





# Desarrollo de aplicaciones móviles mediante ingeniería dirigida por modelos

Caso de estudio basado en Android



# Desarrollo de aplicaciones móviles mediante ingeniería dirigida por modelos

Caso de estudio basado en Android

Héctor Arturo Flórez Fernández,  
Daniel Esteban Sánchez Rodríguez





# Resumen

La industria del *software* debe satisfacer el cumplimiento de atributos de calidad, como mantenibilidad, extensibilidad, entre otros. El Object Management Group (OMG) ha propuesto la arquitectura dirigida por modelos como una especificación de interoperabilidad que permite resolver problemas asociados con el cumplimiento de atributos de calidad. También, propuso la ingeniería dirigida por modelos como una práctica para modelar artefactos de procesos de ingeniería de *software*.

Cada día se producen nuevas versiones de tecnologías, así como necesidades de negocio que demandan aplicaciones de *software* extensibles y escalables. Sin embargo, todas estas acciones requieren un esfuerzo altísimo que puede ser mayor que el de desarrollar una nueva aplicación. De esta forma, cobra mucha importancia la posibilidad de generar aplicaciones de *software* de manera semiautomática.

Este trabajo de investigación presenta una aproximación para generar de manera semiautomática aplicaciones de *software* con la capacidad de satisfacer atributos de calidad requeridos para determinado proyecto. Esta aproximación está basada en la construcción de una cadena de transformación de modelos con base en un modelo de dominio, con el fin de generar código fuente de una aplicación de *software*.

**Palabras clave:** ingeniería dirigida por modelos, aplicaciones móviles, arquitectura limpia, cadenas de transformación de modelos.

## ¿Cómo citar este libro?

Flórez Fernández, H. A. y Sánchez Rodríguez, D. E. (2025). *Desarrollo de aplicaciones móviles mediante ingeniería dirigida por modelos. Caso de estudio basado en Android*. Universidad Distrital Francisco José de Caldas-Editorial UD. <https://doi.org/10.69740/UD.9789587877557.9789587877588>



# Abstract

The software industry must satisfy quality attributes such as maintainability, extensibility, among others. The Object Management Group (OMG) has proposed Model Driven Architecture as an interoperability specification to solve problems associated with meeting quality attributes. It has also proposed Model Driven Engineering as a practice for modelling artefacts of software engineering processes.

New versions of technologies are produced every day, as well as business needs that demand extensible and scalable software applications. However, all these actions require a very high effort that can be higher than developing a new application. Therefore, the possibility of generating software applications in a semi-automatic way becomes very important.

This research work presents an approach to semi-automatically generate software applications with the ability to satisfy quality attributes required for a given project. This approach is based on the construction of a model transformation chain based on a domain model, in order to generate source code of a software application.

**Keywords:** model driven engineering, mobile applications, clean architecture, model transformation chains.

## How to cite this book?

Flórez Fernández, H. A. y Sánchez Rodríguez, D. E. (2025). *Mobile application development using model-driven engineering. Case study based on Android*. Universidad Distrital Francisco José de Caldas-Editorial UD. <https://doi.org/10.69740/UD.9789587877557.9789587877588>



*A nuestras familias*



# Agradecimientos

Los autores agradecen a los miembros del grupo de investigación ITI: Innovación en Tecnologías de la Información adscrito a la Facultad Tecnológica de la Universidad Distrital Francisco José de Caldas, por sus contribuciones en la experimentación realizada en este proyecto.

**Héctor Arturo Flórez Fernández**

Profesor titular de la Universidad Distrital Francisco José de Caldas



© Universidad Distrital Francisco José de Caldas  
© Oficina de Investigaciones  
© Héctor Arturo Flórez Fernández  
© Daniel Esteban Sánchez Rodríguez

**ISBN:** 978-958-787-755-7  
**ISBN digital:** 978-958-787-756-4  
**ISBN ePub:** 978-958-787-758-8

**DOI:** <https://doi.org/10.69740/UD.9789587877557.9789587877588>

Primera edición, abril de 2025

**Líder Unidad de Publicaciones**  
Rubén Eliécer Carvajalino C.

**Gestión editorial**  
Catalina Moreno Correa

**Corrección de estilo**  
HIPERTEXTO SAS

**Diagramación**  
HIPERTEXTO SAS

**Diseño de cubierta**  
Astrid Prieto

#### **Editorial UD**

Universidad Distrital Francisco José de Caldas  
Carrera 24 n.º 34-37 Bogotá, D. C., Colombia  
Teléfono: 6013239300 ext. 6202  
Correo electrónico: [publicaciones@udistrital.edu.co](mailto:publicaciones@udistrital.edu.co)

#### **Cláusula de responsabilidad**

La Editorial UD no es responsable de las opiniones y de la información recogidas en el presente texto. Los autores asumen de manera exclusiva y plena toda responsabilidad sobre su contenido, ya sea este propio o de terceros, declarando en este último supuesto que cuentan con la debida autorización de terceros para su publicación. Igualmente, expresan que no existe conflicto de interés alguno en relación con los resultados de la investigación propiedad de tales terceros. En consecuencia, los autores serán responsables civil, administrativa o penalmente, frente a cualquier reclamo o demanda por parte de terceros, relativa a los derechos de autor u otros derechos que se vulneren como resultado de su contribución.

#### **Todos los derechos reservados**

Esta obra no puede ser reproducida sin el permiso previo escrito de la Unidad de Publicaciones de la Universidad Distrital Francisco José de Caldas.  
Hecho en Colombia.

Sistema de Bibliotecas de la Universidad Distrital Francisco José de Caldas  
Catalogación en la publicación (CEP)

Flórez Fernández, Héctor Arturo  
Desarrollo de aplicaciones móviles mediante ingeniería dirigida por modelos : caso de estudio basado en Android / Héctor Arturo Flórez Fernández, Daniel Esteban Sánchez Rodríguez. -- Primera edición. -- Bogotá : Universidad Distrital Francisco José de Caldas, 2025.  
183 páginas : gráficos, diagramas ; 24 cm. (Espacios)

ISBN: 978-958-787-755-7    ISBN digital: 978-958- 787-756-4

ISBN ePub: 978-958-787-758-8

1. Ingeniería de sistemas 2. Ingeniería de software 3. Aplicaciones móviles -- Diseño 4. Lenguajes específicos de dominio 5. Android (Sistema operativo de dispositivos móviles) I. Sánchez Rodríguez, Daniel Esteban, autor II. Serie.  
005.26: CDD 23 edición.

# Contenido

|                                                        |    |
|--------------------------------------------------------|----|
| <b>Introducción</b>                                    | 19 |
| <b>Capítulo 1. Marco teórico</b>                       | 25 |
| Modelamiento conceptual                                | 25 |
| Arquitectura dirigida por modelos                      | 28 |
| Ingeniería dirigida por modelos                        | 29 |
| Lenguaje de dominio específico                         | 34 |
| Lenguaje de modelamiento de dominio específico         | 37 |
| Eclipse Modeling Framework                             | 39 |
| Arquitectura limpia                                    | 40 |
| <b>Capítulo 2. Cadena de transformación de modelos</b> | 43 |
| Transformación de modelo a modelo                      | 43 |
| Transformación de modelo a texto                       | 45 |
| Diseño de cadena de transformación de modelos          | 46 |
| <b>Capítulo 3. Metamodelo de dominio</b>               | 49 |
| Metodología                                            | 49 |
| Elementos del dominio                                  | 50 |
| Metamodelo MPML                                        | 52 |
| Metamodelo MPML estático                               | 52 |
| Metamodelo MPML dinámico                               | 56 |
| Metamodelo MPML completo                               | 58 |
| Editor gráfico                                         | 59 |

|                                                      |     |
|------------------------------------------------------|-----|
| Elementos del editor gráfico                         | 60  |
| Reglas de sintaxis del editor gráfico                | 62  |
| Ejemplo de modelo usando el editor gráfico           | 64  |
| <b>Capítulo 4. Metamodelo de arquitectura</b>        | 67  |
| Periféricos                                          | 67  |
| Arquitectura limpia                                  | 69  |
| Elementos del metamodelo                             | 71  |
| Elementos de arquitectura limpia                     | 71  |
| Elementos específicos de la plataforma Android       | 73  |
| Metamodelo de arquitectura completo                  | 75  |
| <b>Capítulo 5. Transformación de modelo a modelo</b> | 79  |
| Atlas Transformation Language                        | 79  |
| Reglas de transformación                             | 80  |
| Reglas de transformación directa                     | 80  |
| Reglas de transformación por invocación              | 83  |
| Ejecución de la transformación                       | 94  |
| <b>Capítulo 6. Transformación de modelo a texto</b>  | 99  |
| Acceleo                                              | 99  |
| Kotlin                                               | 101 |
| Módulos                                              | 102 |
| Estructura de los elementos Android generados        | 104 |
| Estructura estática                                  | 104 |
| Estructura dinámica                                  | 105 |
| Desarrollo del proyecto Acceleo                      | 107 |
| Directorios <code>main</code> y <code>request</code> | 107 |
| Directorio <code>common</code>                       | 113 |
| Directorio <code>files</code>                        | 115 |
| <b>Capítulo 7. Validación</b>                        | 119 |
| Caso de estudio GPS                                  | 119 |
| Modelo de dominio                                    | 119 |

|                                            |     |
|--------------------------------------------|-----|
| Transformación de modelo a modelo          | 121 |
| Transformación de modelo a texto           | 121 |
| Caso de estudio sensor                     | 122 |
| Modelo de dominio                          | 123 |
| Transformación de modelo a modelo          | 124 |
| Transformación de modelo a texto           | 125 |
| Caso de estudio NFC                        | 125 |
| Primera aplicación                         | 127 |
| Segunda aplicación                         | 128 |
| <b>Capítulo 8. Discusión</b>               | 131 |
| Síntesis                                   | 131 |
| Trabajo futuro                             | 132 |
| <b>Capítulo 9. Conclusiones</b>            | 135 |
| <b>Referencias</b>                         | 139 |
| <b>Anexo A. Metamodelo de dominio</b>      | 145 |
| <b>Anexo B. Metamodelo de arquitectura</b> | 151 |
| <b>Anexo C. Código fuente ATL</b>          | 159 |
| <b>Anexo D. Código fuente Acceleo</b>      | 173 |
| <b>Autores</b>                             | 179 |



# Introducción

El paradigma de programación orientada a objetos es uno de los más importantes para el desarrollo de *software*, debido a que muchos elementos de desarrollo de *software*, como marcos de trabajo, patrones de diseño de *software*, arquitecturas, entre otros, están relacionados con este paradigma. Por tanto, este paradigma ha sido de gran utilidad en la abstracción de características de la realidad. Sin embargo, en la práctica se han presentado algunas limitaciones que sugieren que el paradigma ha llegado a su punto de agotamiento (Greenfield y Short, 2003).

Además, en los últimos años, el Ministerio de Tecnologías de la Información y las Comunicaciones de Colombia (MinTIC) ha trabajado en el fortalecimiento de varios aspectos de la industria, enmarcados en el fortalecimiento de la *economía digital*, en la que es imperativo implementar procesos tecnológicos para satisfacer las necesidades del mercado. Una de las propuestas del *Plan Vive Digital* consiste en promover la formación académica de colombianos en profesiones de tecnologías de la información para impulsar la innovación, investigación, competitividad y proyección internacional.

Según el MinTIC, la penetración de los teléfonos móviles alcanzó el 118,9 % a finales del año 2015 y el número de servicios de telefonía móvil en Colombia se situó en 57,3 millones. En cuanto a la exportación en *software* y tecnologías de información, desde el 2012 hasta el 2015 hubo un aumento del 36 %; mientras que en los últimos cuatro años en Colombia se ha consolidado la industria de contenidos y aplicaciones digitales.

En el año 2012, el subsector *software* y servicios asociados de Colombia contaba con 39.430 empleados y en el año 2015 aumentó a 70.318. De los datos anteriores, se destaca que en los últimos años en el país ha habido un incremento en el desarrollo de *software*, así como una notoria acogida de las tecnologías móviles y un aumento de los negocios en este sector específico. Por tanto, es pertinente realizar una investigación de esta naturaleza dirigida a brindar conocimiento no solo a la industria, sino también a la academia sobre la construcción de aplicaciones móviles. La creciente comunidad de investigadores en MDE también proporciona los fundamentos teóricos en los que se sustenta esta investigación.

Por tanto, hay ciertos desafíos en la industria del desarrollo de *software* que requieren esfuerzo y tiempo por parte de los desarrolladores para reconocer la estructura del *software* y la interacción entre sus componentes. Entre los desafíos más comunes están el mantenimiento, actualización y extensión del *software*. En la academia y la industria ha habido propuestas de alternativas que no parten propiamente de la programación orientada a objetos, sino que emergen diversas estrategias para hacer frente a estos desafíos como patrones de diseño de *software* o arquitecturas que permitan realizar interoperabilidad entre plataformas como en la arquitectura orientada a servicios (Service Object Architecture-SOA) (Ludewig, 2003).

Por un lado, existe una gran variedad de tecnologías y plataformas en la industria que se utilizan para ejecutar el *software* como servidores de Internet Information Services (IIS) para tecnologías de Microsoft, Apache para PHP, Tomcat, JBoss y Glassfish para Java, NodeJS con JavaScript en el servidor, entre otros. Por tanto, cuando se requiere un cambio de plataforma, es necesario dedicar tiempo y esfuerzo para construir la aplicación nuevamente desde el principio, aunque la lógica de negocios sea la misma. Las mismas reglas de negocio se pueden aplicar al desarrollo móvil. Por ejemplo, para el caso de iOS, es necesario usar Xcode como herramienta de desarrollo y Swift como lenguaje de programación; mientras que para crear aplicaciones para Android existen varias herramientas, como Eclipse o Android Studio, como ambiente de desarrollo integrado (Integrated Development Environment-IDE) y Java como lenguaje de programación.

Actualmente, respecto al desarrollo móvil, existen iniciativas para crear aplicaciones híbridas que utilizan tecnología común para desarrollar y exportar el respectivo instalador para cada plataforma. Por ejemplo, Phonegap y Cordova permiten desarrollar una aplicación solo con tecnologías utilizadas para el desarrollo web front-end, como HTML, CSS y JavaScript, exponiendo conectores para sensores de dispositivos (GPS, acelerómetro, giroscopio), que dan como resultado una API consumible de JavaScript.

Por otro lado, la ingeniería dirigida por modelos (Model Driven Engineering -MDE) es un enfoque para abordar tecnologías relacionadas con lenguajes de modelado de dominio específicos (Domain Specific Modeling Language-DSML), así como motores de transformación y generación (Schmidt, 2006). Los conceptos centrales de MDE son modelos que representan sistemas y metamodelos que definen los lenguajes en los que se expresan los modelos. MDE utiliza modelos para apoyar el diseño, construcción, operación, mantenimiento y modificación de sistemas.

Con base en MDE, las cadenas de transformación de modelos se han presentado de manera revolucionaria como una forma de *tecnificar* la producción de *software*. Por esto, es más importante capacitar a los ingenieros de *software* para que se centren en investigar nuevas tecnologías y crear componentes de *software* reutilizables, en lugar de hacer traducciones entre plataformas y lenguajes de programación.

Asimismo, el acceso a los dispositivos móviles ha aumentado en la última década, por lo que la industria necesita crear cada vez más *software* de manera más eficiente y efectiva que permita cumplir con las necesidades de calidad (Ghezzi *et al.*, 2004).

Este libro de investigación profundiza en el tema de cadenas de transformación de modelos, específicamente para la construcción de productos de *software*. Además, presenta una cadena de transformación de modelos que permite generar de manera semiautomática aplicaciones móviles para plataformas Android que requieran acceso a periféricos en el dispositivo, lo que asegura la misma calidad entre el *software* generado y el *software* desarrollado, teniendo como entrada los mismos modelos conceptuales. Esta aproximación

proporciona conocimientos útiles para la creciente comunidad académica e industrial centrada en la creación de aplicaciones móviles.

Este libro de investigación tiene la siguiente estructura:

- Capítulo 1. Marco teórico: describe los conceptos principales en el marco del proyecto de investigación relacionados con modelamiento conceptual, arquitectura dirigida por modelos, ingeniería dirigida por modelos, *Eclipse Modeling Framework* y arquitectura limpia.
- Capítulo 2. Cadena de transformación de modelos: describe los conceptos de transformación de modelo a modelo, así como transformación de modelo a texto, con el fin de presentar la propuesta de solución mediante el diseño de la cadena de transformación de modelos.
- Capítulo 3. Metamodelo de dominio: presenta la construcción del metamodelo de dominio que se denomina MPML. Para esta construcción, se ha planteado una metodología que permita identificar los elementos del dominio. Además, en este trabajo se creó un lenguaje de modelamiento de dominio específico que ofrece un editor gráfico, para permitir al modelador crear de manera sencilla modelos conformes con el metamodelo de dominio MPML.
- Capítulo 4. Metamodelo de arquitectura: presenta la construcción del metamodelo de arquitectura, el cual se encuentra alineado con los conceptos de arquitectura limpia y de la plataforma Android. Previo a la creación de este metamodelo, se presenta una descripción de los periféricos que incluye el metamodelo, para poderlos abstraer satisfaciendo los requerimientos de arquitectura limpia y Android.
- Capítulo 5. Transformación de modelo a modelo: presenta la conversión de modelo a modelo de la cadena de transformación de modelos. Esta modificación se ha desarrollado usando el lenguaje de dominio específico para transformación de modelos ATL. Además, incluye un conjunto de reglas de transformación que permite tomar cada elemento del modelo de dominio según el metamodelo de dominio

MPML, para generar los correspondientes elementos en el modelo de arquitectura conforme con el metamodelo de arquitectura.

- Capítulo 6. Transformación de modelo a texto: presenta la transformación de modelo a texto de la cadena de transformación de modelos. Esta modificación se ha desarrollado usando el lenguaje de dominio específico Acceleo. Asimismo, incluye un conjunto de módulos que contienen los algoritmos para transformar un modelo conforme con el metamodelo de arquitectura a texto, que corresponde a archivos conformes con el lenguaje de programación Kotlin para Android.
- Capítulo 7. Validación: ilustra tres casos de estudio que demuestran el proceso de creación del modelo de dominio, la ejecución de la cadena de transformación y el resultado final, que es la aplicación en plataforma Android.
- Capítulo 8. Discusión: presenta una síntesis de las contribuciones del proyecto de investigación, y un contraste con los objetivos de investigación propuestos. Además, también presenta orientaciones de posibles investigaciones futuras a partir de los resultados presentados.
- Capítulo 9. Conclusiones: expone los elementos más relevantes que concluyen el proyecto de investigación.



# Capítulo 1. Marco teórico

Este capítulo describe los conceptos más relevantes utilizados en este proyecto de investigación. La sección Modelamiento conceptual describe el proceso de creación de un modelo, y define las características esenciales de los modelos. La sección Arquitectura dirigida por modelos define pautas para diseñar *software* de forma estructurada a través de modelos. La sección Ingeniería dirigida por modelos discute sus elementos fundamentales de la ingeniería dirigida por modelos y propone abstracciones de la realidad mediante modelos y metamodelos. La sección Lenguaje de dominio específico introduce este concepto, el cual es un lenguaje enfocado en solucionar problemas específicos en un dominio en particular. La sección Lenguaje de modelamiento de dominio específico introduce este concepto, que es un lenguaje de modelamiento gráfico orientado a la solución de problemas de un dominio en particular. La sección Eclipse Modeling Framework introduce esta herramienta, la cual está basada en el IDE Eclipse para la creación de proyectos en el contexto de MDE. Finalmente, la sección Arquitectura limpia propone la creación de proyectos basados en capas para asegurar la separación de preocupaciones de los componentes de *software*.

## Modelamiento conceptual

Los cerebros humanos interpretan la realidad tomando información de los órganos sensoriales y creando un modelo mental del mundo basado en dicha información (Hawking y Mlodinow, 2010). Por tanto, modelar es una actividad fundamental para los seres humanos, porque toda decisión

y acción en la vida común está precedida por la construcción explícita o implícita de un modelo. Entonces, si un modelo es incorrecto, la acción tomada con base en este puede ser inapropiada (Bézivin, 2005). Los humanos crean modelos mentales observando la realidad de diferentes maneras. La mayoría de los humanos miran directamente la realidad, pero los humanos ciegos pueden hacer sus observaciones escuchando o tocando la realidad. Sin embargo, los humanos también pueden hacer modelos utilizando información proporcionada por otros humanos, documentos y otros tipos de fuentes de información.

En las últimas décadas, los sistemas informáticos han cobrado gran importancia en todo el mundo y se presume que siga siendo de esta forma en el futuro. En esta área es posible representar los conceptos de un sistema a través de un modelo conceptual. Además, los modelos conceptuales pueden especializarse para proporcionar una representación de un contexto específico.

