

- action: EString – La dirección URL (*Uniform Resource Locator*) a la que se enviará el formulario.
- method: MethodType – El método HTTP usado para el envío del formulario.

- target: TargetType - Especifica donde se abrirá el documento indicado en el atributo action.

C.2.5. Clase HTMLElement

Abstract: Sí.

Generalizaciones: MVC::ViewComponent

Especializaciones: Anchor, Button, Form, Image, Input, Text.

Descripción:

Un HTMLElement es un componente individual del documento HTML que una vez parseado pasa a formar parte del DOM (*Document Object Model*).

Atributos:

- isEmpty: EBoolean – Default: false. Los tags vacíos no tienen contenido.
- isPairedTag: EBoolean – Default: true. Los tags pares tienSen etiqueta de apertura y cierre.
- tagName: HTMLTag – El nombre de la etiqueta HTML.

Asociaciones:

- htmlElements[0..*] : HTMLElement – Una etiqueta puede contener otras etiquetas dentro de su cuerpo.

C.2.6. Clase Image

Abstract: No.

Generalizaciones: HTMLelement

Especializaciones: No.

Descripción:

Define una imagen en un documento HTML.

Atributos:

- source : EString – URL de la imagen.

C.2.7. Clase Input

Abstract: Sí.

Generalizaciones: HTMLElement

Especializaciones: CheckBox, RadioButton y TextField.

Descripción:

Esta clase especifica los campos de entrada a través de los cuales los usuarios pueden ingresar datos.

Atributos:

- type: InputType – Enumeración que referencia el tipo de Input.
- value: EString – El valor que contendrá el input.

C.2.8. Clase RadioButton

Abstract: No.

Generalizaciones: Input

Especializaciones: No.

Descripción:

Elementos renderizados como pequeños iconos circulares que permiten realizar una selección única sobre un conjunto de opciones.

C.2.9. Clase Text

Abstract: No.

Generalizaciones: HTMLElement

Especializaciones: No.

Descripción:

Representa una porción de texto.

Atributos:

- content : EString – El texto en cuestión.
- language: EString – El lenguaje en el que está escrito el texto.

C.2.10. Clase TextField

Abstract: No.

Generalizaciones: Input.

Especializaciones: No.

Descripción:

Este tipo de Input renderiza en una caja de texto que permite a los usuarios ingresar texto en una sola línea.

C.2.11. Enumeración ButtonType

Descripción:

Definen el comportamiento del botón una vez que ha sido activado

Literales:

- button
- reset
- submit

C.2.12. Enumeración HTMLTag

Descripción:

Encapsula las etiquetas que son los bloques del lenguaje HTML.

Literales:

- a
- button
- form
- img
- text

C.2.13. Enumeración InputType

Descripción:

Especifica el tipo de datos y el control asociado a un elemento input dado.

Literales:

- checkbox
- radio
- text

C.2.14. Enumeración MethodType

Descripción:

Especifica el método HTTP con el cuál serán enviados los *request* de una página a otra.

Literales:

- delete
- get
- head
- patch
- post
- put

C.2.15. Enumeración TargetType

Descripción:

Especifica donde se abrirá un documento HTML.

Literales:

- blank
- framename
- parent
- self
- top

D. METAMODELO: PROPUESTA INICIAL

En este Anexo se introduce la primera propuesta que se realizó del metamodelo resultante en el presente trabajo. El mismo fue se presentó en el Informe Final de la Práctica Profesional Supervisada de la pasantía en Proyecto de Investigación y Desarrollo acerca de Ingeniería Web Dirigida por Modelos[28].

El metamodelo presentado en Figura X, además de tener las características que ya son parte del metamodelo desarrollado en este TFC facilita una serie de funciones adicionales que constituyen una base para aquellos interesados en extender la propuesta actual.

Las funciones adicionales que facilita este metamodelo pueden ser implementadas a través de metaclases. A manera de ejemplo, se listan a continuación las consideradas más relevantes:

- Hookeable: que permitiría la generación de código a ejecutar antes y/o después de la ejecución de determinados métodos.
- AppType: para permitiría especificar que tipo de aplicación se generará, entre APIs, aplicaciones de consola o Web.
- EntryScript: para permitiría determinar a su vez que ambiente destino tendrá la generación de código, entre desarrollo, testing o producción.