El modelamiento conceptual se refiere a la actividad de crear modelos conceptuales que abstraen la realidad independientemente de la tecnología y la estrategia utilizadas para resolver un problema específico (Kaschek, 2008). Los modelos conceptuales se han utilizado en varias áreas de la informática. Hoy en día, hay muchas iniciativas basadas en modelos conceptuales para proporcionar abstracción en diferentes niveles (Roussopoulos y Karagiannis, 2009).

Una preocupación importante con respecto al modelamiento conceptual es la calidad que se clasifica en los siguientes tres tipos (Lindland *et al.*, 1994):

- *Calidad sintáctica*: se refiere a qué tan bien se encuentran alineados el modelo con el lenguaje de modelamiento. Si el modelo se adhiere a las reglas del lenguaje, tiene una alta calidad sintáctica.
- *Calidad semántica*: se refiere a qué tan bien el modelo se corresponde con el dominio. Por tanto, el modelo debe garantizar: a) validez, lo que significa que todos los elementos del modelo son correctos y relevantes para el problema y b) integridad, lo que significa que

el modelo contiene todos los elementos que son relevantes para el dominio.

- *Calidad pragmática*: se refiere a qué tan bien el modelo es entendido y utilizado por la audiencia relevante.

Las características requeridas y deseadas de un modelo dependen de su propósito y dominio. Sin embargo, algunos autores han propuesto conjuntos de características genéricas que los modelos deben cumplir independientemente del dominio.

Además, un modelo debe cumplir con las siguientes tres características (Stachowiak, 1973):

- *Característica de mapeo*: se refiere a que un modelo se basa en un original (el original se refiere a un sistema en estudio).
- *Característica de reducción*: se refiere a que un modelo mapea solo una cantidad reducida de propiedades del original.
- *Característica pragmática*: se refiere a que un modelo se puede usar reemplazando el original con respecto a su propósito.

Por ejemplo, un mapa cartográfico es un modelo de la Tierra (*característica de mapeo*), es útil cuando representa un área específica de la Tierra, como países o ciudades (*característica de reducción*), y puede haber varias representaciones de una misma zona, pero con diferentes propiedades específicas; por ejemplo, mapas de carreteras, mapas de ríos o mapas de temperatura (*característica pragmática*).

Además, un modelo debe proporcionar las siguientes características adicionales (Selic, 2003):

- *Abstracción*: permite presentar una simplificación del sistema.
- *Entendimiento*: significa que el modelo debe ser intuitivo.
- *Precisión*: indica que el modelo debe proporcionar una representación fiel del sistema modelado.
- *Predictividad*: implica que el modelo debe predecir las propiedades interesantes del sistema.

- **Económico:** significa que la construcción del modelo debe ser significativamente más económica que el sistema modelado.

Un modelo se considera bueno cuando contiene pocos elementos ajustables, explica correctamente todas las observaciones presentes, permite hacer predicciones sobre observaciones futuras y además es elegante. La elegancia se refiere a la forma de un modelo y está relacionada con su precisión (es decir, la falta de elementos ajustables). Cuando el modelo es exacto y no requiere ajustes, observarlo debe ser equivalente a observar la realidad. En consecuencia, el modelo puede servir para predecir estados futuros de la realidad (Hawking y Mlodinow, 2010).

## Arquitectura dirigida por modelos

La arquitectura dirigida por modelos (*Model Driven Architecture-MDA*) es un enfoque liderado por la OMG orientado en el diseño, el desarrollo y la implementación de productos de *software*. MDA proporciona una guía para estructurar especificaciones de *software* expresadas mediante modelos.

El aspecto principal de MDA es ayudar a los diferentes usuarios de computadoras a resolver problemas de integración, proporcionando especificaciones de interoperabilidad abiertas e independientes del proveedor. Por tanto, MDA es el siguiente paso de OMG para resolver problemas de integración (Frankel *et al.*, 2001).

MDA investiga para una estandarización de diferentes plataformas y proveedores de *software*, en ese caso almacena los modelos en el *Meta Object Facility (MOF)* y utiliza UML como lenguaje para crear los modelos. MOF proporciona un marco abierto para tratar con modelos, esto define los niveles utilizados por MDA y es el metametamodelo aceptado por la arquitectura MDA. Además, define el intercambio de modelos utilizando *XML Metadata Interchange-XML*. MOF define dos estándares MOF completo (CMOF) y MOF esencial (EMOF) (Object Management Group, 2015).

MDA es una propuesta adecuada para resolver problemas presentes en los procesos de construcción de *software* que introduce el modelo independiente de plataforma (*Platform Independent Model-PIM*) y el modelo específico

de plataforma (*Platform Specific Model-PSM*) (Bézivin *et al.*, 2003). Sin embargo, para hacer una separación de preocupaciones, los modelos necesitan dimensiones más específicas como área temática (por ejemplo, el área que procesa pedidos en una empresa o el área de nómina), propuesta en el paradigma de programación orientada a objetos (Harrison y Ossher, 1993).

## Ingeniería dirigida por modelos

La ingeniería dirigida por modelos (*Model Driven Engineering-MDE*) es un enfoque para abordar tecnologías relacionadas con lenguajes de modelamiento de dominio específico, así como motores de transformación y generación (Schmidt, 2006). Desde hace varias décadas, MDE adapta a la ingeniería de *software*, principios y prácticas bien entendidas y establecidas de la ingeniería de sistemas. MDE se usa ampliamente en empresas que desarrollan diferentes tipos de *software* porque ofrece varios beneficios en torno a la productividad y consistencia mediante la generación automática de código fuente.

MDE proporciona conceptos comunes para comprender las nociones básicas de modelos y metamodelos que se definen:

- *Modelo*: es una representación abstracta simplificada de un sistema bajo estudio que permite razonar sobre el sistema, enfocándose en los detalles deseados y relevantes para lograr un propósito previsto.
- *Metamodelo*: es una especificación que define conceptos y su relación en un dominio particular y especifica restricciones asociadas con los conceptos definidos.

MDE utiliza modelos para apoyar el diseño, desarrollo y actualización de sistemas de *software*. El uso de modelos específicos de dominio permite a los modeladores obtener información fundamental del sistema en estudio, con un nivel de detalle adecuado dentro de su dominio (Muller *et al.*, 2012). Posteriormente, los modelos son utilizados para la automatización del trabajo, lo que permite a los ingenieros concentrar sus esfuerzos hacia los elementos intelectualmente exigentes del sistema en estudio.

Un modelo contiene declaraciones sobre el sistema modelado, y es correcto si todas las declaraciones son verdaderas para dicho sistema (Seidewitz, 2003). Un sistema puede estar representado por diferentes modelos, cada uno representa aspectos específicos y se caracteriza por un metamodelo dado (Bézivin, 2005). Además, para un aspecto específico, puede haber metamodelos reutilizables en diferentes sistemas. Sin embargo, varios sistemas pueden pertenecer al mismo dominio; entonces, el metamodelo que representa dicho dominio define sus conceptos y relaciones, facilitando el proceso de modelado de esos sistemas comunes (Chaudhary *et al.*, 2022).

La tecnología basada en el paradigma de orientación a objetos era prometedora (Bézivin, 2005). Por ejemplo, la extensibilidad de la aplicación se puede aplicar a través de la herencia de clases. Sin embargo, actualmente, un tipo importante de extensibilidad de la aplicación se implementa con complementos, como lo hace el sistema IDE Eclipse; aunque este concepto de complemento *plug-in* no está relacionado con la herencia de clases.

En este sentido, MDE destaca y tiene como objetivo las representaciones abstractas del conocimiento y las actividades que gobiernan un dominio de aplicación particular, en lugar de los conceptos informáticos. Con relación con MDE, existe una extensa terminología, conceptos y áreas de estudio. Una de estas áreas se relaciona con las cadenas de transformación de modelos.

La forma actual de desarrollar *software* utiliza modelos del lenguaje de modelado unificado (*Unified Modeling Language-UML*) únicamente en las fases de análisis y diseño. Sin embargo, no tiene responsabilidad de la implementación en el código fuente, lo que puede producir una desconexión de la implementación con el análisis y diseño.

Cuando la arquitectura de *software* está bien definida en las fases de análisis y diseño, la puesta en producción es más sencilla. Sin embargo, en un entorno real, la fase de implementación implica algunos problemas, ya que los equipos de desarrollo están cambiando dinámicamente. La persona que inicia la construcción del *software* no necesariamente es la misma que la termina. Los requerimientos del *software* también son dinámicos, por lo

que el *software* debe cumplir con estos cambios de requerimientos, lo que a veces implica hacer los ajustes en el código fuente dejando el modelo sin estos cambios.

Por tanto, se deben realizar manualmente los cambios correspondientes en el modelo. Además, si el *software* necesita ejecutarse en otras plataformas, sistemas operativos o entornos, entonces el equipo de desarrollo debe reconstruir el sistema para migrar a la nueva plataforma. Si en principio el *software* debe ejecutarse en varias plataformas, entonces construir una aplicación para cada plataforma implica más tiempo y costos para el equipo de desarrollo.

Con los conceptos de MDE es posible construir un metamodelo de dominio con el cual sea posible construir modelos específicos de cada cliente o problema en particular. Luego, el modelador construye un modelo que es conforme con ese metamodelo de dominio, y este es el insumo principal de una cadena de transformación de modelos que, en la última transformación, da como resultado un conjunto de archivos de texto en un formato específico. De esta manera, una cadena de transformación de modelos dará como resultado una o varias aplicaciones integradas en un lenguaje de programación definido para ejecutarse en una plataforma específica.

Con este proceso, se producirá un desacoplamiento conceptual entre el diseño mediante el modelado y la implementación que involucra plataformas de ejecución y lenguajes de programación. Con este nuevo punto de vista, es apropiado hablar de otros roles como *modeladores* y *metamodeladores*, quienes serán los encargados de crear los modelos y metamodelos mediante un lenguaje de modelado.

En modelamiento de dominio específico (Domain Specific Modeling-DSM), un DSML admite abstracciones para dominios específicos. Por el contrario, un lenguaje de modelado de propósito general (General Purpose Modeling Language-GPML) representa las entidades y propiedades de un sistema general. Un GPML bien conocido es UML, que es un lenguaje estándar adecuado para modelar sistemas con un alcance muy amplio que cubre un conjunto grande y diverso de dominios de aplicación (Open Management

Group, 2005). No se limita al *software* de modelado, sino que puede modelar diferentes tipos de sistemas estándar (Rumbaugh *et al.*, 2004).

El Object Management Group (OMG)<sup>1</sup> definió una arquitectura que cuenta con cuatro niveles de abstracción. Esta se presenta en la figura 1. En esta arquitectura, las relaciones pueden ser *instancia de* y *conforme con*. El primero se utiliza para relacionar el sistema con el modelo, mientras que el segundo sirve para relacionar un modelo con el metamodelo correspondiente, así como para relacionar un metamodelo con el metametamodelo correspondiente. La relación *conforme con* indica que los elementos, atributos y relaciones del modelo deben cumplir con los conceptos y reglas definidos en el metamodelo con el cual el modelo es conforme. Esto generalmente significa que:

- El elemento de un modelo debe ser una instancia de un metatipo definido en el metamodelo con el cual el modelo es conforme.
- Las relaciones entre los elementos del modelo deben ser instancias de relaciones entre metatipos sujetas a las reglas de cardinalidad ya descritas.
- Los atributos de un elemento de un modelo deben ser los únicos definidos para los metatipos correspondientes.

En la arquitectura, el nivel  $M_0$  se refiere al sistema en estudio, el cual es una instancia de un modelo, lo que indica que el sistema es representado a través de un modelo. El modelo corresponde al nivel  $M_1$  y es conforme con un metamodelo, que define los conceptos y relaciones del dominio y se ubica en el nivel  $M_2$ . El metamodelo es conforme con un metametamodelo colocado en el nivel  $M_3$ , el cual es conforme consigo mismo.

Además, las transformaciones de modelos son muy importantes en el enfoque de MDE. Por tanto, el desarrollo de definiciones de transformación de modelos es una actividad bastante frecuente en los procesos de desarrollo de *software* mediante un enfoque dirigido por modelos. En consecuencia, los ingenieros de *software* deben contar con el apoyo necesario para realizar

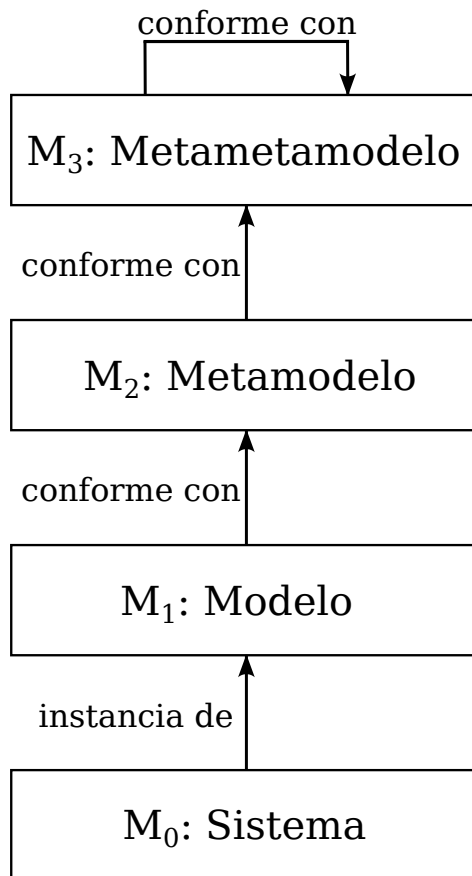
---

<sup>1</sup> <http://www.omg.org/>

esta tarea mediante herramientas maduras de MDE compatibles con IDE, compiladores y depuradores clásicos en su trabajo de programación (Jouault *et al.*, 2008).

En este proyecto de investigación, los conceptos de MDE se utilizan para desarrollar cadenas de transformación de modelos, con el fin de ofrecer herramientas que permitan generar código fuente para aplicaciones móviles en plataforma Android. La arquitectura de cuatro niveles se utiliza para crear varios DSML para el contexto de aplicaciones móviles. Estos DSML se especifican a través de metamodelos que definen conceptos y reglas del negocio.

**Figura 1.** Arquitectura de cuatro niveles



Fuente: Bézivin, 2005

Por tanto, los modelos deben ser conformes con uno de estos metamodelos; es decir, cada modelo debe cumplir las reglas especificadas en el correspondiente metamodelo. Además, estos metamodelos deben ser conformes con el metamodelo proporcionado por la tecnología utilizada para crear los DSML.

## Lenguaje de dominio específico

Un lenguaje de dominio específico (Domain Specific Language-DSL) es un lenguaje de programación o un lenguaje de especificación que se dedica a un dominio de problema en particular, un método de resolución de problemas en particular o una estructura de datos en particular (Mernik *et al.*, 2005). Se adapta a un dominio de aplicación particular, en lugar de ser un lenguaje de programación de propósito general. Los DSL están diseñados para expresar los conceptos particulares de un dominio en una forma más explícita y fácil de usar que un lenguaje de programación de propósito general. A menudo se utilizan para simplificar tareas complejas y acelerar el desarrollo (Iung *et al.*, 2020). Las características principales de un DSL son (Van Deursen y Klint, 2002):

- *Simplicidad.* Se debe diseñar un DSL para simplificar la tarea en estudio y hacerla más fácil de entender.
- *Especificidad del dominio.* Un DSL debe diseñarse para satisfacer las necesidades de un dominio en particular y debe tener características que sean específicas de ese dominio.
- *Flexibilidad.* Un DSL debe diseñarse para ser lo suficientemente flexible para que cuente con la capacidad de adaptarse a diferentes enfoques y soluciones.
- *Legibilidad.* Un DSL debe diseñarse para que los expertos en el dominio puedan leerlo y comprenderlo fácilmente.
- *Reutilización.* Un DSL debe diseñarse para que sea reutilizable y fácilmente extensible, de modo que pueda usarse en diferentes dominios.

- *Eficiencia*. Un DSL debe diseñarse para ser eficiente, de modo que pueda usarse para resolver problemas de manera rápida y confiable.

Algunos lenguajes que se clasifican como un DSL son:

- *Structured Query Language (SQL)*<sup>2</sup>. Es un lenguaje de consulta estructurado utilizado para administrar datos en bases de datos relacionales.
- *Hypertext Markup Language (HTML)*<sup>3</sup>. Es un lenguaje de marcado de hipertexto utilizado para crear páginas web.
- *Cascading Style Sheets (CSS)*<sup>4</sup>. Es un lenguaje de hojas de estilo en cascada utilizadas para diseñar páginas web.
- *LaTeX*<sup>5</sup>. Es un lenguaje tipográfico utilizado para crear documentos técnicos.
- *Verilog*<sup>6</sup>. Es un lenguaje de descripción de hardware utilizado para diseñar circuitos digitales.

Algunas herramientas que sirven para crear un DSL son:

- *ANother Tool for Language Recognition (ANTLR)*<sup>7</sup>. Es un lenguaje que permite la generación de analizadores de código fuente, con el fin de leer, procesar, ejecutar o traducir texto estructurado o archivos binarios. Es normalmente utilizado para crear lenguajes de programación de dominio específico, herramientas de desarrollo de *software* y marcos de trabajo. Con base en una gramática, ANTLR permite generar un analizador que puede construir y analizar gramaticalmente árboles de decisión (Parr y Fisher, 2011).
- *Xtext*<sup>8</sup>. Es un marco de trabajo de código abierto basado en Eclipse para crear lenguajes de programación y un DSL. Proporciona un entorno de desarrollo de lenguaje completo que incluye resaltado

<sup>2</sup> <https://www.iso.org/standard/63555.html>

<sup>3</sup> <https://html.spec.whatwg.org/>

<sup>4</sup> <https://www.w3.org/TR/CSS/>

<sup>5</sup> <https://www.latex-project.org/>

<sup>6</sup> <https://ieeexplore.ieee.org/document/1620780/>

<sup>7</sup> <https://www.antlr.org/>

<sup>8</sup> <https://www.eclipse.org/Xtext/>

de sintaxis, finalización automática, validación y depuración, además de permitir a los desarrolladores definir sintaxis textuales e integrarlas en IDE basados en Eclipse. También, se puede usar para crear intérpretes, compiladores y generadores de código (Bettini, 2016).

- *MetaEdit*<sup>9</sup>. Es un marco de trabajo de lenguaje visual y generador de código. Es una herramienta de desarrollo de un DSL diseñada para ayudar a reducir el costo y el tiempo relacionados con el desarrollo de productos de *software* integrados. Proporciona un enfoque gráfico basado en MDE para el desarrollo de *software* integrado, lo que permite a los desarrolladores crear y modificar fácilmente modelos de *software* y generar código a partir de ellos (Kelly y Tolvanen, 2021).

La metodología para crear un DSL se define en los siguientes pasos (Mernik *et al.*, 2005):

- *Evaluar el dominio*. Antes de desarrollar un DSL, es importante evaluar el dominio con el fin de encontrar la mejor opción para resolver el problema en estudio. El dominio debe analizarse a fondo para comprender las capacidades y limitaciones del lenguaje.
- *Diseñar la sintaxis*. Una vez evaluado el dominio, se debe diseñar la sintaxis del lenguaje. Esto incluye definir la gramática del lenguaje, los tipos de datos del lenguaje y todas las estructuras de control del lenguaje.
- *Implementar el lenguaje*. Después de determinar la sintaxis, se puede implementar el lenguaje que consiste en escribir el código que analizará e interpretará la sintaxis del lenguaje.
- *Probar y depurar el lenguaje*. Una vez implementado el lenguaje, es importante probarlo para asegurarse de que funciona correctamente. Se debe realizar una depuración para corregir cualquier error que se encuentre.

---

<sup>9</sup> <https://www.metacase.com/products.html>

- *Desplegar el lenguaje.* Después de probado y depurado el lenguaje, se puede desplegar. Esto implica poner el lenguaje a disposición de los usuarios y proporcionar documentación y soporte.

## Lenguaje de modelamiento de dominio específico

Un lenguaje de modelamiento de dominio específico (*Domain-Specific Modeling Language-DSML*) es un lenguaje de modelamiento gráfico que se adapta a un dominio de aplicación particular. Está diseñado para simplificar la descripción, visualización, construcción y documentación de modelos de dominio. Es una forma de programación visual y utiliza elementos gráficos para representar acciones, decisiones y otros elementos que forman parte del dominio de la aplicación (Kardas *et al.*, 2010). Las características principales de un DSML son (Sousa *et al.*, 2012):

- *Diseño.* Un DSML utiliza elementos gráficos para el modelado facilitando su comprensión y uso.
- *Flexibilidad.* Un DSML está diseñado para ser extensibles y personalizables para permitir a los usuarios modificar fácilmente el lenguaje para sus fines particulares.
- *Automatización.* Un DSML utiliza generadores de código para automatizar la generación de código a partir de modelos.
- *Reutilización.* Un DSML debe estar diseñado para permitir que los componentes se reutilicen en múltiples proyectos.
- *Facilidad de uso.* Un DSML debe estar diseñado para minimizar la complejidad del uso del lenguaje, con el fin de facilitar que los usuarios aprendan y comprendan el lenguaje rápidamente.

Algunos lenguajes que se clasifican como un DSML son:

- *Unified Modeling Language (UML)*<sup>10</sup>. Es un lenguaje que se utiliza para crear diagramas y modelos de aplicaciones y sistemas de *software*.

---

<sup>10</sup> <http://www.uml.org/>

- *Systems Modeling Language (SysML)*<sup>11</sup>. Es un lenguaje que se utiliza para modelar proyectos de ingeniería de sistemas y para analizar las relaciones entre los componentes de un sistema.
- *Yet Another Next Generation (YANG)*. Es un lenguaje que se utiliza para modelar redes de datos y otros sistemas de comunicación.
- *Entity Relationship Diagram (ERD)*. Es un lenguaje que se utiliza para modelar las relaciones entre entidades en una base de datos.
- *Business Process Model and Notation (BPMN)*<sup>12</sup>. Es un lenguaje para modelar y visualizar procesos de negocio.
- *Business Process Execution Language (BPEL)*<sup>13</sup>. Es un lenguaje que se utiliza para describir procesos de negocios usados en arquitectura orientada a servicios.
- *Simple Interface for Modeling Programming Languages (SIMPL)*. Es un lenguaje que se utiliza para diseñar e implementar lenguajes de programación.
- *XML Process Definition Language (XPDL)*<sup>14</sup>. Es un lenguaje basado en XML utilizado para definir procesos de negocio.
- *Semantics of Business Vocabulary and Rules (SBVR)*<sup>15</sup>. Es un lenguaje que se utiliza para expresar reglas y conceptos de negocio utilizados en la gestión de procesos de negocio.
- *Geographic Markup Language (GML)*<sup>16</sup>. Es un lenguaje basado en XML que se utiliza para describir características geográficas y sus relaciones para sistemas de información geográfica.

---

<sup>11</sup> <https://sysml.org/>

<sup>12</sup> <https://www.bpmn.org/>

<sup>13</sup> <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>

<sup>14</sup> <http://xpdl.org/>

<sup>15</sup> <https://www.omg.org/spec/SBVR/>

<sup>16</sup> <https://www.ogc.org/standards/gml>

## Eclipse Modeling Framework

El proyecto Eclipse Modeling Framework (EMF)<sup>17</sup> es un marco de modelado basado en el IDE Eclipse<sup>18</sup>, para construir herramientas y aplicaciones desde un modelo de datos estructurados. Entonces, con base en una especificación de modelo descrita en formato XML Metadata Interchange (XMI), EMF proporciona herramientas para producir: a) un conjunto de clases en lenguaje de programación Java para representar un modelo; b) un conjunto de clases de adaptadores que permiten la visualización y edición del modelo con base en comandos relacionados con el mismo y c) un editor de modelamiento básico (Steinberg *et al.*, 2008).

EMF se basa en EMF (*core*), que es un estándar común para modelos de datos. Este estándar incluye soluciones de servidor, persistencia, interfaz de usuario y soporte para transformaciones. EMF se compone de tres partes fundamentales (Budinsky *et al.*, 2004):

- *Metametamodelo EMF*. EMF incluye el metamodelo *Ecore* para describir metamodelos, modelos, soporte en tiempo de ejecución para los modelos, soporte de persistencia con serialización en formato XMI predeterminada y una API reflexiva para manipular objetos EMF de forma genérica.
- *EMF.Edit*. Es un componente que incluye clases genéricas reutilizables para crear editores para modelos EMF. *EMF.Edit* ofrece los siguientes elementos:
  - Conjunto de clases de proveedores de etiquetas y contenido, compatibilidad con fuentes de propiedades y otras clases prácticas que permiten que los modelos EMF se desplieguen, usando vistas de escritorio estándar y vistas de propiedades.
  - Marco de comandos que incluye un conjunto de clases de implementación de comandos genéricos para compilar editores que admiten deshacer y rehacer acciones de manera automática.

<sup>17</sup> <https://www.eclipse.org/modeling/emf/>

<sup>18</sup> <https://www.eclipse.org/>

- *EMF.Codegen*. Es un componente que ofrece la función de generación de código, el cual tiene la capacidad de generar todos los componentes necesarios para construir un editor de modelos completo para un modelo EMF. Incluye una interfaz de usuario, con la cual es posible especificar opciones de generación de código fuente.

EMF admite tres niveles de generación de código (Budinsky *et al.*, 2004):

- *Modelo*. Proporciona interfaces en lenguaje de programación Java, clases de implementación para todas las clases del modelo.
- *Adaptadores*. Genera clases de implementación denominadas *Item Providers* que adaptan las clases del modelo para su edición y visualización.
- *Editor*. Produce un editor estructurado que se ajusta al estilo recomendado para los editores de modelos EMF de Eclipse y sirve como punto de partida para la personalización del mismo editor.

## Arquitectura limpia

La arquitectura limpia (*Clean Architecture*) es una arquitectura de *software* propuesta por Robert C. Martin, derivada de otras propuestas arquitectónicas como la arquitectura hexagonal, la arquitectura cebolla, entre otras, la cual se basa en la organización de los componentes del sistema mediante capas (Martin, 2018).

El objetivo fundamental de la arquitectura limpia es la separación de preocupaciones (conocido como *separation of concerns*), dividiendo los proyectos de *software* en capas. De esta forma, la arquitectura permite construir sistemas de *software* con las siguientes características (Martin, 2018):

- *Independiente de Frameworks*. La arquitectura no debe depender de la inclusión de alguna librería de *software* en un proyecto. Esto le permite utilizar marcos de trabajo en lugar de incluir en el sistema componentes que tienen sus propias restricciones.

- *Comprobable*. Los requerimientos funcionales de la aplicación se pueden probar sin hacer uso de una interfaz de usuario ni de algún elemento externo como servidor web o base de datos.
- *Independiente de la interfaz de usuario*. La interfaz de usuario debe estar completamente desacoplada de la lógica de negocio. Esto significa que la interfaz de usuario puede modificarse sin impactar otros componentes de la aplicación. Por ejemplo, una interfaz de usuario web se puede reemplazar con una interfaz de usuario de escritorio o una interfaz de usuario de aplicaciones móviles, sin cambiar reglas del negocio.
- *Independiente de la persistencia*. El sistema puede cambiar el sistema de persistencia, que puede ser bases de datos relacionales, bases de datos NoSQL o archivos, debido a que las reglas de negocio no están vinculadas a la persistencia. Por ejemplo, el sistema puede cambiar de base de datos relacional como Postgres o SQL Server por base de datos NoSQL como Mongo o Firebase.
- *Independiente de cualquier elemento externo*. Las reglas de negocio no tienen conocimiento del exterior del sistema.

La arquitectura limpia se compone de las siguientes cuatro capas, las cuales tienen un propósito específico (Rachovski *et al.*, 2022):

- *Frameworks and Drivers*. Esta es la capa de infraestructura que se compone de elementos externos al sistema. Estos elementos son bases de datos, redes, interfaces externas, elementos de entrada/-salida, interfaz de usuario, dispositivos móviles, dispositivos IoT, entre otros.
- *Interface Adapters*. Esta es la capa de adaptadores que permite convertir datos que se encuentran en un formato conveniente para los componentes relacionados con las reglas de negocio del sistema, a un formato conveniente para algún elemento externo como la base de datos. En esta capa también se incluye cualquier adaptador requerido para convertir datos de algún formato externo, al formato interno utilizado para satisfacer las reglas de negocio del sistema.

- *Application Business Rules*. Esta es la capa de aplicación que contiene *software* con reglas de negocio de la aplicación. Esta capa encapsula e implementa todos los casos de uso del sistema.
- *Enterprise Business Rules*. Esta es la capa de dominio que encapsula las reglas de negocio de la organización. Se compone de entidades que representan los conceptos fundamentales del negocio con sus propiedades y operaciones. Estas entidades pueden ser clases con atributos y métodos o también estructuras de datos con variables y funciones.

La arquitectura limpia tiene una regla de dependencia que indica que las dependencias del código fuente solo pueden estar dirigidas hacia las capas internas. Es decir, que los elementos de la capa *Enterprise Business Rules* no tienen acceso a la capa *Application Business Rules* y esta no tiene acceso a la capa *Interface Adapters* y esta no tiene acceso a la capa *Frameworks and Drivers* (Arango y Loaiza, 2021).

## Capítulo 2. Cadena de transformación de modelos

Este capítulo explica las técnicas de transformación de modelo a modelo y transformación de modelo a texto. Con base en estas dos técnicas, se presenta la sección Diseño de cadena de transformación de modelos, la cual está basada en EMF y lenguajes de transformación de modelo a modelo y de modelo a texto.

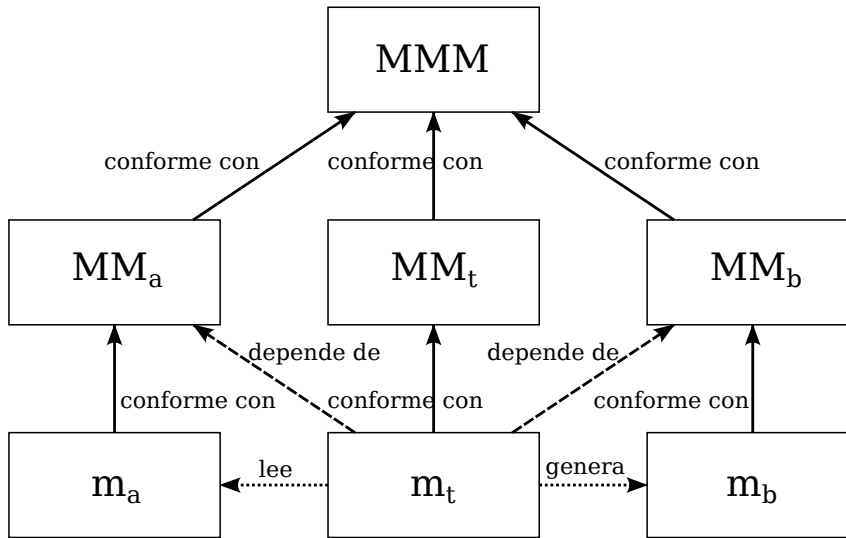
### Transformación de modelo a modelo

Una transformación de modelo a modelo en MDE es una forma automatizada de modificar y crear modelos. Una transformación de modelos garantiza que una familia de modelos sea consistente con base en reglas de negocio establecidas para un dominio específico. El objetivo principal de usar una transformación de modelo a modelo, es reducir esfuerzo y minimizar los riesgos de cometer errores en la creación de modelos a través de la automatización para la creación y para la modificación de modelos (Bruehl *et al.*, 2020). La figura 2 representa conceptualmente una transformación de modelo a modelo con las siguientes características:

- La transformación permite convertir un modelo  $m_a$  que es conforme con un metamodelo  $MM_a$  a un modelo  $m_b$  que es conforme con un metamodelo  $MM_b$ .
- La transformación está definida por un modelo de transformación  $m_t$  que es conforme con un metamodelo de transformación  $MM_t$ .

Además, la transformación  $m_t$  depende de los metamodelos  $MM_a$  y  $MM_b$ , lee como entrada el modelo  $m_a$  y genera como salida el modelo  $m_b$ .

**Figura 2.** Transformación de modelo a modelo



Fuente: elaboración propia

- Los metamodelos  $MM_a$  y  $MM_b$  deben tener relaciones directas entre conceptos para permitir la solución a los requerimientos de negocio establecidos para un proyecto en particular.
- Los metamodelos  $MM_a$ ,  $MM_b$  y  $MM_t$  deben ser conformes con un metametamodelo como *Meta Object Facility (MOF)*<sup>1</sup>. En la tecnología EMF, el metametamodelo es *Ecore*<sup>2</sup>.

Para realizar transformaciones entre modelos, es necesario utilizar un lenguaje especializado para esta tarea. Dos lenguajes comunes son *Atlas Transformation Language (ATL)*<sup>3</sup> y *Query View Transformation Operational (QVTo)*<sup>4</sup>.

<sup>1</sup> <https://www.omg.org/mof/>

<sup>2</sup> <https://wiki.eclipse.org/Ecore>

<sup>3</sup> <https://www.eclipse.org/atl/>

<sup>4</sup> <https://projects.eclipse.org/projects/modeling.mmt.qvt-oml>

Por un lado, ATL permite la transformación mediante la definición de módulos.

Un módulo debe contener un nombre, un metamodelo de entrada y un metamodelo de salida.

Además, un módulo puede definir reglas, y cada una debe contener un nombre, la clase del metamodelo de entrada y una o más clases del metamodelo de salida. Cada regla permite crear clases en el metamodelo de salida y asignar los valores correspondientes a cada atributo de cada clase del metamodelo de salida (Oakes *et al.*, 2018).

Por otro lado, QVTo permite desarrollar transformaciones creando funciones de mapeo entre el modelo de entrada y el modelo de salida. En este lenguaje, es necesario incluir los metamodelos de entrada y salida a través de la cláusula `modelType`. La ejecución de la transformación debe ser iniciada mediante una función principal, la cual puede apelar a las funciones de mapeo requeridas para ejecutar la transformación y así generar el modelo de salida (Kahani *et al.*, 2019).

## Transformación de modelo a texto

Una transformación de modelo a texto permite generar archivos de texto con el contenido y extensión deseados, de tal forma que dicho texto generado puede ser código fuente en cualquier lenguaje de programación. Las transformaciones de modelo a texto siempre tienen un modelo y un metamodelo de entrada. La salida es el conjunto de archivos de código fuente generados en el lenguaje de programación seleccionado para implementar la aplicación final.

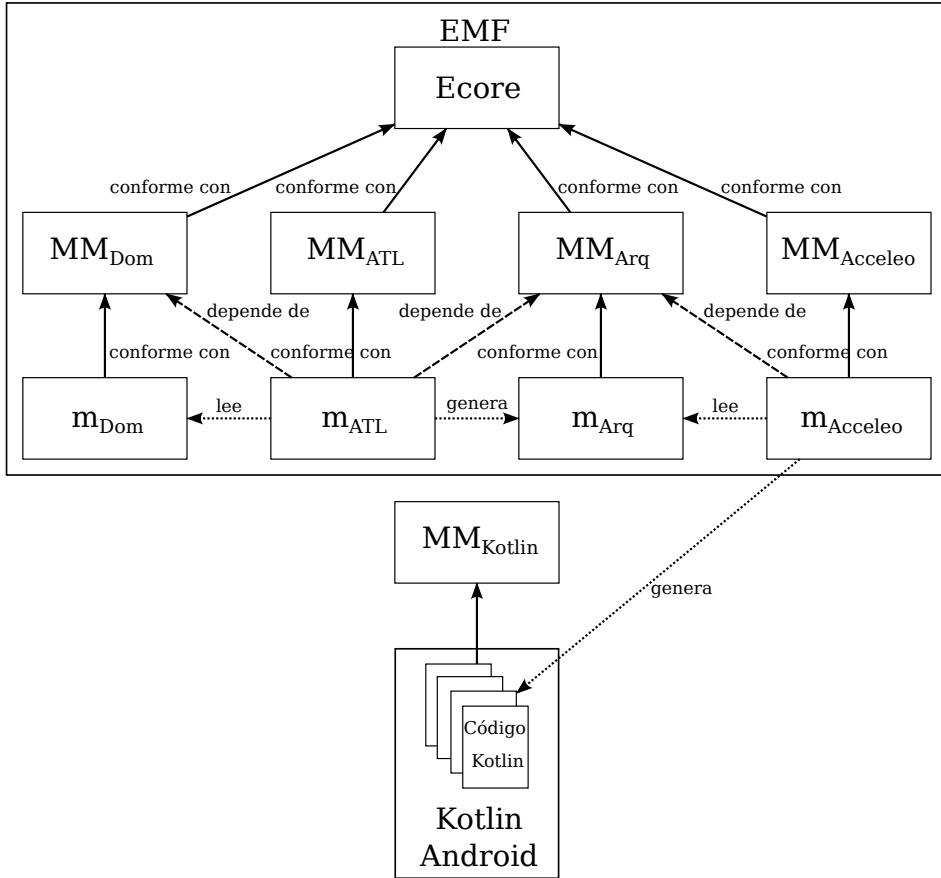
Acceleo<sup>5</sup> es un proyecto que permite desarrollar este tipo de transformaciones. Acceleo permite construir un conjunto de plantillas que son los componentes en los que se implementan las decisiones de la transformación. También tiene una instrucción `@main` que indica qué plantillas se deben ejecutar. Si una plantilla no contiene la instrucción `@main`, entonces, la plantilla debe ser llamada desde otra para su ejecución (Di Sipio *et al.*, 2021).

<sup>5</sup> <https://www.eclipse.org/acceleo/>

## Diseño de cadena de transformación de modelos

La estrategia utilizada para construir una cadena de transformación de modelos es representada en la figura 3, la cual se encuentra en el marco trabajo de EMF e incluye los siguientes artefactos:

- Metametamodelo *Ecore*. Este metamodelo es usado porque es el estándar utilizado en la tecnología EMF.
- Metamodelo de dominio  $MM_{Dom}$ . Este metamodelo es conforme con *Ecore* y contiene todos los elementos relacionados con el dominio específico, el cual, en este caso, corresponde al desarrollo en dispositivos móviles específicamente para la configuración de periféricos.
- Metamodelo ATL  $MM_{ATL}$ . Este metamodelo es conforme con *Ecore* y contiene los elementos para el desarrollo de transformaciones de modelo a modelo usando el lenguaje ATL.
- Metamodelo de arquitectura  $MM_{Arq}$ . Este metamodelo es conforme con *Ecore* y contiene todos los elementos del metamodelo de dominio  $MM_{Dom}$ , agregando elementos relacionados con los conceptos arquitectónicos de Android, conceptos del paradigma de orientación a objetos en el lenguaje de programación Java, conceptos de MDA y conceptos de arquitectura limpia.
- Metamodelo Acceleo  $MM_{Acceleo}$ . Este metamodelo es conforme con *Ecore* y contiene los elementos para el desarrollo de transformaciones de modelo a texto, usando el lenguaje Acceleo.
- Metamodelo Kotlin. Este metamodelo contiene los elementos que hacen referencia a los conceptos del lenguaje de programación Kotlin.
- Modelo de dominio  $m_{Dom}$ . Este modelo es conforme con el metamodelo de dominio  $MM_{Dom}$  y permite definir los conceptos y reglas para el desarrollo de un producto específico de *software* para dispositivos móviles con sistema Android, teniendo en cuenta la configuración de los periféricos requeridos.

**Figura 3.** Diseño de cadena de transformación de modelos

Fuente: elaboración propia

- Modelo de transformación ATL  $m_{ATL}$ . Este modelo es conforme con el metamodelo ATL  $MM_{ATL}$  y depende del metamodelo de dominio  $MM_{Dom}$  y del metamodelo de arquitectura  $MM_{Arq}$ . Además, tiene como entrada el modelo de dominio  $m_{Dom}$  y define las reglas de transformación necesarias para transformar los conceptos modelados en el modelo de dominio.
- Modelo de arquitectura  $m_{Arq}$ . Es conforme con el metamodelo de arquitectura  $MM_{Arq}$  y resulta de la ejecución del modelo de transformación ATL  $m_{ATL}$ .

- Modelo transformación Acceleio  $m_{Acceleio}$ . Este modelo es conforme con el metamodelo Acceleio  $MM_{Acceleio}$  y depende del metamodelo de arquitectura  $MM_{Arq}$ . Además, tiene como entrada el modelo de arquitectura  $m_{Arq}$  y define las reglas de transformación necesarias para transformar los conceptos modelados en el modelo de arquitectura  $m_{Arq}$  para generar los archivos de código fuente.
- Código fuente. Corresponde a los archivos generados en la ejecución del modelo de transformación Acceleio  $m_{Acceleio}$ . Estos archivos corresponden a código fuente conforme con el lenguaje de programación Kotlin.

Con esta propuesta, el modelo de dominio y el modelo de arquitectura son independientes de plataforma, pero los archivos generados por el modelo de transformación Acceleio dependen de la plataforma; es decir, dependen de Kotlin como lenguaje de programación para el desarrollo de aplicaciones Android. Como consecuencia, es necesario crear una transformación Acceleio diferente para otra plataforma como, por ejemplo, lenguaje de programación Objective C para el desarrollo de aplicaciones iOS.