28 *Ref. Expte. N°020/17-PPS* – Universidad Gastón Dachary.
Pasantía realizada en el contexto del Proyecto de I+D adjudicado y registrado por la Sec. de Investigación y Desarrollo de UGD –Código / N° 11 en Res. de Rectorado UGD N° 18/A/14; Área/s Temática/s: Tecnologías de la Información; Ingeniería de Requerimientos; Ingeniería Web– *"Estrategias basada en ontologías para reducir el gap semántico entre modelos CIM en el marco de MDA"*. Directora: Dra. María Laura Caliusco (UTN FR Santa Fe); Docentes-Investigadores (UGD): Ing. Héctor Ruidias e Ing. Edgardo A. Belloni.

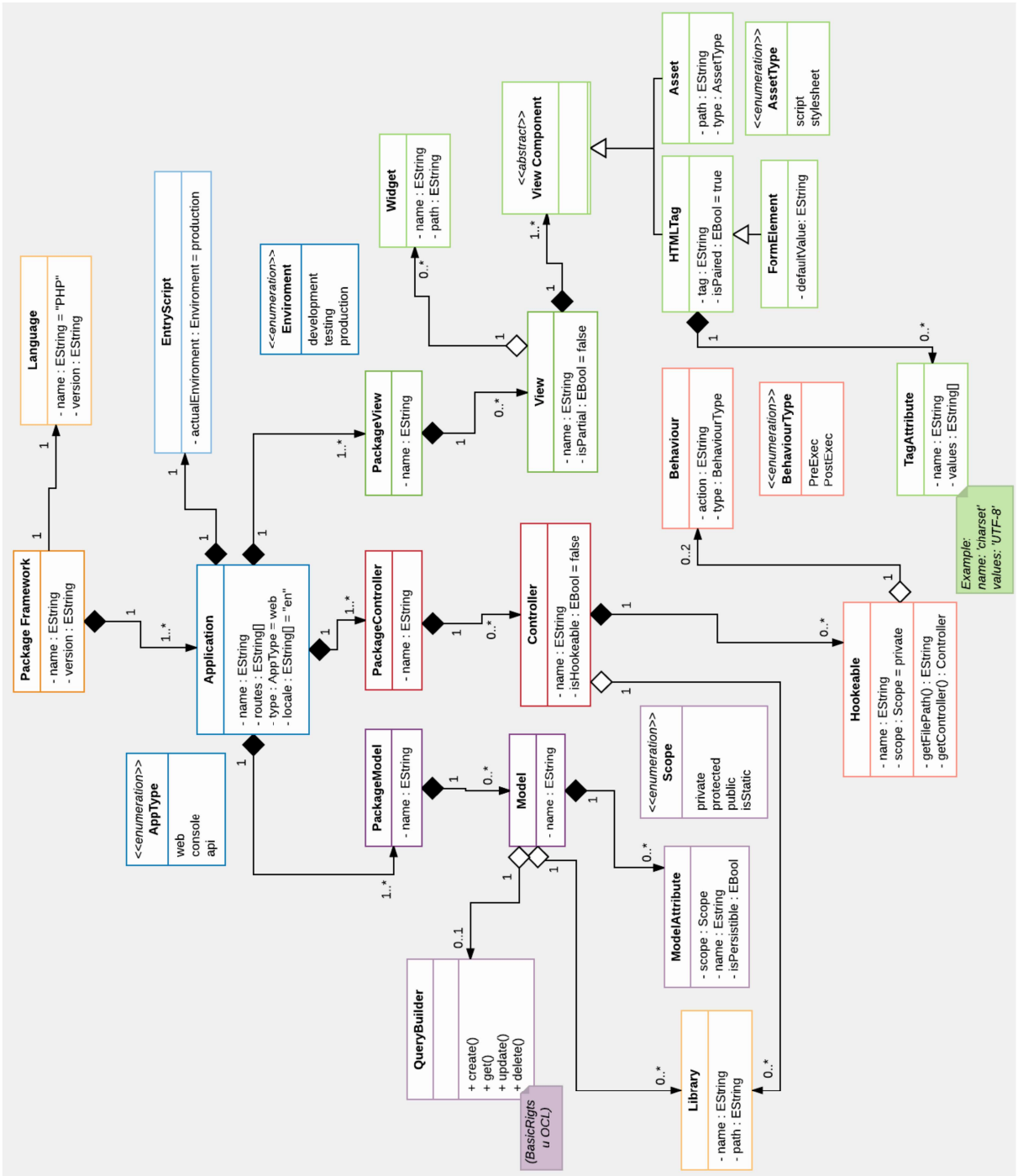

Figura D.1. Propuesta inicial del Metamodelo

E. CONFIGURACIÓN: ENTORNO DE DESARROLLO

En este anexo se detallan los requerimientos con que se debe contar para ejecutar las herramientas de transformación utilizadas para la realización del trabajo, como así también de las aplicaciones finales resultantes. También se agregan una serie de problemas comunes que se afrontan al configurar los programas y sus soluciones.