## Capítulo 3. Metamodelo de dominio

Este capítulo presenta la primera parte de la creación de la cadena de transformación de modelos, que es la creación del metamodelo de dominio. Para crear este metamodelo, se propone una Metodología que puede ser generalizada a cualquier dominio. Posteriormente, se presenta la identificación de los elementos del dominio en la sección Elementos del dominio, los cuales son el insumo principal del metamodelo de dominio. Asimismo, la sección Metamodelo MPML presenta detalladamente el proceso de creación del mismo, debido a que este también es el punto fundamental del lenguaje de modelamiento de dominio específico creado para diseñar modelos. Finalmente, como parte del lenguaje de modelamiento de dominio específico se ha creado un editor gráfico.

### Metodología

En la literatura relacionada con MDE, no hay una metodología explícita que permita extraer los conceptos de un dominio específico, con el fin de crear su correspondiente metamodelo de dominio. Por tanto, el diseño del metamodelo de dominio para este proyecto se obtuvo siguiendo cuatro pasos que podrían usarse de manera genérica en otros dominios. Estos cuatro pasos son los siguientes:

1. Identificación de los elementos del dominio. Este paso consiste en crear una lista de los elementos del dominio que son representados a través del metamodelo.

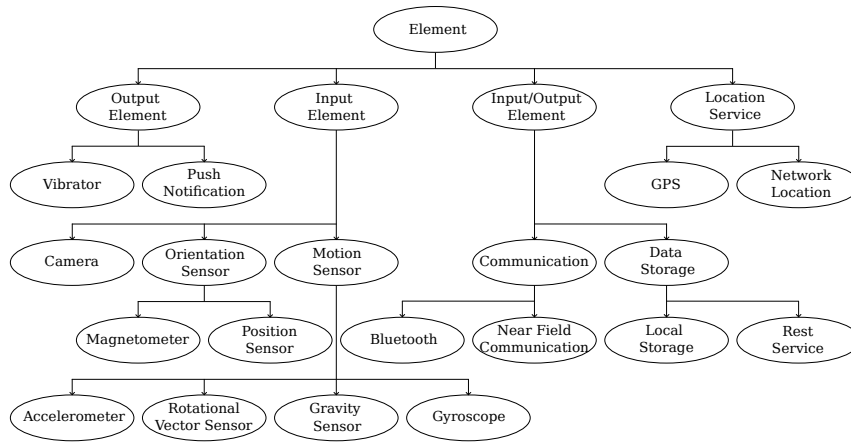
2. Identificación de la interacción entre los elementos. Este paso consiste en identificar la interacción entre los elementos extraídos en el paso anterior; es decir, las relaciones dentro de los elementos y algunos comportamientos especiales que se deseaban modelar.
3. Creación del metamodelo. Una vez identificados los elementos y las relaciones del dominio, se lleva a cabo este paso, que consiste en incluir los elementos y relaciones en el metamodelo de dominio. Esto implica identificar e incluir sus atributos. Además, incluir las relaciones en el metamodelo implica definir el tipo de cada relación y su cardinalidad.
4. Diseño de un DSML para el metamodelo. Este consiste en la creación de un lenguaje de modelado que incluya un editor gráfico, para permitir al modelador crear modelos que sean conformes con el metamodelo de dominio. Para este propósito, se utilizó un enfoque diferente al de los elementos del metamodelo.

## Elementos del dominio

El dominio de este proyecto se encuentra enmarcado en la configuración de periféricos en dispositivos móviles. A partir de los elementos identificados, fue desarrollada la taxonomía presentada en la figura 4. Esta taxonomía inicia con el concepto *Element* que tiene el mayor nivel de abstracción. A partir de este concepto, la taxonomía tiene los siguientes conceptos (Sánchez y Flórez, 2018):

- *Input elements*. Son elementos de solo entrada de datos en un dispositivo móvil. Estos elementos son:
  - *Camera*. Esta entrada representa las cámaras frontal y trasera incorporadas en los dispositivos móviles.
  - *Orientation sensors*. Estos sensores miden la posición física de un dispositivo. Esta categoría incluye *magnetometers* y *position sensors*.

**Figura 4.** Taxonomía de elementos de dominio



Fuente: elaboración propia

- *Motion sensors.* Son sensores que miden fuerzas de aceleración y de rotación en tres coordenadas. Esta categoría incluye *accelerometers*, *rotational vector sensors*, *gyroscopes* y *gravity sensors*.
- *Output elements.* Son elementos de solo salida de datos en un dispositivo móvil. Estos elementos son *vibrator* y *push notifications*.
- *Input/Output Elements.* Son elementos de entrada y salida de datos en un dispositivo móvil. Estos elementos son:
  - *Communication.* Son elementos que hacen la interacción con otros dispositivos usando diferentes protocolos como *Bluetooth* y *NFC* (*Near Field Communication*).
  - *Data Storage.* Son elementos que permiten la persistencia de datos usados en una aplicación móvil, con el fin de poder modelar operaciones *CRUD* (*Create, Read, Update and Delete*). Estos elementos son *Local Storage*, que permite el almacenamiento de datos localmente en el dispositivo móvil, y *Rest Services*, que permite modelar las operaciones *CRUD* usando la especificación *RestFull*.

- *Location Services*. Son elementos que permiten identificar la posición del dispositivo. Estos elementos son *GPS* y *Network Location*.

## Metamodelo MPML

En este proyecto, el metamodelo de dominio no solo permite hacer la abstracción de los elementos de la realidad, sino que también permite crear un DSML que se denomina *Mobile Peripheral Modeling Language* (MPML). De esta forma, el metamodelo de dominio en este proyecto se denomina metamodelo MPML.

El metamodelo MPML se divide en dos partes fundamentales que son metamodelo MPML estático y metamodelo MPML dinámico.

### Metamodelo MPML estático

El metamodelo MPML estático contiene los elementos del metamodelo que hacen referencia únicamente a los componentes estructurales del dominio. Se compone de las siguientes clases:

- **MPMLComponent**. En cuanto a los periféricos, este es el mayor nivel de abstracción del metamodelo. A partir de este elemento, se representa cada sensor de entrada o elemento de salida a modelar en el dispositivo. Por tanto, a partir de este elemento se heredan todos los periféricos.
- **MPMLInputComponent**. Permite modelar los sensores de entrada de acuerdo con la clasificación de sensores presentada en el sitio web de desarrolladores de Android<sup>1</sup>. Estos sensores de entrada son parte del metamodelo como clases que heredan de **MPMLInputComponent**. Estas clases son:
  - **MotionSensor**. Representa sensores que miden las fuerzas de aceleración y las fuerzas de rotación a lo largo de tres ejes. Esta categoría incluye acelerómetros, sensores de gravedad, giroscopios y sensores de vectores rotacionales.

---

<sup>1</sup> <https://developer.android.com/>

- **EnvironmentalSensor**. Representa sensores que miden varios parámetros ambientales como temperatura y presión del aire del ambiente, iluminación y humedad. Esta categoría incluye barómetros, fotómetros y termómetros.
- **PositionSensor**. Representa sensores que miden la posición física de un dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.
- **MPMLOutputComponent**. Permite modelar elementos de los teléfonos móviles que se van a utilizar únicamente con el fin de dar una salida al usuario. Dentro de estos elementos, Android cuenta con vibrador, mensajes de texto SMS, lista de interfaz gráfica y discador telefónico.
- **DataStorage**. Permite modelar elementos de interacción con los datos, incluyendo lo concerniente a la capacidad de manipular datos en un mecanismo externo o interno del dispositivo. En esta categoría se añadieron dos elementos concretos:
  - **NFCWriter**. Permite modelar la interacción para escribir información en texto plano a otros dispositivos o en Tags MIFARE NFC.
  - **RealTimeDatabase**. Permite modelar el almacenamiento de datos mediante operaciones CRUD. Para no depender de la implementación de la lógica del servidor, este componente utiliza *Firestore Real-Time Database*<sup>2</sup>.
- **MPMLModeledComponent**. Este tipo de componente contiene un modelo, es decir, los datos que el periférico proporciona normalmente como salida. Por ejemplo, el GPS proporciona un modelo como salida (`location:{latitud:double, longitude:double}`). También puede ser un modelo de entrada que especifica las restricciones de entrada a un periférico. Por ejemplo, el vibrador especifica un modelo que es el tiempo para vibrar en milisegundos (`vibratorModel:time:int`).

---

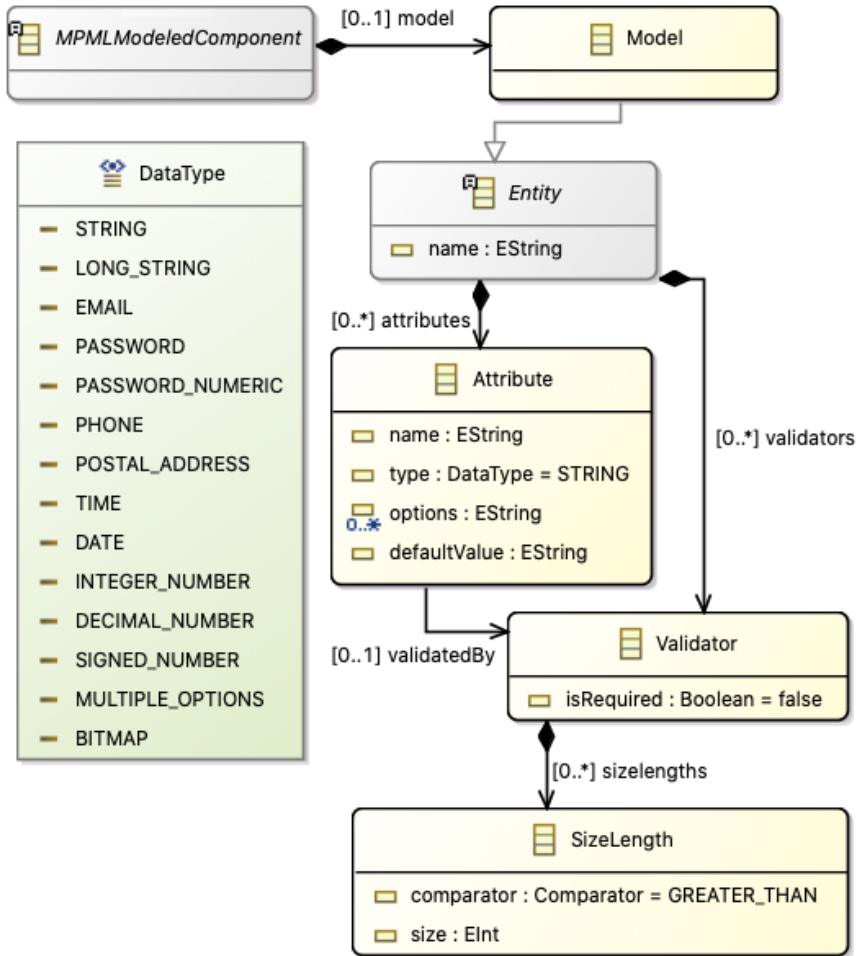
<sup>2</sup> <https://firebase.google.com/products/firestore>

- **Entity.** Es la representación abstracta de cualquier modelo que se vinculará a un `MPMLModeledComponent`. Todos los elementos relacionados con el elemento `Entity` se pueden ver en la figura 5. Cada entidad puede tener varios atributos con su propio nombre y `DataType`. Los `DataTypes` soportados son los siguientes, aunque pueden ser extendidos:

- `STRING`
- `LONG_STRING`
- `EMAIL`
- `PASSWORD`
- `PASSWORD_NUMERIC`
- `PHONE`
- `POSTAL_ADDRESS`
- `TIME`
- `DATE`
- `INTEGER_NUMBER`
- `DECIMAL_NUMBER`
- `SIGNED_NUMBER`
- `MULTIPLE_OPTIONS`
- `BITMAP`

En el caso de que un atributo sea `MULTIPLE_OPTIONS`, entonces las opciones disponibles para elegir están en el atributo `options` como una colección de tipo `EString`. Finalmente, los atributos pueden tener un valor predeterminado de tipo `EString`. Cada atributo tiene su propio validador dado por la clase `Validator`, que indica si el valor del atributo es obligatorio, así como varios elementos, como `SizeLength` que permite especificar un tipo de comparador a partir de los valores `GREATER_THAN`, `LESS_THAN`, y `EQUALS_TO` y un tamaño tipo `EInt`.

**Figura 5.** Elementos relacionados con el elemento Entity del metamodelo MPML



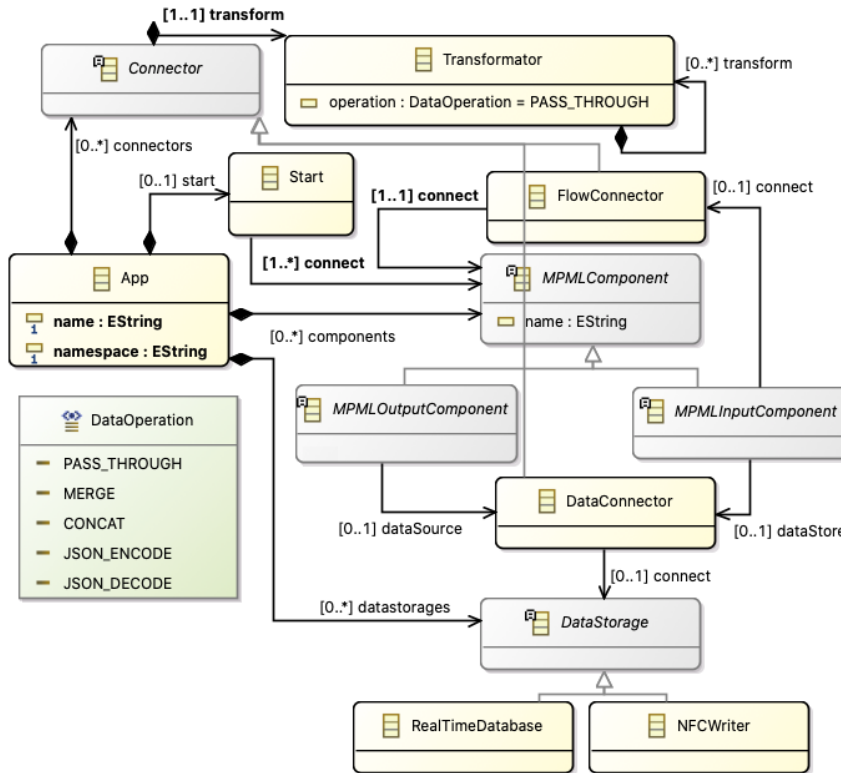
Fuente: elaboración propia

En general, los elementos del modelo relacionados con cualquiera de los **MPMLModeledComponent** heredan del elemento abstracto **Entity**.

La taxonomía presentada en la figura 4 permite la identificación de los elementos a representar en MPML. La tabla 1 muestra los diferentes modelos de los periféricos de entrada. Todos ellos tienen un atributo `dataStore` que apunta a un **RealTimeDatabase** o **NFCWriter** donde se almacenan

los datos. La tabla 2 muestra los diferentes modelos de los periféricos de salida. Todos ellos tienen un atributo `dataSource` que apunta a un `RealTimeDatabase` donde se van a recuperar los datos.

**Figura 6.** Elementos de flujo de datos del metamodelo MPML



Fuente: elaboración propia

## Metamodelo MPML dinámico

El metamodelo MPML estático contiene los elementos del metamodelo que hacen referencia únicamente a los componentes comportamentales del dominio. Este metamodelo incorpora elementos que permiten el modelamiento del flujo de datos entre componentes.

**Tabla 1.** Periféricos de entrada

| Periférico        | Modelo                                       |
|-------------------|----------------------------------------------|
| UIcreateForm      | Configurable                                 |
| NFCReader         |                                              |
| GPS               | location: {latitude:double,longitude:double} |
| Camera            | photo (Bitmap)                               |
| MotionSensor      | Predefinido dependiendo del sensor           |
| PositionSensor    | Predefinido dependiendo del sensor           |
| EnvironmentSensor | Predefinido dependiendo del sensor           |

Fuente: elaboración propia

**Tabla 2.** Periféricos de salida

| Periférico   | Modelo                                 |
|--------------|----------------------------------------|
| UICollection | Configurable                           |
| NFCWriter    |                                        |
| Vibrator     | time:int                               |
| PhoneCaller  | phoneNumber: String                    |
| SMSSender    | {phoneNumber: String, message: String} |

Fuente: elaboración propia

Para lograr este flujo de datos, el metamodelo estático fue aumentado con nuevos elementos que lograron modelar el transporte y las transformaciones del flujo de datos sobre los diferentes periféricos. La figura 6 presenta el metamodelo con los elementos dinámicos para modelar el flujo de datos. Esta fracción del metamodelo MPML incluye los siguientes elementos:

- **Start.** Desde la perspectiva del flujo de datos, este es el primer punto de acceso donde se encadenan todos los periféricos para enviar y recibir modelos y manejar las transformaciones de datos sobre ellos.

- **Connector.** Este elemento es una definición abstracta de la conexión entre los diferentes componentes. Hay dos tipos de conectores concretos:
  - **FlowConnector.** Permite la conexión entre el elemento `InputComponent` y un elemento `OutputComponent`.
  - **DataConnector.** Conecta los elementos `InputComponent` y `OutputComponent` a un `DataStorage`. La conexión de los `InputComponent` se llama `DataStore` y la conexión de los `OutputComponent` se llama `DataSource`.

El elemento `Connector` tiene un elemento `Transformator`, el cual tiene una referencia recursiva a sí mismo para permitir encadenar varias transformaciones sobre la misma `Entity`. Además, el elemento `Transformator` contiene una operación que se puede seleccionar a partir de los siguientes valores predefinidos, los cuales pueden ser aumentados: `PASS_THROUGH`, `MERGE`, `CONCAT`, `JSON_ENCODE` y `JSON_DECODE`.

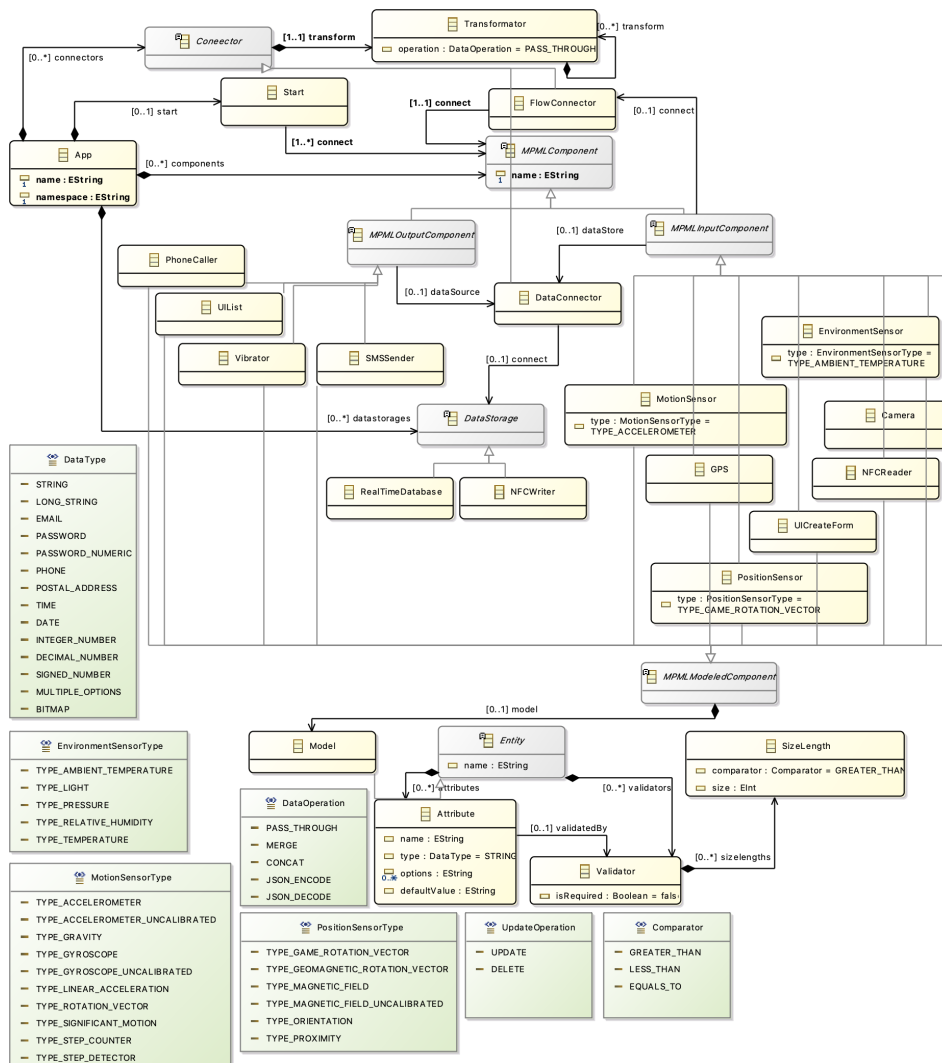
- **App.** Este elemento define el nombre y el *namespace* de la aplicación. Además, se conecta al elemento `Start` para definir todo el flujo de datos desde la perspectiva dinámica del modelo.

Desde la perspectiva estática del modelo, el modelo contiene las siguientes colecciones de elementos: componentes (0 a muchos elementos `MPMLComponent`), conectores (0 a muchos elementos `Connector`), conectores (0 a muchos elementos `DataStorage`), inicio (solamente un punto de acceso `Start`).

## Metamodelo MPML completo

En resumen, la figura 7 presenta el metamodelo de dominio MPML completo, donde están involucrados todos los elementos presentados en el metamodelo MPML estático y dinámico.

Figura 7. Metamodelo de dominio MPML



Fuente: elaboración propia

## Editor gráfico

Un lenguaje tiene una sintaxis y una semántica, en la cual la sintaxis corresponde a las reglas gramaticales del lenguaje, mientras que la semántica corresponde al significado de los modelos que son conformes con el lenguaje (Schmidt, 1996).

En este sentido, el lenguaje de modelamiento de dominio específico desarrollado en este proyecto denominado MPML, tiene una semántica expresada por el metamodelo MPML y tiene una sintaxis que corresponde a las restricciones que debe cumplir los modelos conformes con el metamodelo MPML. Estas restricciones fueron desarrolladas en el lenguaje *Acceleo Query Language (AQL)*<sup>3</sup> usando Eclipse Sirius<sup>4</sup>. Con base en Eclipse Sirius, MPML incluye un editor gráfico para facilitar al modelador la tarea de crear modelos conformes con el metamodelo MPML.

### Elementos del editor gráfico

El editor gráfico de MPML permite al modelador dibujar y conectar las diferentes interacciones entre los periféricos con solo arrastrar y soltar acciones. La figura 8 presenta los elementos gráficos que están disponibles para el modelador. Estos elementos gráficos son:

- *Camera*. Periférico de entrada que ayuda a activar la cámara del dispositivo móvil.
- *GPS*. Periférico de entrada que permite activar el sensor GPS para obtener la ubicación del dispositivo móvil.
- *Real time Database*. Periféricos de entrada y salida para leer y escribir datos en una base de datos.
- *Vibrator*. Periférico de salida que permite al modelador hacer vibrar el dispositivo móvil.
- *Environment sensor*. Periférico de entrada que permite leer varias condiciones ambientales del entorno, como la intensidad de la luz.
- *Motion sensor*. Periférico de entrada que permite leer diferentes sensores de movimiento compatibles con la plataforma Android<sup>5</sup>.
- *Position sensor*. Periférico de entrada que permite leer diferentes sensores de posición compatibles con la plataforma Android<sup>6</sup>.

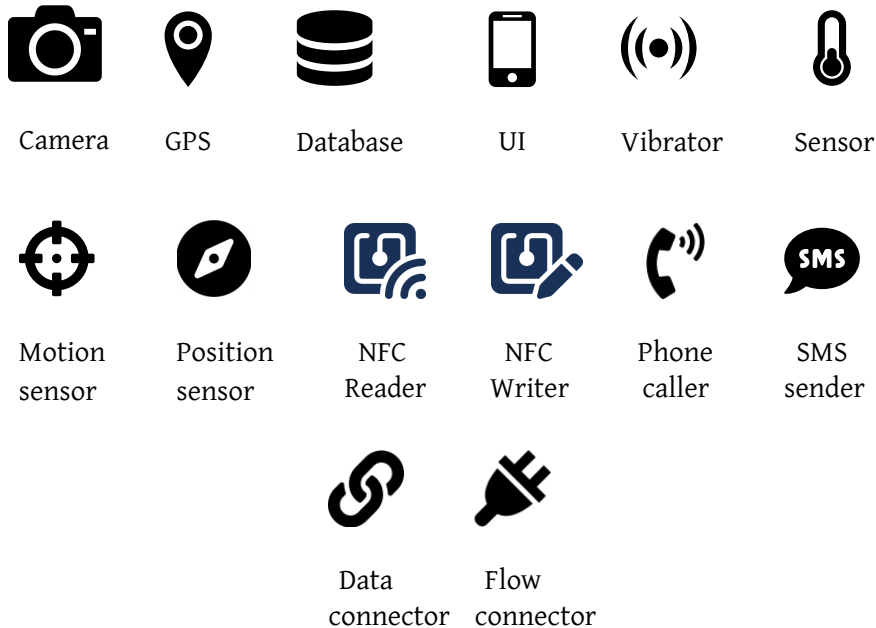
<sup>3</sup> <https://www.eclipse.org/acceleo/documentation/>

<sup>4</sup> <https://www.eclipse.org/sirius/>

<sup>5</sup> [https://developer.android.com/guide/topics/sensors/sensors\\_motion](https://developer.android.com/guide/topics/sensors/sensors_motion)

<sup>6</sup> [https://developer.android.com/guide/topics/sensors/sensors\\_position](https://developer.android.com/guide/topics/sensors/sensors_position)

**Figura 8.** Elementos gráficos de entrada y de salida



**Fuente:** elaboración propia

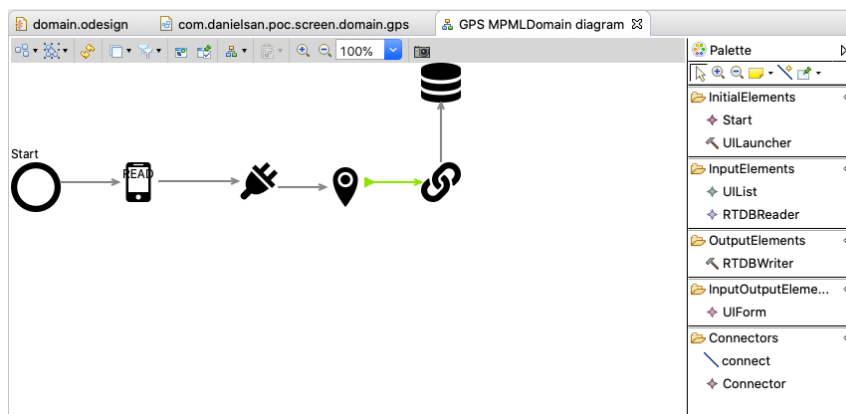
- *NFC Reader*. Periférico de entrada que permite leer datos de cualquier dispositivo móvil que soporte la tecnología NFC.
- *NFC Writer*. Periférico de salida que permite escribir datos a cualquier dispositivo móvil que soporte la tecnología NFC.
- *Phone caller*. Permite iniciar la aplicación de llamada de forma pre-determinada enviando el número de teléfono correspondiente.
- *SMS sender*. Permite enviar mensajes SMS con base en el número de teléfono y el mensaje correspondiente.
- *Data Connector*. Permite la conexión desde cualquier componente a un componente que servirá como DataStore (actualmente RealTimeDatabase, NFCReader y NFCWriter).
- *Flow Connector*. Permite la conexión desde cualquier componente a otro componente, para permitir lanzar uno tras otro; por ejemplo,

una pantalla que inserta una entidad y luego mostrará otra pantalla que presenta la lista de entidades insertadas.

La figura 9 presenta una impresión de pantalla del editor gráfico de MPML. Este editor cuenta con una área gráfica de modelamiento y una paleta de elementos. Esta impresión de pantalla muestra un caso de uso típico: una aplicación que presenta un elemento de pantalla que lee la ubicación del GPS y, posteriormente, almacena la información de ubicación en una base de datos. El flujo de la aplicación es la siguiente:

- Comienza con su elemento Start.
- Inicia un UIElement que es del tipo READ.
- El UIElement se conecta a un GPS desde donde el UIElement va a leer la ubicación.
- El GPS se conecta a un elemento RealTimeDatabase que almacena la ubicación del dispositivo móvil cada vez que el elemento GPS lea un cambio en la ubicación del dispositivo móvil.

**Figura 9.** Editor gráfico de MPML



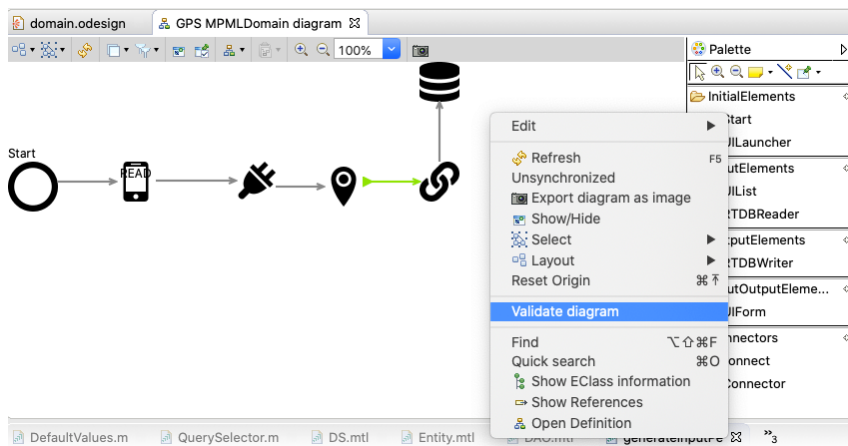
Fuente: elaboración propia

## Reglas de sintaxis del editor gráfico

El editor gráfico de MPML también incluye una opción denominada *Validate diagram* que sirve para verificar que el modelo se creó correctamente; es

decir, que el modelo es conforme con el metamodelo MPML y que el modelo cumple con las reglas de sintaxis definidas para el lenguaje. El hecho de que el modelo sea conforme con el metamodelo, no garantiza que el modelo sea correcto sintácticamente. Para acceder a la validación, es necesario hacer clic derecho sobre el editor gráfico, el cual despliega un menú emergente que tiene la opción *Validate diagram*, como se muestra en la figura 10.

**Figura 10.** Validar diagrama en el editor gráfico de MPML



Fuente: elaboración propia

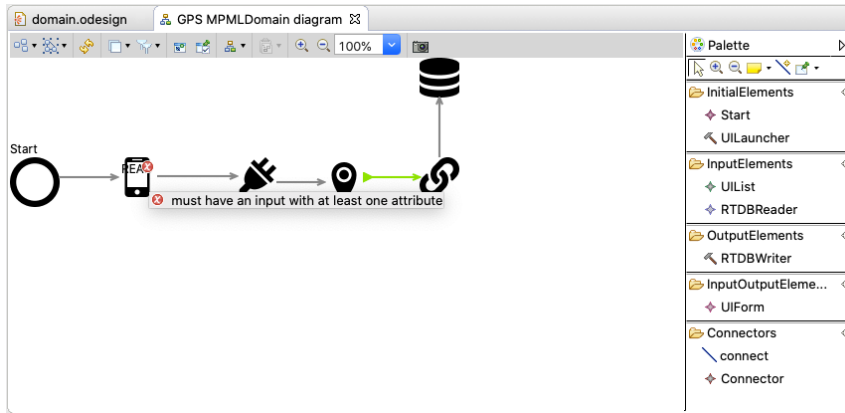
Algunas reglas de validación están incorporadas en el propio metamodelo MPML. Por ejemplo, los campos obligatorios como en la relación entre *FlowConnector* y *MPMLComponent* que tiene una cardinalidad de 1 indica que se requiere ese atributo y debe estar conectado.

Mientras que un atributo se define como opcional en el metamodelo, hay algunos elementos para los que se requiere un valor en dicho atributo, como en los elementos *UICreateForm* y *UIList*. En estos casos, la siguiente regla de validación es incluida y es parte del lenguaje MPML.

```
aql:self.input.attributes -> notEmpty()
```

Una vez se ejecuta la validación desde el editor gráfico, se despliegan sobre el modelo mensajes emergentes indicando el error de sintaxis del modelo. Por ejemplo, la figura 11 presenta el resultado de validar un modelo, en el que se presenta un error en un elemento *UICreateForm*.

**Figura 11.** Mensaje de validación en el editor gráfico de MPML



Fuente: elaboración propia

El mensaje de error es *must have a model with at least one attribute*, el cual indica que al elemento le hace falta un atributo. Las demás reglas sintácticas aplicadas son:

1. El elemento `DataStorage NFCWriter` no se puede conectar a través de un atributo `dataStore`, a pesar de que el metamodelo MPML lo permite.
2. El nombre y el namespace son obligatorios en el elemento `App`.

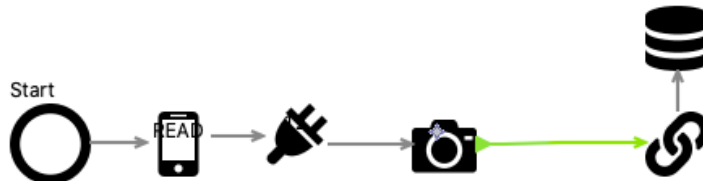
## Ejemplo de modelo usando el editor gráfico

La figura 12 presenta la representación de algunos elementos dinámicos del editor gráfico de MPML en un ejemplo. Este ejemplo muestra la interacción entre los componentes, donde todos los elementos están conectados secuencialmente entre ellos.

1. `Start`. Es el primer punto de acceso de la aplicación.
2. `UICreateForm`. La conexión directa desde el elemento `Start` al elemento `UICreateForm` representa una ventana en blanco en la aplicación.

3. FlowConnector. Este elemento permite conectar el UICreateForm al periférico Camera.
4. Camera. Representa la cámara del dispositivo móvil.
5. DataConnector. El conector de datos vincula el elemento Camera con el elemento RealTimeDatabase. Esto indica que el modelo de Camera (en este caso la imagen) se almacenará en un Realtime Database.
6. RealTimeDatabase. Representa la base de datos en tiempo real usando Firestore.

**Figura 12.** Ejemplo de modelo conforme con el metamodelo MPML



Fuente: elaboración propia



## Capítulo 4. Metamodelo de arquitectura

Este capítulo presenta la segunda parte de la creación de la cadena de transformación de modelos, que es la creación del metamodelo de arquitectura. Para crear este metamodelo, la sección Periféricos propone una jerarquía de periféricos. Posteriormente, en la sección Arquitectura limpia se explica cómo el metamodelo satisface la arquitectura limpia y sus principios. La sección Elementos del metamodelo presenta los elementos del metamodelo clasificados por los elementos de arquitectura limpia y elementos específicos de la plataforma Android. Finalmente, se presenta la sección Metamodelo de arquitectura completo.

### Periféricos

El metamodelo de arquitectura define una jerarquía de periféricos, con el fin de cumplir los requerimientos específicos para el desarrollo de aplicaciones Android. En este metamodelo, el elemento de mayor jerarquía que representa los periféricos es la clase abstracta `Peripheral`, de la cual heredan todos los periféricos de la plataforma Android.

De la clase `Peripheral` heredan las siguientes clases abstractas:

- `InputPeripheral`. Representa todos los elementos que ingresan datos al sistema. De esta clase heredan las siguientes clases:

- **NFCReader**. Representa el periférico de entrada que permite leer datos de cualquier dispositivo móvil que soporte la tecnología NFC.
- **RealTimeDatabase**. Representa el periférico de entrada para leer datos en base de datos.
- **MotionSensor**. Representa el periférico de entrada que permite leer diferentes sensores de movimiento compatibles con la plataforma Android.
- **PositionSensor**. Representa el periférico de entrada que permite leer diferentes sensores de posición compatibles con la plataforma Android.
- **EnvironmentSensor**. Representa el periférico de entrada que permite leer varias condiciones ambientales del entorno como la intensidad de la luz.
- **Camera**. Representa el periférico de entrada que ayuda a activar la cámara del dispositivo móvil.
- **GPS**. Representa el periférico de entrada que permite activar el sensor GPS para obtener la ubicación del dispositivo móvil.
- **OutputPeripheral**. Representa todos los elementos que exportan datos del sistema. De esta clase heredan las siguientes clases:
  - **RealTimeDatabase**. Representa el periférico de salida para escribir datos en base de datos.
  - **Vibrator**. Representa el periférico de salida que permite al modelador hacer vibrar el dispositivo móvil.
  - **NFC Writer**. Representa el periférico de salida que permite escribir datos a cualquier dispositivo móvil que soporte la tecnología NFC.

## Arquitectura limpia

La arquitectura seleccionada en este proyecto es arquitectura limpia presentada por Robert C. Martin (Martin, 2018), como un enfoque de ingeniería de *software* basada en componentes (*Components Based Software Engineering-CBSE*) centrado en el dominio, que permite explicar todos los elementos del dominio, con la que se puede llevar a cabo una separación de responsabilidades relacionadas con las funcionalidades específicas de la plataforma y las independientes de plataforma.

Con respecto a este enfoque, adoptar una arquitectura en capas es un buen punto de partida, pero el problema es que una vez que el *software* crece en complejidad, habrá tres grandes contenedores de código que no están separando el código correctamente (Fowler, 2015). Robert C. Martin define *detalles* como componentes de *software* que no están estrechamente relacionados con las reglas de negocio y que también pueden ser sustituidos. Estos pueden incluir los dispositivos IO, bases de datos, sistemas web, servidores, frameworks, protocolos de comunicación. Así, la lógica de negocio debe estar *limpia* para poder ser reutilizable en caso de que los *detalles* sufran algún cambio (Martin, 2000).

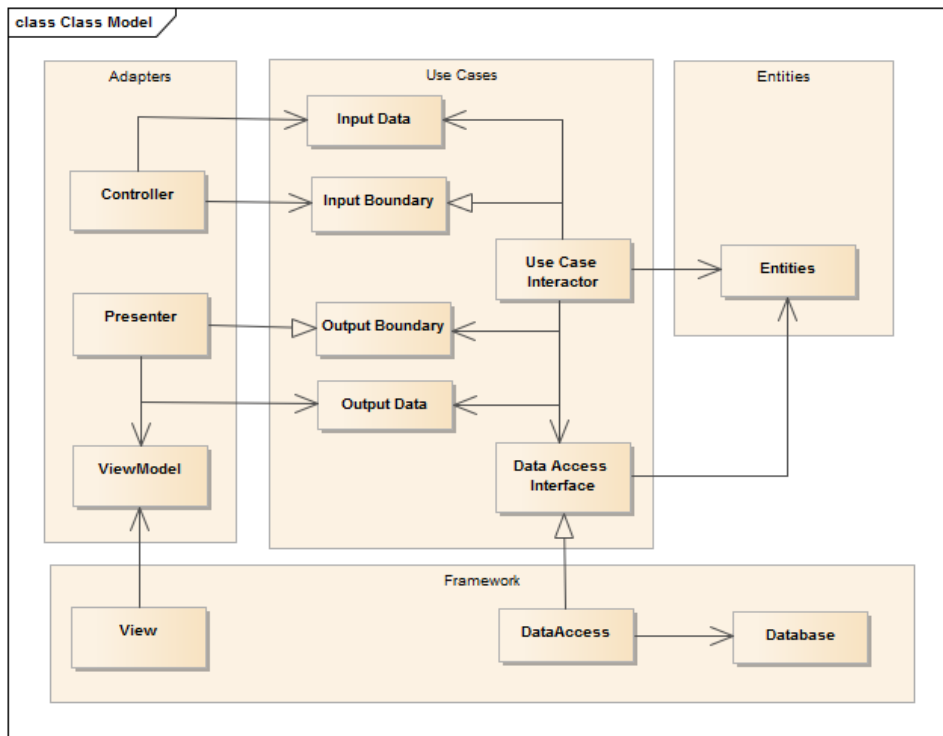
Este concepto engloba muy bien atributos de los periféricos de Android, que son el concepto fundamental para la construcción del metamodelo de arquitectura. Así, esta es la razón principal por la que se seleccionó la arquitectura limpia.

En este proyecto, se creó una arquitectura en la que las políticas están aisladas de aquellos componentes conectables, es decir, tecnología de acceso a datos Firestore, UI, GPS, Camera, etc.

La figura 13 presenta un diagrama de clases, que es un escenario típico de separación de componentes que utilizan esta arquitectura para una app Android que necesita conectarse a una base de datos. En este diagrama, Input Data, Output Data y ViewModel son objetos de transferencia de datos transportados entre los diferentes Input Boundary y Output Boundary. El Input Boundary es la interfaz que define el contrato que el controlador está utilizando para invocar servicios, mientras que el

Output Boundary es la interfaz que el presentador está implementando. El Data Access Interface también forma parte del Output Boundary para los casos de uso; por tanto, es la interfaz que implementa la clase Data Access (Sánchez *et al.*, 2022).

**Figura 13.** Separación de componentes



Fuente: elaboración propia

Un elemento importante de la arquitectura son los casos de uso, porque son el núcleo de la lógica de negocio y definen los límites para separar el código fuente independiente de la plataforma del código fuente específico de la plataforma.