E.1. REQUERIMIENTOS

- Eclipse Luna Modeling Tools[29]
- Composer[30]
- Apache Commons Lang[31]
- Laravel Framework[32]
- Laravel Colletive[33]
- Yii2 Framework[34]

E.1.1. Eclipse

Los *frameworks* ATL[35] y Acceleo[36] están diseñados para la realización de transformaciones M2T y M2M, respectivamente y son las herramientas que usaremos a lo largo de este proyecto.

El primer paso es descargar e instalar Eclipse Modeling Tools del link provisto en la nota al pie.

Una vez que Eclipse está listo, necesitamos instalar los *frameworks* ATL y Acceleo. Podemos hacer esto, siguiendo los siguientes pasos[37]:

1. Help → Installing Modelling Components.
2. Elegir Acceleo y ATL.
3. Reiniciar Eclipse una vez que los *frameworks* ATL y Acceleo se hayan instalado correctamente.

Una vez que Eclipse inició nuevamente necesitamos instalar IFML Editor[38], podemos hacerlo siguiendo las instrucciones que se encuentran en el repositorio de la herramienta[39].

29 https://www.eclipse.org/downloads/packages/eclipse-modeling-tools/lunasr2
30 https://getcomposer.org/download/
31 https://commons.apache.org/proper/commons-lang/
32 https://laravel.com/
33 https://laravelcollective.com/docs/5.0/html
34 http://www.yiiframework.com/
35 https://eclipse.org/atl/
36 https://www.eclipse.org/acceleo/
37 https://github.com/Dipiert/ifml2php/blob/master/images/install_atl-acceleo.gif
38 https://github.com/ifml/ifml-editor
39 http://ifml.github.io/

E.1.2. Composer

Composer es un gestor de dependencias para PHP que nos va a permitir instalar tanto los *frameworks* Laravel como Yii2.

Instalación

Para Windows, la mejor manera de hacerlo es descargando y ejecutando su instalador[40].

Para distribuciones Linux basadas en Ubunto el siguiendo comando debería bastar:

```
sudo apt install composer
```