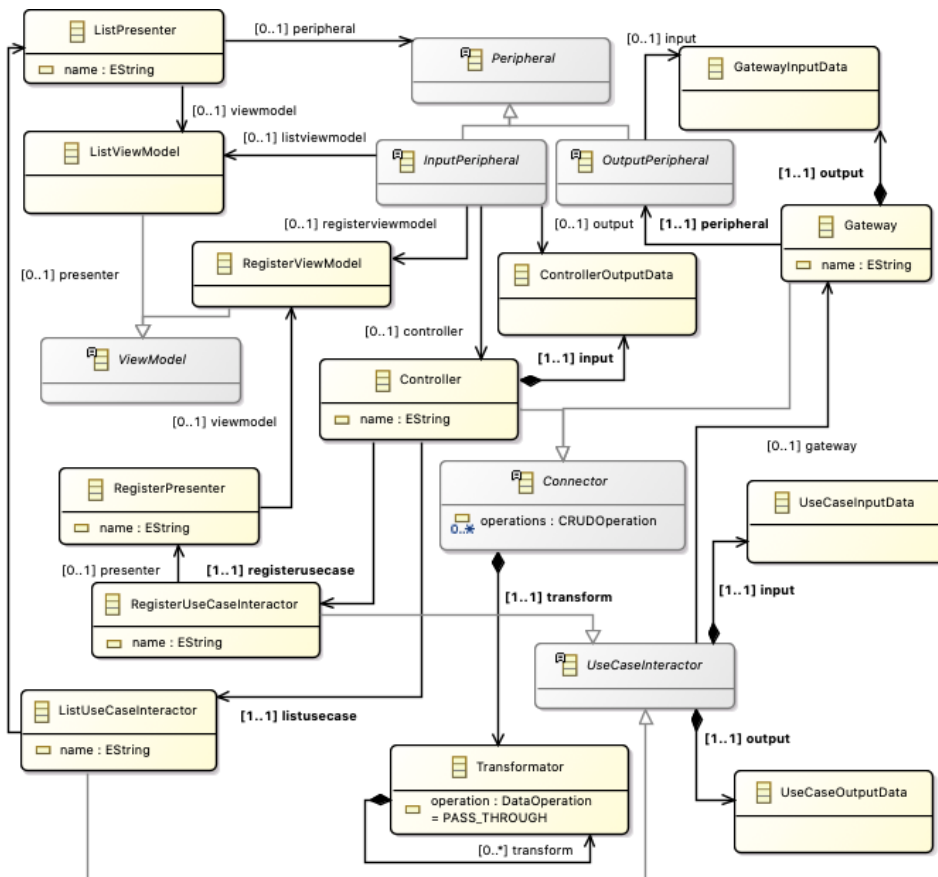
## Elementos del metamodelo

Los elementos del metamodelo de arquitectura pueden ser elementos destinados para cumplir con la arquitectura limpia o elementos específicos de la plataforma Android.

## Elementos de arquitectura limpia

Con el fin de cumplir con los principios de arquitectura limpia, el metamodelo de arquitectura incluye los siguientes elementos presentados en la figura 14.

**Figura 14.** Elementos que cumplen con la arquitectura limpia



Fuente: elaboración propia

- **UseCaseInteractor.** Representa una acción como `saveLocation`, `getPersons`. Este elemento tiene un nombre y contiene un `InputData` y un `OutputData`, los cuales heredan de `Entity` y son los objetos de transferencia de datos (*Data Transfer Object-DTO*), que serán transportados con los elementos de la capa de adaptadores de la arquitectura limpia. Este elemento es el núcleo de la arquitectura, ya que es el responsable de conectar la respectiva entrada con su respectivo disparador a la respectiva salida.
- **Controller.** Dado que los `InputPeripheral` traen información al sistema, estos actúan como los disparadores que le dirán al `UseCaseInteractor` que necesitan para trabajar. Este `Controller` es el puente entre el `InputPeripheral` de la capa externa en la arquitectura limpia con el `UseCase`; es decir, todos los `InputPeripheral` tendrán una referencia a un `Controller` que contiene los datos que recibirá del periférico. Dicho elemento se llama `Controller` `OutputData` que implementa el elemento abstracto `Entity` y actúa como DTO para este elemento. `Controller` hereda del elemento abstracto `Connector`.
- **Gateway.** De manera inversa al `Controller`, el elemento `Gateway` es referenciado desde el `UseCaseInteractor` como uno de sus puertos de salida, ya que este utiliza sus servicios para comunicarse con el `OutputPeripheral`. De nuevo, para enviar información de la capa exterior de la arquitectura limpia. De forma similar al `Controller`, el `Gateway` tiene su propio DTO llamado `GatewayInputData`. `Gateway` hereda del elemento abstracto `Connector`.
- **Presenter.** Este elemento se utiliza para el `UseCaseInteractor` como uno de sus puertos de salida, con el fin de lanzar una respuesta formateada para cualquier operación realizada. Normalmente, este elemento se utiliza para generar un `ViewModel` para la UI. En el presente trabajo, `Presenter` se usa exclusivamente para los elementos UI en cualquier otro elemento `InputPeripheral`. Este elemento no se crea, pero el metamodelo está abierto para cualquier extensión

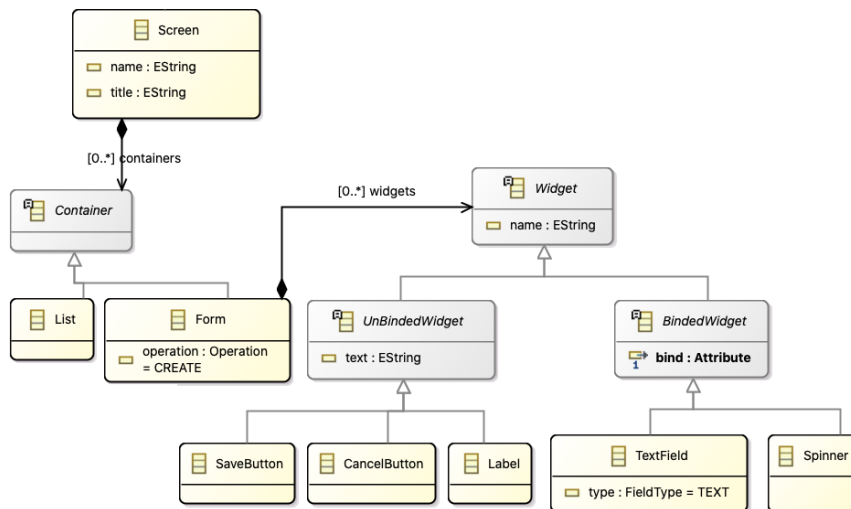
en la que otro periférico requiera una entrada formateada de una tarea completa. Su elemento DTO se llama `ViewModel`.

## Elementos específicos de la plataforma Android

La arquitectura de la plataforma Android define algunos elementos específicos que se deben tener en cuenta para crear aplicaciones. Los principales componentes son `Activity` y `Fragments`, que son las clases que representan el contrato entre el sistema operativo Android y la aplicación<sup>1</sup>.

En este trabajo, los elementos específicos de la plataforma Android se han clasificado en elementos para la creación de interfaces gráficas de usuarios y en elementos que son periféricos embebidos.

**Figura 15.** Elementos para la creación de interfaces gráficas



Fuente: elaboración propia

<sup>1</sup> <https://developer.android.com/topic/architecture>

- Elementos para la creación de interfaces gráficas. La figura 15 presenta un fragmento del metamodelo que incluye estos elementos para la creación de interfaces gráficas.
  - Screen. Tiene su propio elemento de menú y es navegable. Contiene un nombre y un título.
  - Container. Es un elemento que agrupa otros. Un Screen puede tener una lista de elementos abstractos Container. Existen dos tipos:
    - List. Representa una lista que muestra una colección de sus entidades vinculadas.
    - Form. Representa un formulario que realice cualquier operación de creación y edición sobre su entidad vinculada. El elemento Form tiene varios Widget que están vinculados a una de las propiedades de la entidad vinculada a su Container. Los Widget soportados son los siguientes:
      - ◇ SPINNER
      - ◇ TextField. Tiene un atributo que identifica el tipo de cuadro de texto. Las opciones son las siguientes:
        1. TEXT
        2. EMAIL
        3. PASSWORD
        4. PASSWORD\_NUMERIC
        5. PHONE
        6. POSTAL\_ADDRESS
        7. MULTILINE
        8. TIME
        9. DATE
        10. NUMBER

11. NUMBER\_SIGNED

12. NUMBER\_DECIMAL

Además, Form tiene los siguientes elementos no vinculados:

- ◇ SaveButton
- ◇ CancelButton
- ◇ Label

- Periféricos embebidos. Cuando la app dispara un periférico, se crea una solicitud que debe enviarse a una referencia de un Activity. Debido a esta situación, no es posible utilizar completamente la arquitectura limpia. En el metamodelo de arquitectura, este elemento se denomina `EmbeddedPeripheral`. Esta decisión no afecta la conexión de los periféricos en el metamodelo de dominio; sin embargo, sí afecta al metamodelo de arquitectura, el cual es dependiente de la plataforma Android y engloba el pensamiento de que este metamodelo de arquitectura es un PSM.

El `EmbeddedPeripheral` se conecta directamente desde un `InputPeripheral`, a través de un `EmbeddedConnector`. De esta forma, la tabla 3 presenta la lista de periféricos organizada por su distribución, teniendo en cuenta que algunos `InputPeripheral` se clasifican como `EmbeddedPeripheral`; por tanto, pueden estar inmersos en el flujo de arquitectura limpia para comunicarse con otros periféricos.

## Metamodelo de arquitectura completo

Finalmente, la figura 16 presenta el metamodelo completo a través de la perspectiva *ViewPoint* del diagrama de clases utilizando EMF. Este metamodelo contiene todos los elementos descritos anteriormente, los cuales permiten crear modelos de arquitectura con todos los elementos necesarios para modelar una aplicación completa para la plataforma Android, incluyendo no solo su estructura, sino además su comportamiento.

**Tabla 3.** Periféricos en metamodelo de arquitectura

| OutputPeripheral | InputPeripheral | EmbeddedPeripheral | InputPeripheral<br>EmbeddedPeripheral |
|------------------|-----------------|--------------------|---------------------------------------|
| RealTimeDatabase | NFCReader       | Vibrator           | Camera                                |
|                  | MotionSensor    | SMSSender          | GPS                                   |
|                  | PositionSensor  | PhoneCaller        | EnvironmentSensor                     |
|                  | Screen          | NFCWriter          |                                       |

**Fuente:** elaboración propia





## Capítulo 5. Transformación de modelo a modelo

Este capítulo presenta la primera transformación de la cadena de transformación de modelos. Esta corresponde a la transformación de modelo a modelo que, en este caso, transforma un modelo de dominio conforme con el metamodelo de dominio MPML a un modelo de arquitectura conforme con el metamodelo de arquitectura para plataforma Android. La transformación fue desarrollada usando el lenguaje ATL, el cual es presentado en la sección Atlas Transformation Language. La sección Reglas de transformación presenta las reglas aplicadas al modelo de dominio, las cuales están escritas en lenguaje ATL. Estas reglas de transformación pueden ser directas o por invocación. Finalmente, la sección Ejecución de la transformación presenta el proceso de ejecución de las reglas de transformación.

### Atlas Transformation Language

Atlas Transformation Language es un lenguaje de transformación de modelos en el contexto específico de MDE. Atlas Transformation Language (ATL) se utiliza para generar un modelo destino conforme con un metamodelo destino, con base en un modelo origen conforme con un metamodelo origen (figura 2). El metamodelo origen y el metamodelo destino deben ser conformes con el mismo metamodelo que en el caso de EMF es *Ecore*. ATL se creó como parte de EMF y se usa ampliamente para transformar modelos UML en otros modelos (Amrani *et al.*, 2015).

Debido a que ATL es un DSL, ATL cuenta con un metamodelo, el cual es conforme con el metametamodelo *Ecore*. Por tanto, un modelo de transformación conforme con ATL, corresponde a un conjunto de reglas que se ejecutan sobre el modelo origen para generar el modelo destino.

ATL tiene las siguientes características (Jouault *et al.*, 2008):

- Reglas reutilizables. ATL proporciona una biblioteca de reglas de transformación reutilizables que se pueden componer para realizar transformaciones más complejas.
- Depuración y trazabilidad. ATL proporciona funciones para depurar y rastrear las transformaciones, incluida la trazabilidad entre los modelos de origen y de destino.
- Validación de modelos. ATL incluye un mecanismo de validación de modelos para verificar la integridad de los modelos antes y después de la transformación.
- Flexibilidad. ATL es flexible y se puede ampliar con código Java personalizado para manejar requerimientos específicos de la transformación.
- Integración con otras herramientas. ATL se integra con otras herramientas de modelamiento como editores de UML. Además, se puede usar en combinación con otras técnicas de MDE.

En este proyecto de investigación, la transformación modelo a modelo usando ATL tiene como objetivo tomar un modelo de dominio conforme con el metamodelo de dominio MPML (figura 7) y generar un modelo de arquitectura conforme con el metamodelo de arquitectura que está alineado con la plataforma Android y que satisface los principios de arquitectura limpia (figura 16).

## Reglas de transformación

### Reglas de transformación directa

Las reglas de transformación directa permiten al desarrollador definir reglas de transformación de forma imperativa. Por tanto, este tipo de reglas

permiten hacer coincidir algunos elementos del modelo origen (es decir, el modelo de dominio MPML) y generar a partir de ellos un conjunto de elementos diferentes en el modelo destino (es decir, el modelo de arquitectura). ATL recorre automáticamente la definición del modelo y una vez que visita un elemento de cierto tipo, busca una regla de transformación directa definida para ese elemento y aplica la transformación. Por convención en ATL, el nombre de una regla se define como [nombre metamodelo origen]![elemento origen]2[nombre metamodelo destino]![elemento destino].

### Regla Domain!App2Architecture!App

Esta regla define el elemento fuente App del metamodelo Domain, que se define con la sintaxis Dominio!App dentro de un alias s. Del mismo modo, se define el elemento destino App del metamodelo Architecture, que se define con la sintaxis Architecture!App acompañado de las declaraciones de transformación de atributos entre paréntesis y la asignación directa de los elementos name, namespace y start utilizando el alias s definido anteriormente. El listado 5.1 presenta el código fuente de esta regla.

Listado 5.1: Domain!App2Architecture!App

```

1 rule App2App {
2   from
3     s: Domain!App
4   to
5     t: Architecture!App (
6       name <- s.name,
7       namespace <- s.namespace,
8       start <- s.start
9     )
10 }
```

### Regla Domain!Start2Architecture!Start

Esta regla define el elemento fuente Start del metamodelo Domain que se define con la sintaxis Dominio!Start dentro de un alias s. Del mismo modo, se define el elemento destino Start del metamodelo Architecture que se define con la sintaxis Architecture!Start acompañado de las declaraciones de transformación de atributos entre paréntesis y la asignación

directa del elemento `connect` utilizando el alias `s` definido anteriormente. El listado 5.2 presenta el código fuente de esta regla.

Listado 5.2: Regla `Domain!Start2Architecture!Start`

```

1 rule
2     Start2Start {
3         from
4         s: Domain!Start
5         to
6         t: Architecture!Start (
7             connect <- s.connect
8         )
9     }

```

### Regla `Domain!MPMLElement a Architecture!Peripheral`

Junto al elemento `Domain!Start`, es posible conectar un elemento `Domain!MPMLElement`, el cual es abstracto, del que heredan los elementos abstractos `Domain!MPMLInputComponent` y `Domain!MPMLOutputComponent`.

Normalmente, el modelador empieza conectando un elemento `MPMLInputElement`, pero depende del modelador si decide empezar conectando un `MPMLOutputComponent` en el caso en que el objetivo de la transformación dé como resultado un código de implementación parcial.

En el metamodelo de arquitectura está definida una estructura similar con el elemento `Architecture!Peripheral` del que heredan los elementos `Architecture!InputPeripheral` y `Architecture!OutputPeripheral`. Esta semejanza en la estructura nos permite definir reglas para estos elementos, en la que el modelo destino se genera aplicando los conceptos de arquitectura limpia.

Para cada `Architecture!InputPeripheral`, la transformación debe asignar los siguientes atributos:

- `controller`. Es la conexión a una instancia `Architecture!Controller`.
- `output`. Es el nuevo `Controller` que debe tener una entrada que corresponde al elemento `Architecture!ControllerOutputData`, que a su vez corresponde a la salida del elemento `Architecture!`

InputPeripheral, de acuerdo con los límites definidos por la arquitectura limpia.

- `viewModel`. Es el elemento de transferencia de datos entre el `Presenter` y el elemento `Architecture!InputPeripheral`, de acuerdo con los límites definidos por la arquitectura limpia.

El listado 5.3 presenta la implementación de la regla para el elemento `Domain!NFCReader` que es uno de los `Architecture!InputPeripherals`.

Listado 5.3: Regla `Domain!NFCReader2Architecture!NFCReader`

```

1 rule NFCReader2NFCReader {
2   from
3     s: Domain!NFCReader
4   to
5     t: Architecture!NFCReader (
6       controller <- thisModule.createControllerFromDataConnector(s),
7       output <- thisModule.createControllerOutputData(s.model),
8       viewModel <- thisModule.createViewModel(s.model)
9     )
10 }
```

En las líneas 6, 7 y 8 se está haciendo una llamada a algunas funciones con la palabra reservada `thisModule`. Entonces, esta regla introduce las siguientes reglas de invocación de la transformación:

- `createControllerFromDataConnector`
- `createControllerOutputData`
- `createViewModel`.

## Reglas de transformación por invocación

Este tipo de regla debe ser llamada explícitamente desde un bloque imperativo ATL. Hay dos tipos de reglas de transformación por invocación:

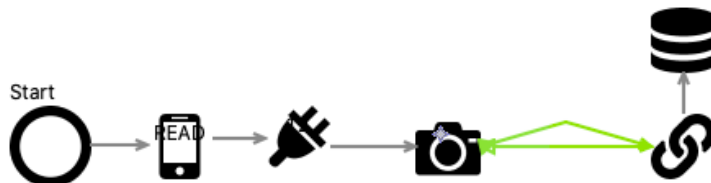
- *Lazy rule*. Esta regla es invocada por una regla imperativa para ser ejecutada de manera explícita y genera un elemento objetivo por cada invocación.
- *Unique lazy rule*. Esta regla es invocada por una regla imperativa para ser ejecutada de manera explícita y genera un único

elemento objetivo independientemente de que reciba múltiples invocaciones.

### Regla createControllerFromDataConnector

El objetivo de esta regla es crear un nuevo elemento destino Architecture!Controller. Para realizar esta transformación, la regla lee el modelo y obtiene la instancia asociada al elemento origen Domain!MPMLComponent, obtiene el nombre de ese modelo (que es una instancia Domain!Entity) y concatena la cadena Controller para asignar el atributo name del Controller.

Figura 17. Caso de estudio de cámara



Fuente: elaboración propia

Por ejemplo, la figura 17 presenta un caso de estudio en el que la aplicación representada por el dominio utiliza varios periféricos que se tratan con la misma entidad. Este caso presenta un modelo de dominio, en el que el Domain!Camera toma una fotografía y, a continuación, la almacena en una base de datos en tiempo real a través del elemento Domain!RealTimeDatabase. En este caso, hay dos interacciones con el mismo modelo que comprende la foto tomada. Si se hubiera utilizado una *lazy rule* entonces habría creado dos instancias del controlador. En consecuencia, la regla es creada como *unique lazy rule* con el fin de crear solo una instancia del controlador, incluso si la regla se invoca múltiples veces. El listado 5.4 presenta la implementación de la regla createControllerFromDataConnector.

#### Listado 5.4: Regla createControllerFromDataConnector

```
1 unique lazy rule createControllerFromDataConnector {
2   from
3     s : Domain!MPMLComponent
4   to
5     sm : Architecture!Controller (
6       input <- thisModule.createControllerOutputData(s.model),
```

```

7 |     name <- s.model.name.toString() + "Controller",
8 |     usecase <- thisModule.createUseCaseInteractorFromDataConnector(s),
9 |     transform <- thisModule.createDefaultTransformator(
10 |         s.getDataConnector())
11 | )
12 | }

```

La regla `createControllerFromDataConnector` invoca las siguientes tres reglas que también son *unique lazy rule* y funcionan de la misma manera.