En caso de que eso no haya funcionado, los siguientes comandos dejarán el trabajo hecho:

```
php -r "copy('https://getcomposer.org/installer', 'composer-
setup.php');"
php -r "if (hash_file('SHA384', 'composer-setup.php') ===
'544e09ee996cdf60ece3804abc52599c22b1f40f4323403c44d44fdfdd586475ca98
13a858088ffbc1f233e9b180f061') { echo 'Installer verified'; } else
{ echo 'Installer corrupt'; unlink('composer-setup.php'); } echo
PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

E.2. PROYECTOS

La convención de nombres para todos los proyectos es:

<dominio>.<nombre-proyecto>.<tipo-entrada>.<nombre-metamodelo-entrada>.gen.<nombre-lenguaje-salida>

Donde tipo-entrada puede ser:

- pim: Platform-Independent Model
- psm: Platform-Specific Models
- dsl: Domain-Specific Language

E.3. ATL

Antes que nada, usted debería importar el proyecto ATL en Eclipse:

1. File→ Import (Alt + F + I)
2. General
3. Existing project into workspace

Nuestro proyecto ATL se llama edu.ifml2php.pim.ifml.gen.lycmm, donde *lycmm* significa “Laravel, Yii2 & CodeIgniter Meta-Model”.

E.3.1. Proyecto ATL

El proyecto ATL consiste en:

- Un archivo llamado ifml2OurMM.atl, donde están escritas las reglas de transformación Modelo a Modelo.

- Una carpeta llamada atlLibraries, donde están los helpers de IFML.

40 https://getcomposer.org/Composer-Setup.exe

El archivo ifml2OurMM.launch en la raíz del proyecto ATL contiene los parámetros de configuración de la ejecución. No debería ser necesario cambiar este archivo, pero en caso de que quieras saber los detalles sobre él, puedes leerlos en la sección **Creando un archivo .launch** de este anexo.

Finalmente, un click derecho en el archivo ifml2OurMM.atl, entonces “Run As” → “ATL transformation” debería ejecutar por fin, la transformación.

E.3.2. Creando Un Archivo .launch

Haz click derecho en el archivo ifml2OurMM.atl, entonces “Run As” → “Run Configuration”. Esto abre una ventana donde puedes configurar los siguientes parámetros:

Se incentiva, siempre que sea posible, el uso de URIs en lugar de paths absolutos o relativos.

E.3.3. Módulo ATL

Debería apuntar al path del archivo ifml2OurMM.atl sobre el cuál acabas de hacer click.

E.3.4. Metamodelos

En la Tabla E.1 se presentan los metamodelos utilizados con su correspondiente directorio y la URI del mismo.

Tabla E.1. Configuración Metamodelos de Entrada - ATL

Nombre	Path	URI
UMLMM	-	uri:http://www.eclipse.org/uml2/5.0.0/UML
IFMLMM	/models/metamodels/IFML-Metamodel.ecore	uri:http://www.omg.org/spec/20130218/core
extMM	/models/metamodels/IFML-Metamodel.ecore	uri:http://www.omg.org/spec/20130218/ext
ourMM	/edu.ifml2php.pim.ifml.gen.lycmm/models/metamodels/Metamodel.ecore	uri:http://www.application.org

Es importante aclarar que al configurarlos no se debe marcar el checkbox *“Is Metametamodel”*.

E.3.5. Modelos De Origen

En la Tabla E.2 se presentan los modelos que son entradas en las transformaciones generadas con ATL.

Tabla E.2. Modelos de Origen - ATL

Nombre	Conforma a	Path*	Tipo
uml	UMLMM	model.uml	Domain
ifml	IFMLMM	movies.core	Interaction Flow

Nombre	Conforma a	Path*	Tipo
extm	extMM	movies.core	

*/edu.ifml2php.pim.ifml.gen.lycmm/models/models/

E.3.6. Modelos Destino

Para este proyecto, **ourm** es el path en donde se guardará el archivo .xmi generado. En este caso, la raíz del proyecto.

E.3.7. Librerías

En la Tabla E.3 se presentan las librerías ATL con que cuenta este proyecto y sus correspondiente ubicación en el directorio.

Tabla E.3. Librerías - ATL

Nombre	Path*
ifmlCoreLibrary	ifmlCoreLibrary.asm
ifmlExtLibrary	/ifmlExtLibrary.asm
mvcLibrary	/mvcLibrary.asm
systemLibrary	/atlLibraries

*/edu.ifml2php.pim.ifml.gen.lycmm/atlLibraries/

E.4. ACCELEO

Con el fin de usar los modelos en un proyecto Acceleo es necesario incluir Metamodel.ecore en el registro de EMF.

La manera más fácil de hacer esto es[41]:

1. Cambiar la perspectiva de Eclipse a ATL Perspective: Window→ Open Perspective → Other → ATL
2. Click derecho en Metamodel.ecore → Register Metamodel

No se verá ningún cambio visual en el IDE, pero ahora el metamodelo está registrado.

E.4.1. Proyecto Acceleo

Ahora se debe importar el proyecto ATL en Eclipse:

1.File → Import (Alt + F + I)
2. General
3. Existing project into workspace

En en el caso de este proyecto, se encuentran 2 proyectos Acceleo, llamados

edu.ifml2php.psm.lycmm.gen.laravel
edu.ifml2php.psm.lycmm.gen.yii2

41 https://github.com/Dipiert/ifml2php/blob/master/images/register_metamodel.gif

E.5. SOLUCIÓN DE PROBLEMAS (TROUBLESHOOTING)

E.5.1. ATL

En la Tabla E.4 se especifican una serie de problemas comunes en ATL y la solución a los mismos.

Tabla E.4: Descripción Problema-Solución ATL.

Problema	Solución
You must specify a path for <model-name>	Verificar los parámetros de la transformación M2M en el archivo .launch
Arguments of a generation cannot be null	
{.asm, .atl} does not existe	

E.5.2. Acceleo

- Problema: Package with URI <uri> not found:

  Solución: haz un registro manual del metamodelo colocando el siguiente código en los métodos "registerPackages" del Java Launcher asociado al main de la transformación que se quiere realizar. Puede usar el siguiente código:

```
URI uri =
URI.createFileURI("../edu.ifml2php.pim.ifml.gen.lycmm/models/metamodels/
Metamodel.ecore");
Resource resource = resourceSet.getResource(uri, true);
EPackage PHPMVC = (EPackage) resource.getContents().get(0);
List<EPackage> subPackages = PHPMVC.getESubpackages();
  for(EPackage subPack : subPackages){
    EPackage.Registry.INSTANCE.put(subPack.getNsURI(), subPack);
  }
```

- Problema: ClassNotFoundException: org.eclipse.uml2.types.TypesPackage

  Solución: Agrega org.eclipse.uml2.types[42] al classpath.

- Problema: Could not find public template <nombre-template> en el modulo <nombre-modulo>.

  Solución: Asegurate de tener al menos un template con la anotación @main y que sea público.

- Problema: java.lang.NoClassDefFoundError: org/eclipse/core/resources/IresourceChangeListener.

  Solución: Agregar org.eclipse.core.resources[43] al classpath

42 http://central.maven.org/maven2/org/eclipse/uml2/types/2.0.0-v20140602-0749/types-2.0.0-v20140602-0749.jar

43 http://central.maven.org/maven2/org/eclipse/birt/runtime/org.eclipse.core.resources/3.9.1.v20140825-1431/org.eclipse.core.resources-3.9.1.v20140825-1431.jar

F. ATL: CÓDIGO FUENTE

En el presente anexo se adjunta el código fuente ATL, utilizado para las transformaciones modelo a modelo (M2M). Se adjuntaran el código del archivo principal con todas las reglas y de las librerías que sirven de soporte a las transformaciones.

F.1. MODULO PRINCIPAL

Este módulo es a partir del cual se generan las transformaciones, contiene todas las reglas y las relaciones entre el metamodelo IFML de entrada y el metamodelo propuesto en el este trabajo, que conforma el resultado de las transformaciones. Este archivo es llamado ifml2OurMM.atl

```
-- @nsURI ourMM=http://www.application.org
-- @nsURI UMLMM=http://www.eclipse.org/uml2/5.0.0/UML
-- @nsURI IFMLMM=http://www.omg.org/spec/20130218/core
-- @nsURI extMM=http://www.omg.org/spec/20130218/ext

module tinyRule;
create ourm : ourMM from uml : UMLMM, ifml : IFMLMM, extm : extMM;

uses ifmlCoreLibrary; uses ifmlExtLibrary; uses mvcLibrary; uses
systemLibrary;


rule IFMLModel2ApplicationClass{
  from
    ifmlModel : IFMLMM!IFMLModel
  to
    app : ourMM!Application(
      name <- ifmlModel.name,
      aPackageModel <- ifmlModel.domainModel,
      aPackageView <- ifmlModel.interactionFlowModel,
      aPackageController <-
thisModule.DomainModel2PackageController(ifmlModel.domainModel)
  )
}

lazy rule DomainModel2PackageController{
  from
    domainModel : IFMLMM!DomainModel
  to
    pc  : ourMM!PackageController(
      name <- domainModel.name,
      controllers <- UMLMM!Class.allInstances()
                     ->collect(r|thisModule.UMLClass2Controller(r))
   )
}

lazy rule UMLClass2Controller{
  from
    class : UMLMM!Class
  to
    cont : ourMM!Controller(
      name <- class.name,
```

```
      methods <- IFMLMM!IFMLAction.allInstances()
                ->select(b|
b.name.toLower().endsWith(class.name.toLower())
     ))
}

rule IFMLAction2Method{
  from
    ifmla : IFMLMM!IFMLAction
  to
    a : ourMM!Method(
      name <- ifmla.name.decapitalize(),
      inParameters <- ifmla.getInParameters(),
      outParameters <- ifmla.getOutParameters()
    )
}

lazy rule Parameter2Attribute{
  from
    st : String
  to
    at : ourMM!Attribute(
      name <- st.toLower()
    )
}

rule DomainModel2PackageModel{
  from
    dm : IFMLMM!DomainModel
  to
    pm : ourMM!PackageModel(
      name <- dm.name,
      models <- UMLMM!Class.allInstances()
    )
}

rule UMLClass2Model{
  from
    class : UMLMM!Class
  to
    model : ourMM!Model(
      name<-class.name,
      attributes <- UMLMM!Property.allInstances()
                   ->select(e | e.namespace.name = class.name),
      primaryKeys <- UMLMM!Property.allInstances()
                    ->select(e | e.namespace.name = class.name)
    )
}

rule UMLProperty2Attribute{
  from
    prop : UMLMM!Property
   (not prop.isID)
  to
    modAt : ourMM!Attribute(
      name <- prop.name
    )
```