- `createControllerOutputData`
- `createUseCaseInteractorFromDataConnector`
- `createDefaultTransformator`

El caso de estudio presentado en la figura 17 representa los siguientes elementos del modelo de dominio:

1. `Domain!Start` conectado a `Domain!UICreateForm`.
2. `Domain!UICreateForm` conectado a `Domain!FlowConnector`.
3. `Domain!FlowConnector` conectado a `Domain!Camera`. Este elemento se utiliza para conectar un `Domain!MPMLInputComponent` con un `Domain!MPMLOutputComponent`.
4. `Domain!Camera` conectado a `Domain!DataConnector`. Este elemento tiene un modelo llamado `CameraModel` que tiene un atributo llamado `photo` que es de tipo `BITMAP` y tiene un validador donde `isRequired = True`.
5. `Domain!DataConnector` conectado al elemento `Domain!RealTimeDatabase`. Este elemento tiene un `Transformator` con un `Operation = PASS_THROUGH`. No es necesario asignar un modelo a `Domain!RealTimeDatabase`, porque este lo lleva desde el elemento de entrada conectado inmediatamente antes del `Domain!DataConnector` conectado a él.

### Regla `createControllerFromFlowConnector`

Desde el punto de vista de la arquitectura limpia aplicada al metamodelo de arquitectura, la transformación toma el elemento `Domain!FlowConnector`

y da como resultado la creación del elemento `Architecture!Controller` usando la regla presentada en el listado 5.5.

#### Listado 5.5: Regla `createControllerFromFlowConnector`

```
1 unique lazy rule createControllerFromFlowConnector {
2   from
3     sUI : Domain!MPMLInputComponent
4   to
5     sm : Architecture!Controller (
6       input <- thisModule.createControllerOutputData(sUI.model),
7       name <- sUI.model.name.toString() + "Controller",
8       transform <- thisModule.createDefaultTransformator(sUI.connect),
9       usecase <- thisModule.createUseCaseInteractorFromFlowConnector(sUI)
10    )
11 }
```

Esta regla crea los cuatro atributos para el elemento `Controller`:

- `input`: es un elemento `Architecture!ControllerOutputData` que hereda del elemento `Architecture!Entity`.
- `name`: es la concatenación del nombre del modelo con el sufijo `Controller`.
- `transform`: se crea usando la regla `createDefaultTransformator`.
- `usecase`: es el siguiente elemento de la arquitectura limpia.

#### Regla `createDefaultTransformator`

Esta regla crea un elemento `Transformator`, la cual es presentada en el listado 5.6. Esta regla es invocada por la regla `createControllerFromFlowConnector`.

#### Listado 5.6: Regla `createDefaultTransformator`

```
1 unique lazy rule createDefaultTransformator {
2   from
3     s : Domain!Connector
4   to
5     t : Architecture!Transformator (
6       operation <- s.operation
7     )
8 }
```

## Regla createUseCaseInteractorFromFlowConnector

A partir del elemento `Domain!FlowConnector`, utilizando la regla presentada en el listado 5.7, se crea el elemento `Architecture!UseCaseInteractor`.

Listado 5.7: Regla createUseCaseInteractorFromFlowConnector

```

1 unique lazy rule createUseCaseInteractorFromFlowConnector {
2   from
3     sIn : Domain!MPMLInputComponent
4   to
5     t : Architecture!UseCaseInteractor (
6       gateway <- Set{sIn.connect} -> select(con |
7         con.oclIsUndefined() = false) -> collect(con |
8           thisModule.createGateWayFromFlowConnector(con)) -> first(),
9       input <- thisModule.createUseCaseInputData(sIn.model),
10      name <- sIn.model.name.toString() + "UseCase",
11      output <- thisModule.createUseCaseOutputData(sIn.model),
12      presenter <- thisModule.createPresenter(sIn)
13    )
14 }
```

La creación del elemento `Architecture!UseCaseInteractor` implica varios atributos. Los atributos `input`, `name`, `output` y `presenter` se obtienen invocando la regla correspondiente que obtiene el elemento deseado.

El elemento `Domain!MPMLInputComponent` se puede conectar a dos tipos de elementos diferentes: el primero es un `Domain!FlowConnector` utilizando el atributo `connect` y el segundo un `Domain!DataConnector` utilizando el atributo `dataStore`.

En este caso, es posible que el atributo `connect` no se haya establecido en el elemento. Con el fin de realizar una doble comprobación de validación para evitar aplicar la regla para crear un gateway basado en una instancia `connect` con valor `NULL`, se comprueba previamente que la instancia `connect` ya existe. Por tanto, se realizan los siguientes pasos:

- Se crea un Set de un elemento que es el atributo `connect` de la instancia del elemento `Domain!MPMLInputComponent`.
- Mediante una expresión lambda se confirma que el elemento existe mediante la función `oclIsUndefined()`.

- Mediante la función `select()` se filtra el conjunto creado anteriormente.
- Se utiliza la función `collect()` y de nuevo con una expresión lambda se invoca la regla `createGateWayFromFlowConnector`.
- Con la función `first()` se obtiene el primer elemento, el cual es el único posible.

### Regla `createDataSourceNFCWriter`

La regla `createDataSourceNFCWriter` hace parte de un conjunto de reglas basadas en `InputPeripheral`. El listado 5.8 presenta la regla `createDataSourceNFCWriter`.

Listado 5.8: Regla `createDataSourceNFCWriter`

```

1 lazy rule createDataSourceNFCWriter
2 {
3   s: Domain!MPMLInputComponent
4   to
5     t: Architecture!NFCWriter (
6       inputModel <- thisModule.createViewModel(s.model)
7     )
8 }
```

Esta regla recibe la instancia abstracta `Domain!MPMLInputComponent` como entrada en lugar de la instancia concreta `Domain!NFCReader`. En este caso, se puede aplicar a esta regla la expresión lambda presentada en el listado 5.9.

Listado 5.9: Expresión lambda para la regla `createDataSourceNFCWriter`

```

1 Set{s} -> select(
2   p | p.getDataConnector().connect.oclIsTypeOf(Domain!NFCWriter)
3 ) -> collect(
4   p | thisModule.createDataSourceNFCWriter(p)
5 ) -> first()
```

Sin embargo, cada transformación de `Domain!MPMLElement` debe ser tratada de forma individual, ya que el metamodelo de arquitectura introduce un nuevo tipo de periférico específico de la plataforma llamado periférico

embebido. En este caso, el `inputModel` se delega a la regla que transforma un elemento `Domain!Entity` en un elemento `Architecture!Entity`.

## Regla `Domain!Entity2Architecture!Entity`

**Tabla 4.** Reglas de mapeo de `Domain!DataType` a `Architecture!DataType`

| <code>Architecture!DataType</code> | <code>Domain !DataType</code> |
|------------------------------------|-------------------------------|
| STRING                             | STRING                        |
| STRING                             | LONG_STRING                   |
| STRING                             | EMAIL                         |
| STRING                             | PASSWORD                      |
| STRING                             | PASSWORD_NUMERIC              |
| STRING                             | PHONE                         |
| STRING                             | POSTAL_ADDRESS                |
| STRING                             | TIME                          |
| INT                                | INTEGER_NUMBER                |
| INT                                | SIGNED_NUMBER                 |
| DATE                               | DATE                          |
| DOUBLE                             | DECIMAL_NUMBER                |
| BITMAP                             | BITMAP                        |

**Fuente:** elaboración propia

Esta transformación requiere que cada uno de los `DataType` relacionados con sus atributos se debe mapear siguiendo las reglas de mapeo que se muestran en la tabla 4, dado que el metamodelo de arquitectura trata de acercarse más a la implementación real y debe representar tipos de datos específicos para la plataforma Android.

La transformación de los tipos de datos se lleva a cabo con la regla presentada en el listado 5.10.

Listado 5.10: Regla `getDataTypeString`

```

1 helper context Domain!Attribute def: getDataTypeString():
    String =
2   Set{self.type} -> select(t | t.toString() = "STRING" or t.toString() =
3   "LONG_STRING" or t.toString() = "EMAIL" or t.toString() = "PASSWORD" or
4   t.toString() = "PASSWORD_NUMERIC" or t.toString() = "PHONE" or
5   t.toString() = "POSTAL_ADDRESS" or t.toString() = "TIME") ->
6   collect(t | "STRING") -> union(Set{self.type} -> select(t |
7   t.toString() = "DATE") -> collect(t | t.toString())) -> union(
8   Set{self.type} -> select(t | t.toString() = "INTEGER_NUMBER" or
9   t.toString() = "SIGNED_NUMBER") -> collect(t | "INT")
10  ) -> union(
11   Set{self.type} -> select(t | t.toString() = "DECIMAL_NUMBER") ->
12   collect(t | "DOUBLE")
13  ) -> union(
14   Set{self.type} -> select(t | t.toString() = "BITMAP") -> collect(t |
15   "BITMAP")
16  ) -> first();

```

### Regla `Domain!MPMLInputComponent2Architecture!InputPeripheral`

Normalmente, los atributos intrínsecos de los periféricos se mapean directamente desde el elemento del dominio a su correspondiente elemento de arquitectura. Además, con base en la entidad, se crean las reglas de transformación reescribiendo los nombres para generar el resto de los elementos en la arquitectura limpia cuando sea necesario. Por ejemplo, un `MPMLInputElement` usa un `DataConnector` para conectarse a una base de datos en tiempo real y también tiene una entidad llamada `Person`. Entonces, se generan los siguientes elementos de arquitectura limpia:

- `Controller:{name:"PersonController"}`  
Con su correspondiente  
`ControllerOutputData:{name:"Person"}`
- `UseCaseInteractor:{name:"PersonUseCase"}`  
Con su correspondiente  
`UseCaseInputData:{name:"Person"}`  
`UseCaseOutputData:{name:"Person"}`
- `Gateway:{name:"personGateway"}`  
Con su correspondiente  
`GatewayInputData:{name:"Person"}`

En el caso de que cualquier otro elemento que requiera el presentador que hasta ahora solo lo requiere los elementos de UI, se genera:

- `Presenter:{name:"PersonPresenter"}`  
Con su correspondiente  
`ViewModel:{name:"Person"}`

### Regla `Domain!CreateForm2Domain!Screen`

El elemento `Screen` tiene su propio conjunto de elementos en el meta-modelo de arquitectura. Por tanto, requiere un tratamiento especial, diferente al aplicado para el resto de los periféricos. Esto requiere la creación de `Architecture!Screen`. Hay dos formas de lograr la creación de este elemento. La primera es desde `Domain!UICreateForm`, y se usa la regla presentada en el listado 5.11 ; la segunda es de `Domain!UIList`, y se usa la regla presentada en el listado 5.12. En ambos casos se crea un elemento `Architecture!Screen`. Este elemento contendrá un `Container` de tipo `Form` que, a su vez, incluirá una colección de `Widget` que se creará de acuerdo con el tipo de datos de cada atributo en su entidad relacionada, siguiendo las reglas de mapeo que se muestran en la tabla 5.

Listado 5.11: Regla `UICreateForm2ScreenForm`

```
1 rule UICreateForm2ScreenForm {  
2   from  
3     sUI : Domain!UICreateForm  
4   to
```

```

5     screen: Architecture!Screen (
6         containers <- Set{thisModule.createForm(sUI)},
7         -- ***** Clean Architecture flow *****
8         controller <- Set{sUI} -> select(f | f.dataStore.oclIsUndefined() =
9         false)->select(f | f.dataStore.connect.oclIsTypeOf
10        (Domain!RealTimeDatabase)) -> collect(f |
11        thisModule.createControllerFromDataConnector(f)) -> first(),
12        output <- Set{sUI} -> select(f | f.dataStore.oclIsUndefined() =
13        false) -> select(f | f.dataStore.connect.oclIsTypeOf
14        (Domain!RealTimeDatabase)) -> collect(f |
15        thisModule.createControllerOutputData(sUI.model)) -> first(),
16        -- ***** Embedded Peripherals *****
17        embeddedconnector <- Set{sUI} -> select(f |
18        f.dataStore.oclIsUndefined() = false) -> collect(p | thisModule.
19        createEmbeddedConnectorFromDataConnector(p)) -> union(
20        Set{sUI} -> select(f | f.connect.oclIsUndefined() = false) ->
21        collect(p | thisModule.createEmbeddedConnectorFromFlowConnector(p))
22        ) -> first(),
23        Set{sUI} -> select(f | f.dataStore.oclIsUndefined() = false) ->
24        select(f | f.dataStore.connect.oclIsTypeOf (Domain!NFCWriter)) ->
25        collect(f | thisModule.createEmbeddedConnectorFromDataConnector(f))
26        -> first(),
27        or f.connect.oclIsTypeOf(Domain!PhoneCaller)
28        name <- Set{sUI} -> select(s | s.model.oclIsUndefined() = false) ->
29        collect(s | s.getOperation().toString().toLowerCase() +
30        s.model.name.toLowerCase()) -> union(
31        Set{sUI}->select(s | s.model.oclIsUndefined() = true)->collect(s |
32        "Welcome")
33        ) -> first(),
34        title <- Set{sUI} -> select(s | s.model.oclIsUndefined() = false)
35        -> collect(s | s.getOperation().toString().toLowerCase() + " " +
36        s.model.name.toLowerCase()) -> union(
37        Set{sUI} -> select(s | s.model.oclIsUndefined() = false) ->
38        collect(s | "Welcome")) -> first(),
39        viewmodel <- Set{sUI} -> select(s | s.model.oclIsUndefined() =
40        false) -> collect(s | thisModule.createViewModel(s.model)) ->
41        first()
42    )
43 }

```

### Listado 5.12: Regla UIList2ScreenList

```

1 rule UIList2ScreenList {
2     from
3     sUI: Domain!UIList
4     to
5     screen: Architecture!Screen (
6         containers <- Set{thisModule.createList(sUI)},
7         controller <- thisModule.createControllerFromDataConnector(sUI),
8         -- input <-
9         name <- "read" + sUI.model.name.toLowerCase(),
10        -- output <-

```

```

11 |         title <- "read" + sUI.model.name.toLower(),
12 |         viewmodel <- thisModule.createViewModel(sUI.model)
13 |     )
14 | }

```

**Tabla 5.** Reglas de mapeo de Domain!DataType a Architecture!FieldType.

| Architecture!FieldType | Domain!DataType  |
|------------------------|------------------|
| TEXT                   | STRING           |
| MULTILINE              | LONG_STRING      |
| EMAIL                  | EMAIL            |
| PASSWORD               | PASSWORD         |
| PASSWORD_NUMERIC       | PASSWORD_NUMERIC |
| PHONE                  | PHONE            |
| POSTAL_ADDRESS         | POSTAL_ADDRESS   |
| TIME                   | TIME             |
| DATE                   | DATE             |
| NUMBER                 | INTEGER_NUMBER   |
| NUMBER_DECIMAL         | DECIMAL_NUMBER   |
| NUMBER_SIGNED          | SIGNED_NUMBER    |

**Fuente:** elaboración propia

## Regla Domain!OutputComponent2Architecture! OutputPeripheral

En arquitectura limpia estos elementos no disparan ningún evento a los `UseCaseInteractor`, sino que solo se encargan de recibir una solicitud y emitir una respuesta en algunos casos particulares. Por tanto, estos elementos solo se asignan a su correspondiente elemento `Architecture!OutputPeripheral` en el metamodelo de arquitectura.

## Regla Domain!UIList2Architecture!Screen

Este es un caso especial, porque el elemento `MPMLOutputElement` puede tener adjunto un `DataSource` que puede ser un `RealTimeDatabase`. Dados estos requerimientos híbridos, la regla genera todos los elementos de arquitectura limpia como un `InputPeripheral`, pero también incluye la generación de un `Presenter` con su `ViewModel`. Finalmente, en términos del propio `Peripheral`, genera un `Screen` con una lista de tipo `Container` ligada a su `ViewModel` correspondiente, pero sin ningún `Widget`.

## Ejecución de la transformación

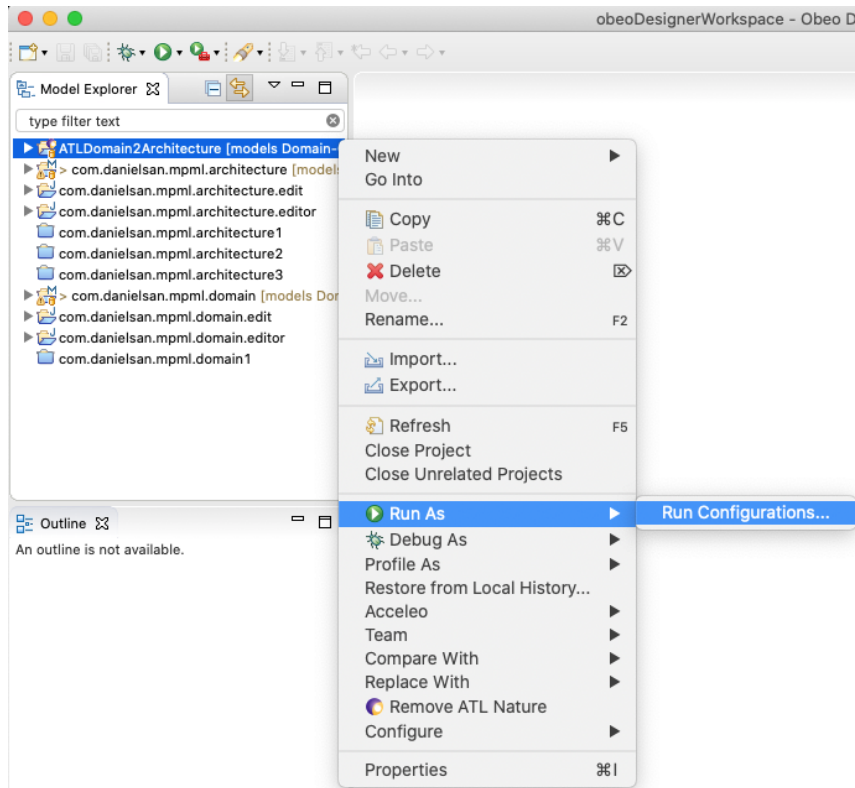
La ejecución de la transformación se lleva a cabo siguiendo los siguientes pasos, usando la herramienta `Obeo Designer`<sup>1</sup> que es parte del proyecto `Eclipse`:

1. En la vista *Model Explorer* se selecciona el proyecto *ATLDomain2Architecture*, se presiona clic derecho y se selecciona *Run As -> Run Configurations* como se presenta en la figura 18.
2. En la ventana de diálogo *Run Configurations* se selecciona *ATL Transformation* y se presiona clic en el ícono *New* señalado en la figura 19.
3. En la ventana de diálogo *Run Configurations*, en el panel *ATL Module*, se presiona clic en la opción *Workspace* y se selecciona el archivo *Domain2Architecture.alt*.

---

<sup>1</sup> <https://www.obeodesigner.com/>

**Figura 18.** Configuración de transformación ATL

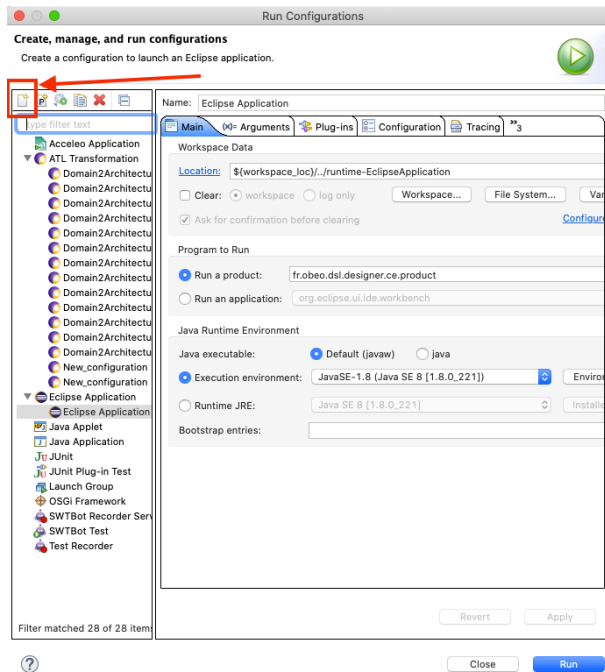


**Fuente:** elaboración propia

4. En la ventana de diálogo *Run Configurations*, en el panel *Metamodels*, en el campo *Domain* se selecciona el metamodelo de dominio MPML. Asimismo, en el campo *Architecture* se selecciona el metamodelo de arquitectura. Para acceder a estos metamodelos hay tres opciones. La primera es buscar en el espacio de trabajo configurado para *Obeo Designer* mediante el botón *Workspace*. La segunda es buscar el archivo del metamodelo en el dispositivo mediante el botón *File System*. La tercera es seleccionar el registro del metamodelo en EMF mediante el botón *EMF Registry*.
5. En la ventana de diálogo *Run Configurations*, en el panel *Source Models*, en el campo *IN* se selecciona el modelo de dominio. Asimismo, en

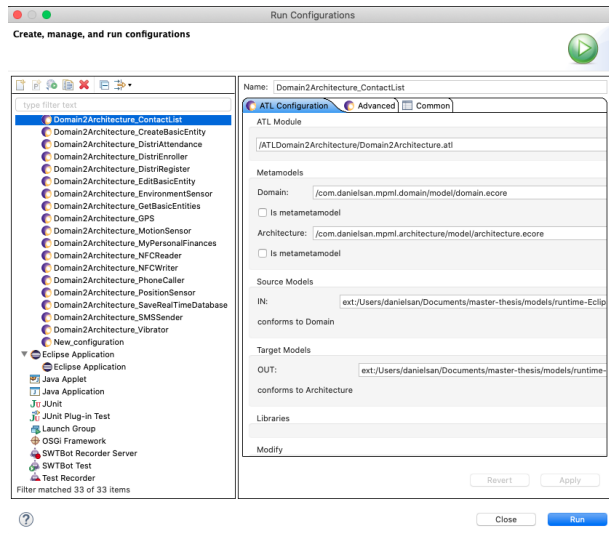
el panel *Target Models*, en el campo *OUT* se selecciona el modelo de arquitectura. Finalmente, se ejecuta la transformación presionando clic en el botón *Run*. La figura 20 presenta la ventana de diálogo *Run Configurations* con todos los campos diligenciados.