```
}

rule UMLProperty2Identifier{
  from
    prop : UMLMM!Property
   (prop.isID)
  to
    modAt : ourMM!Identifier(
      name<-prop.name
    )
}

rule InteractionFlowModel2PackageView{
  from
    ifm : IFMLMM!InteractionFlowModel
  to
    pv : ourMM!PackageView(
      name <- ifm.name,
      views <- ifm.interactionFlowModelElements
                ->select(f|f.oclIsKindOf(IFMLMM!ViewContainer))
    )
}

rule ViewContainer2View{
  from
    vc : IFMLMM!ViewContainer
    (not (vc.refImmediateComposite().oclIsKindOf(IFMLMM!
ViewContainer)))
  to
    v  : ourMM!View(
      name <- vc.name,
      viewComponents <- vc.getViewComponents(),
      innerViews <- vc.viewElements->select(e| e.oclIsKindOf(IFMLMM!
ViewContainer))->collect(e|thisModule.ViewContainer2InnerView(e))
    )
}

lazy rule ViewContainer2InnerView{
  from
    vc : IFMLMM!ViewContainer
    (vc.refImmediateComposite().oclIsKindOf(IFMLMM!ViewContainer))
  to
    v  : ourMM!View(
      name <- vc.name,
      viewComponents <- vc.getViewComponents(),
      innerViews <- vc.viewElements->select(e| e.oclIsKindOf(IFMLMM!
ViewContainer))->collect(e|thisModule.ViewContainer2InnerView(e))
    )
}

rule ConditionalExpression2Text{
  from
    ce : IFMLMM!ConditionalExpression
  to
    t : ourMM!Text(
      tagName <- #p,
      name <- ce.name,
```

```
      language <- ce.language,
      content <- ce.body
    )
}

rule IFMLForm2Form{
  from
    frm : extMM!Form
  to
    f : ourMM!Form(
      name <- frm.name,
      tagName <- #form,
      method <- #post,
      target <- f.getDefaultTarget(),
      htmlElements <- frm.getInputFields(),
      events <- thisModule.OnSubmitEvent2Event(f.name)
    )
}

lazy rule IFMLSlot2RadioButton{
  from
    sl : extMM!IFMLSlot,
    st : String
  to
    rb : ourMM!RadioButton(
      name <- st,
      value <- sl.name,
      tagName <- #input,
      type <- #radio,
      isPairedTag <- false
    )
}

lazy rule IFMLSlot2Checkbox{
  from
    sl : extMM!IFMLSlot,
    st : String
  to
    cb : ourMM!Checkbox(
      name <- st,
      value <- sl.name,
      tagName <- #input,
      type <- #checkbox,
      isPairedTag <- false
    )
}

rule SimpleField2TextField{
  from
    sf : extMM!SimpleField
  to
    f : ourMM!TextField(
      name <- sf.name.decapitalize(),
      tagName <- #input,
      type <- #text,
      isPairedTag <- false
    )
```

```
}

lazy rule OnSubmitEvent2Event{
  from
    n : String
  to
    e : ourMM!Event(
      handler <- e.getHandler(n),
      type <- e.getDefaultEventType()
    )
}

lazy rule Landmarks2MenuItems{
  from
    vc  : IFMLMM!ViewContainer((vc.isLandmark))
  to
    an  : ourMM!Anchor(
      name <- vc.name.decapitalize(),
      hypRef <- vc.name,
      target <- an.getDefaultTarget(),
      tagName <- #a
    )
}

lazy rule ViewElement2Image{
  from
    ve : IFMLMM!ViewElement
  to
    i : ourMM!Image(
      source <- ve.id,
      name <- ve.name,
      tagName <- #img
    )
}
```

F.2. LIBRERÍAS

F.2.1. IFML Core Library

```
library ifmlLibrary;

helper context IFMLMM!ViewContainer def : getAnchors() :
Sequence(ourMM!Anchor) = IFMLMM!ViewContainer.allInstances()
          ->select(a | (a.refImmediateComposite() = IFMLMM!
InteractionFlowModel.allInstances().first()) and (a <> self))
          ->collect(f|thisModule.Landmarks2MenuItems(f));

helper context IFMLMM!IFMLAction def : getInParameters() :
Sequence(ourMM!Attribute) = self.parameters->select(b|b.direction =
#"in" or b.direction = #inout)->collect(b|
thisModule.Parameter2Attribute(b.name));

helper context IFMLMM!IFMLAction def : getOutParameters() :
Sequence(ourMM!Attribute) = self.parameters->select(b|b.direction =
#out or b.direction = #inout)->collect(b|
thisModule.parameters2Attributes(b.name));
```

```
helper context IFMLMM!ViewElement def : getInputFields() :
Sequence(ourMM!Input) =
Sequence{self.getSimpleFields(),self.getSelectionFields()};

helper context IFMLMM!ViewContainer def : getViewComponents() :
Sequence(ourMM!ViewComponent) =
   Sequence{
            if (not (self.refImmediateComposite().oclIsKindOf(IFMLMM!
ViewContainer))) then
                     self.getAnchors()
            else
                     Sequence{}
            endif,
            self.getInputFields(),
            self.getConditionedViewComponentsParts(),
            self.getImages(),
            self.getForms()
   };

helper context IFMLMM!ViewContainer def :
getConditionedViewComponentsParts() : Sequence(ourMM!ViewComponent) =
Sequence{ self.viewElements->select(f | f.oclIsTypeOf(IFMLMM!
ViewComponent))->collect(h|h.viewComponentParts->select(b|
b.oclIsTypeOf(IFMLMM!ConditionalExpression)))};

helper context IFMLMM!ViewContainer def : getImages() :
Sequence(ourMM!Image) = Sequence {self.viewElements->select(e |
e.oclIsTypeOf(IFMLMM!ViewElement))->collect(f|
thisModule.ViewElement2Image(f))};

helper context IFMLMM!ViewContainer def : getForms() :
Sequence(ourMM!Form) = Sequence { self.viewElements->select(e |
e.oclIsTypeOf(extMM!Form))};

helper context IFMLMM!ViewElement def : getSelectionFields() :
Sequence(ourMM!Input) =
   Sequence{extMM!IFMLSlot.allInstances()->select(e|self.parameters-
>includes(e.refImmediateComposite()) and
not(e.refImmediateComposite().isCheckbox()))
                                       ->collect(e|
thisModule.IFMLSlot2RadioButton(e, e.refImmediateComposite().name)),
                     extMM!IFMLSlot.allInstances()
                                       ->select(e|self.parameters-
>includes(e.refImmediateComposite()) and
e.refImmediateComposite().isCheckbox())
                                       ->collect(e|
thisModule.IFMLSlot2Checkbox(e, e.refImmediateComposite().name))};
```

F.2.2. IFML Extension Library

```
library ifmlExtLibrary;

helper context extMM!Field def : isCheckbox() : Boolean =
   self.isMultiSelection;

helper context extMM!IFMLWindow def : getSimpleFields() :
Sequence(ourMM!TextField) = self.viewElements->select(e |
e.oclIsTypeOf(extMM!SimpleField));

helper context extMM!Form def : getSimpleFields() : Sequence(ourMM!
TextField) = self.viewComponentParts->select(e | e.oclIsKindOf(extMM!
SimpleField));
```

F.2.3. MVC Library

```
library mvcLibrary;

helper context ourMM!Event def : getDefaultEventType() : ourMM!
EventType  = #onSubmit;

helper context ourMM!Anchor def : getDefaultTarget() : ourMM!
TargetType = #self;

helper context ourMM!Form def : getDefaultTarget() : ourMM!TargetType
= #self;
helper context ourMM!Event def : getHandler (n : String) : String =
extMM!OnSubmitEvent.allInstances()  ->select(d |
d.viewElement.name.toLower() = n.toLower())->collect(a |
a.outInteractionFlows->collect(f | f.targetInteractionFlowElement)-
>collect(g | g.name.decapitalize()));
```

F.2.4. System Library

```
library systemLibrary;

helper context String def : decapitalize() : String =
self.substring(1,1).toLower() + self.substring(2,self.size());
```

G. CASOS DE PRUEBA

A continuación se presentan las descripciones textuales de los casos de prueba realizados sobre las páginas web resultantes de la transformación modelo a texto. Los mismos se dividen en *tests* realizados a la página web como a los formularios que éstas contienen.

G.1. TESTS PARA LA PÁGINA WEB

Para las pruebas realizadas para la página web, en la Tabla G.1 se observa que se prueba que las páginas tengan el título correcto, mientras que en la Tabla G.2 y Tabla G.3 se realiza la prueba que contenga el botón para enviar los formularios las páginas que así lo requieran y que contenga las anclas, respectivamente. Finalmente, en la Tabla G.4 se presenta la prueba que muestren imágenes las páginas que así lo deberían.

Tabla G.1. Caso de Prueba: Las páginas deben tener el título correcto.

Nombre	Las páginas deben tener el título correcto		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	Se conoce la URL de todas las vistas y, de los modelos, se conoce el título de cada una de ellas.			
Pasos	1	Por cada vista:		
	2	El título de la vista coincide con su nombre.		
Estado	Ejecutado con éxito			

Tabla G.2. Caso de Prueba: Los forms deben tener el botón que permita su envío.

Nombre	Los forms deben tener el botón que permita su envío		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	- Se conoce la URL las vistas que contienen formularios y, de los modelos, se conoce el nombre de cada uno de estos formularios.			
Pasos	1	Por cada vista que contiene formularios:		
	2	Se debe verificar que existe un form que contiene el nombre de uno de los IFML Forms y con elementos HTML hijos de etiqueta “input” y tipo “submit”.		
Estado	Ejecutado con éxito			

Tabla G.3. Caso de Prueba: Las páginas deben tener anchors.

Nombre	Las páginas deben tener Anchors		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	Se conoce la URL las vistas que contienen Anchors y, de los modelos, se conoce el título de los anchors que deben estar en esa vista.			
Pasos	1	Por cada vista que contiene Anchors:		
	2	Por cada uno de los Anchors que debe tener esa página:		
	3	Se verifica que el título del enlace, sin espacios, coincide con el título de alguna de las vistas.		
Estado	Ejecutado con éxito			

Tabla G.4. Caso de Prueba: Algunas páginas deben tener imágenes.

Nombre	Algunas páginas deben tener imágenes		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	Se conoce la URL las vistas que contienen imágenes y, de los modelos, se conoce el título de las imágenes que deben estar en esa vista.			
Pasos	1	Por cada vista que contiene imágenes:		
	2	Por cada una de las imágenes:		
	3	Se verifica que el texto alternativo de la imagen coincide con el nombre de algún IFML ViewElement.		
Estado	Ejecutado con éxito			

G.2. TESTS PARA LOS FORMULARIOS

En cuanto a las pruebas realizadas sobre los formularios, en la Tabla G.5 se presenta el caso de prueba sobre aquellos que contengan entradas de texto. Mientras que en la Tabla G.6 y Tabla G.7, se detallan las pruebas sobre los *labels* del formulario y los botones de radio de los que así lo requieran, respectivamente.

Tabla G.5. Caso de Prueba: Algunos forms deben tener textinputs

Nombre	Algunos forms deben tener TextInputs		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	Se conoce la URL de la vista que contiene el Form y, de los modelos, se conoce el nombre de los Inputs cuya existencia se debe comprobar.			
Pasos	1	Por cada vista que contiene un form:		
	2	Por cada input que debe tener dicha vista:		
	3	Deben existir inputs que sean de tipo “text” y cuyo nombre coincida con el nombre de un IFML SimpleField.		
Estado	Ejecutado con éxito			

Tabla G.6. Caso de Prueba: Algunos forms deben tener labels.

Nombre	Algunos forms deben tener Labels		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	Se conoce la URL de la vista que contiene el Form y, de los modelos, se conoce el nombre de los Labels cuya existencia se debe comprobar.			
Pasos	1	Por cada vista que contiene un form:		
	2	Por cada input que debe tener dicha vista:		
	3	Deben existir labels cuyo nombre coincida con el nombre de un IFML SimpleField.		
Estado	Ejecutado con éxito			

Tabla G.7. Caso de Prueba: Algunos forms deben tener radiobuttons.

Nombre	Algunos forms deben tener RadioButtons		**Versión**	1.0
Entorno	Sistema Operativo	Ubuntu Xenial		
	Arq. Procesador	x86_64		
	PHP	PHP 7.0.22		
	MySQL	Ver 14.14 Distrib 5.7.20		
	Navegador	Mozilla Firefox 57.0.4 Opera 51.0.2830 Chrome 64.0.3282		
Precondiciones	Se conoce la URL de la vista que contiene el Form y, de los modelos, se conoce el nombre de los botones de radio cuya existencia se debe comprobar.			
Pasos	1	Por cada vista que contiene un form:		
	2	Por cada input que debe tener dicha vista:		
	3	Deben existir inputs de tipo “radio” cuyo nombre coincida con el nombre de un IFML Slot.		
Estado	Ejecutado con éxito			