**Figura 19.** Ventana de configuración de transformación ATL



Fuente: elaboración propia

Figura 20. Ventana de configuración final de transformación ATL



Fuente: elaboración propia



## Capítulo 6. Transformación de modelo a texto

Este capítulo presenta la segunda transformación de la cadena de transformación de modelos. Esta corresponde a la transformación de modelo a texto que, en este caso, transforma un modelo de arquitectura conforme con el metamodelo de arquitectura alineado con arquitectura limpia para la plataforma Android a código fuente, en lenguaje de programación Kotlin y archivos de configuración XML. La transformación fue desarrollada usando el lenguaje de transformación Acceleo, el cual es presentado en la sección Acceleo. Además, la sección Kotlin describe los conceptos principales del lenguaje de programación Kotlin. La sección Módulos presenta los módulos generados mediante la transformación de modelo a texto para la aplicación final en Android. Además, la sección Estructura de los elementos Android generados muestra cómo estos pueden ser estáticos y dinámicos. Finalmente, la sección Desarrollo del proyecto Acceleo presenta el proceso de desarrollo con todos los elementos relacionados en la transformación modelo a texto.

### Acceleo

Acceleo es un motor de generación de código que permite generar artefactos de texto a partir de un modelo origen conforme con un metamodelo origen. Acceleo está escrito en el lenguaje de programación Java y utiliza EMF como base para la gestión de modelos; por tanto, Acceleo recibe como entrada un modelo que es conforme con un metamodelo. Acceleo

permite definir plantillas que especifican el texto que se desea generar y usar elementos del modelo como variables en estas plantillas (Benouda *et al.*, 2017).

Acceleo se utiliza para generar código fuente, documentación, reportes o cualquier otro tipo de artefacto basado en texto a partir de un modelo. Acceleo es una herramienta poderosa para reducir el esfuerzo manual y aumentar la productividad en el desarrollo basado en modelos, debido a que permite automatizar la creación de artefactos de texto reduciendo el tiempo, esfuerzo y errores de la codificación manual (Di Sipio *et al.*, 2021).

Acceleo tiene las siguientes características que lo convierten en un potente motor de generación de código (Kim *et al.*, 2011):

- Enfoque basado en modelos que genera artefactos de texto basados en los elementos de un modelo.
- Generación basada en plantillas para especificar el texto que genera. Las plantillas están escritas en una sintaxis simple e intuitiva y pueden incluir variables que hacen referencia a elementos del modelo.
- Extensibilidad que permite crear extensiones personalizadas para realizar tareas que el motor no admite de forma nativa.
- Código reutilizable mediante plantillas que reduce la necesidad de codificación manual y aumenta la consistencia de los artefactos generados.
- Integración con Eclipse, lo que facilita su uso y la integración en su proceso de desarrollo.
- Compatibilidad con múltiples formatos de salida.
- Alto rendimiento que le permite generar grandes cantidades de código de forma rápida y eficiente.
- Depuración que facilita el diagnóstico y la solución de problemas con el código generado.

- Informes de errores personalizables, lo que facilita la identificación y solución de problemas con plantillas o modelos.

## Kotlin

Kotlin es un lenguaje de programación de propósito general, estáticamente tipado que es conciso, expresivo y seguro. Está desarrollado por JetBrains y se lanzó por primera vez en el año 2016. Kotlin está diseñado para ser totalmente compatible con Java y puede ejecutarse en la máquina virtual de Java (Jemerov y Isakova, 2017).

Kotlin es conocido por su inferencia de tipo mejorada, seguridad de apuntador nulo y funciones de extensión que facilitan la escritura de código más legible y fácil de mantener. También ofrece funciones modernas, como alias de tipo, funciones de orden superior y corrutinas que simplifican las operaciones complejas y facilitan la escritura de código asíncrono y paralelo (Nozik, 2019).

Kotlin se usa ampliamente para desarrollar aplicaciones de Android, aplicaciones del lado del servidor, entre otros tipos de *software*. También es compatible con muchas herramientas y plataformas de desarrollo, incluidas Android Studio, IntelliJ IDEA y Spring. En general, Kotlin es un lenguaje de programación, el cual es considerado versátil y pragmático que ha ganado popularidad por su simplicidad, legibilidad y rendimiento (Moskala y Wojda, 2017). Algunas características relevantes que tiene el lenguaje de programación Kotlin son (Gotseva *et al.*, 2019):

- Interoperabilidad con Java. Kotlin es completamente interoperable con Java. Esto significa que es posible usar las librerías y herramientas de Java con Kotlin y también es posible invocar código de Kotlin desde Java.
- Seguridad de apuntador nulo. Kotlin tiene características de seguridad integradas en el lenguaje que ayuda a evitar excepciones de apuntador nulo en tiempo de compilación que hace que su código sea más seguro y confiable.

- **Expresividad.** Kotlin está diseñado para ser conciso y expresivo, lo cual facilita la escritura de código claro y legible.
- **Funciones de orden superior.** Kotlin admite funciones de orden superior que permiten enviar funciones como parámetros de otras funciones y retornar funciones como valores.
- **Corrutinas.** Kotlin admite corrutinas que facilitan la escritura de código asíncrono y sin bloqueo, sin la complejidad de los modelos tradicionales de subprocesos.
- **Funciones de extensión.** Kotlin admite funciones de extensión que permiten agregar nuevas funciones a las clases existentes sin tener que heredar de ellas o usar patrones de diseño como *Decorator*.
- **Clases de datos.** Kotlin proporciona clases de datos que son diseñadas para contener datos y tienen métodos como `equals`, `hashCode` y `toString` generados automáticamente.
- **Inferencia de tipos.** Kotlin ha mejorado la inferencia de tipos que permite escribir código más conciso y legible.
- **Propiedades y campos.** Kotlin proporciona propiedades y campos que permiten declarar variables y controlar su acceso de forma concisa y flexible.
- **Clases selladas.** Kotlin proporciona clases selladas que permiten restringir los posibles subtipos de una clase, facilitando la escritura de código más seguro y fácil de mantener.

## Módulos

Con base en el modelo de arquitectura generado después de la transformación de modelo a modelo, no es posible capturar algunas dimensiones del modelo de negocio, por ejemplo, la separación de componentes para cumplir con el principio de responsabilidad única (Holderer, 2022) (debido a la falta de un actor modelado en cualquiera de los metamodelos). Como resultado, los componentes están siendo separados por los límites definidos en una arquitectura limpia. En este caso, la representación física más

simple de un límite arquitectónico consistiría en un módulo de biblioteca de Kotlin. Esto nos ayuda, en la arquitectura, a restringir la dependencia de referencias, que es inversa al flujo de datos (Martin, 2018) dando como resultado los siguientes módulos generados:

- **App.** El módulo por defecto Android App contendrá todo el código relacionado con el dispositivo dividido en:
  - Código UI que tiene las definiciones de layout, color, estilos, strings, etc.
  - Código de periféricos que tiene los periféricos modelados, incluyendo el código `RealTimeDatabase`.

Para hacer una subseparación de responsabilidad, se genera un nuevo directorio que contiene el código que trabajará con cualquiera de los periféricos, excepto en los casos de algunos periféricos embebidos como el GPS y el vibrador.

- **Adapters.** Este módulo de librería Kotlin incluye el código relacionado con los adaptadores de los UseCase a los periféricos del framework. El módulo contiene los directorios para Controller, Gateway y Presenter.
- **UseCases.** Este módulo contiene en su directorio raíz la definición de UseCase.
- **Entities.** Este módulo contiene una aproximación de las entidades y sus reglas de negocio. Como consecuencia, las entidades contienen exclusivamente la validación de su tipo de dato, por lo que requiere longitudes de valor y tamaño para sus atributos. En la aplicación generada, si alguna de estas reglas se cumple, entonces la aplicación no continuará con el modelo de comunicación a todos de los periféricos y lanzará una excepción.

## Estructura de los elementos Android generados

La transformación de modelo a texto genera un conjunto de archivos y directorios que siguen una estructura de acuerdo con la plataforma Android. Esta estructura se divide en estática y dinámica.

### Estructura estática

La estructura estática consiste en el conjunto de componentes de utilidad que tiene una aplicación Android y valida la existencia de Screen y Controller para generar los archivos. Estos componentes se distribuyen de la siguiente forma:

- Componentes del *framework*.
  - Si el modelo contiene Screen, entonces la transformación genera algunos campos de extensión que se utilizan para los componentes UI.
  - El proyecto Android incluye los siguientes recursos:
    - `colors.xml` que incluye los colores en formato RGB de la aplicación.
    - `styles.xml` que define estilos para la aplicación.
    - Si hay Screen en el modelo, entonces crea todos los elementos relacionados con un elemento `NavigationDrawer`, con el fin de crear solo una `Activity` y varios `Fragment`, uno por cada Screen cuando sea necesario. Esto incluye los siguientes archivos:
      - ◇ `dimens.xml`
      - ◇ `mainLayouts`
      - ◇ `appbarLayouts`
      - ◇ `NavHeader`
      - ◇ `sideNavbar`
      - ◇ `navgraph`

◇ `ActivityMainDrawerMenu`

- `strings.xml` que ofrece internacionalización a la aplicación. El proyecto Android incluye todos los textos en archivos `string` en inglés. Si el desarrollador necesita añadir otros idiomas, entonces solo tiene que añadir los respectivos `strings.xml` con las respectivas claves traducidas.
- Configuración. Esto incluye el `Manifest.xml`, el proyecto Gradle y los archivos del módulo Gradle app.
- En caso de que se incluyan controladores, entonces se genera el archivo Gradle y la clase `Factory` para los módulos `Adapters`, `UseCases` y `Entities`. La clase `Factory` crea las clases concretas que implementan las interfaces que se utilizan como contratos para comunicarse con otros componentes.

## Estructura dinámica

La estructura dinámica sigue los `Peripheral` conectados desde el elemento `Start` en el modelo de arquitectura. Además, como estos elementos están conectados por el atributo `connected`, entonces existe un camino directo a seguir. La única situación bifurcada es cuando el mismo `Peripheral` tiene un `DataSource`, `DataStore` y un `EmbeddedConnector`, entonces se priorizan en el siguiente orden:

- `InputPeripheral`. Dependiendo del tipo de `InputPeripheral`, se generan los archivos indicados en la tabla 6, así como las clases `Controller` o `EmbeddedController` según el caso.
- `Controller`. Cada elemento `Controller` genera las siguientes clases:
  - `Controller`. Clase concreta `Controller`.
  - `IController`. Contrato de entrada para el componente `Framework`.
  - `DS`. Estructura de datos que se transportará desde el componente `Adapter`.

**Tabla 6.** Tabla de generación del InputPeripheral.

| Architecture!InputPeripheral | Archivos generados                                                           |
|------------------------------|------------------------------------------------------------------------------|
| Screen                       | ViewModel, Fragment                                                          |
| NFCReader                    | NFCReader,<br>/nfc_tech_filter.xml,<br>listado de las tecnologías soportadas |
| MotionSensor                 | Activity                                                                     |
| PositionSensor               | Activity                                                                     |
| EnvironmentSensor            | Activity                                                                     |

Fuente: elaboración propia

- **UseCaseInteractor.** Cada elemento UseCaseInteractor genera las siguientes clases:
  - UseCase. Clase concreta UseCase.
  - IUseCases. Contrato de entrada para el componente Adapter.
  - Contrato de salida para el componente adaptadores IGateway, IPresenter.
  - Presenter. Si se cumplen los requisitos, también se genera la clase concreta Presenter en el componente de Adapter.
  - DTO. La estructura de datos debe transportarse desde el componente UseCase.
- **Gateway.** Cada Gateway que se conecta a una RealTimeDatabase genera las siguientes clases:
  - Gateway. Clase concreta Gateway.
  - IDAO. Genera el contrato IDAO de salida que va a utilizar el Gateway.

- `OutputPeripheral`. Si el `OutputPeripheral` es de tipo `RealTimeDatabase`, entonces se crea la clase concreta DAO en el componente `Framework`.

Para el código generado, se utiliza el patrón *Table Data Gateway*, para acceder a los datos de Firebase (Fowler, 2012). Esto es adecuado en este caso porque ayuda con la desnormalización de los datos, ya que cada Gateway va a tratar las consultas de cada sección desnormalizada en la base de datos en tiempo real.

## Desarrollo del proyecto Acceleo

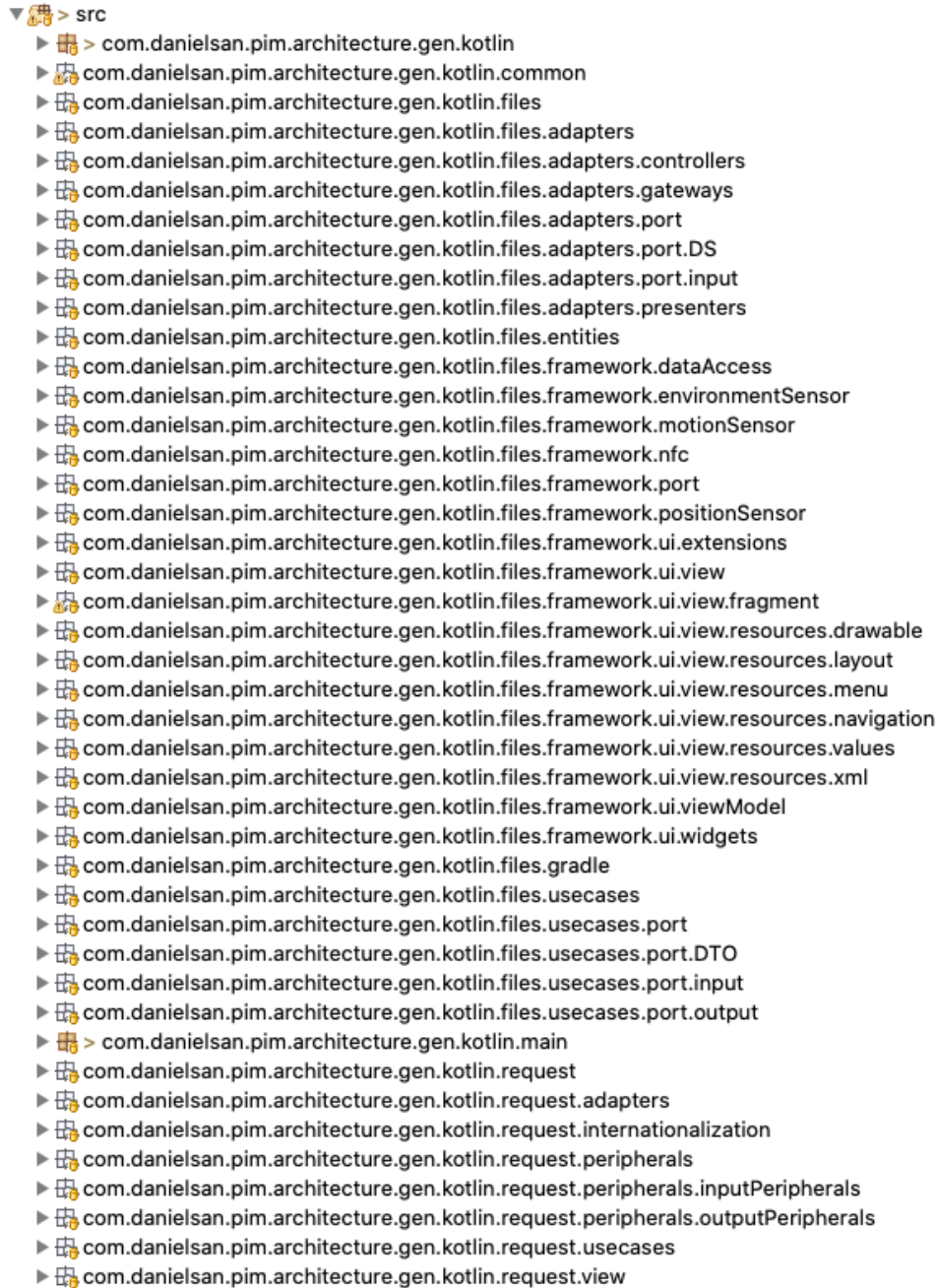
La transformación de modelo a texto en Acceleo es el componente más grande de la cadena de transformación de modelos, debido a que necesita tener una plantilla Android completa por cada archivo que se requiere generar para la aplicación final, además de los archivos que un desarrollador crearía cuando trabaja en la construcción de una aplicación Android. Esta transformación también genera algunos archivos de configuración que son necesarios para el framework Android.

La figura 21 presenta todos los paquetes utilizados en el proyecto Acceleo. Sin embargo, el *namespace* raíz es `com.danielsan.pim.architecture.gen.kotlin`. Por tanto, dentro de este directorio raíz están todos los archivos que son parte de la transformación. De esta manera, solo hay cuatro directorios que son `common`, `files`, `main` y `requests`, como se presenta en la figura 22. El archivo `Activator.java` es generado por Eclipse.

### Directorios `main` y `request`

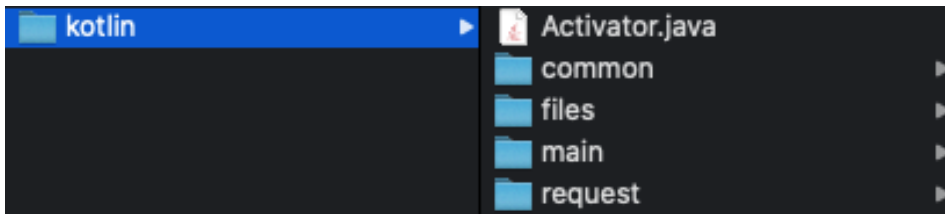
El primer punto de acceso del proyecto Acceleo se lleva a cabo mediante la plantilla `main.mtl`, la cual es un módulo de tipo `Main` y es el único que va acompañado de un archivo `Main.java`, el cual es generado por Acceleo. Este módulo debe invocar todos los demás módulos del proyecto para ejecutar la transformación completa que genera todos los archivos de código fuente y configuración de la aplicación final Android. Este módulo se encuentra en el directorio `main`, como se presenta en la figura 23.

**Figura 21.** Estructura de paquetes del proyecto Acceleo



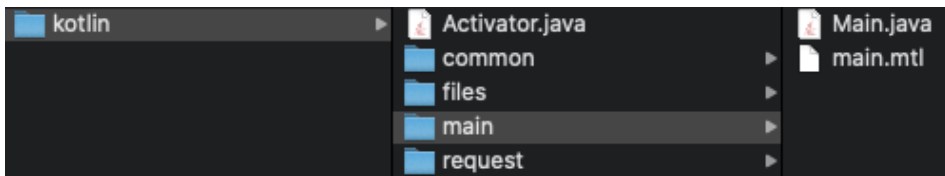
Fuente: elaboración propia

**Figura 22.** Directorios del proyecto Acceleo



Fuente: elaboración propia

**Figura 23.** Módulo main.mtl



Fuente: elaboración propia

Cualquier módulo de Acceleo tiene la misma estructura. En la primera línea está la codificación y en la segunda la referencia del metamodelo fuente, que en este caso es el metamodelo de arquitectura. El listado 6.1 presenta la definición del módulo en Acceleo.

**Listado 6.1:** Definición del módulo en Acceleo

```
1 [comment encoding = UTF-8 /]  
2 [module main("http://danielsan.com/mpml/architecture")]
```

La definición del módulo se incluye en una definición de plantilla como se presenta en el listado 6.2.

**Listado 6.2:** Definición de la plantilla en Acceleo

```
1 [template public main(anApp : App)]  
2 ...  
3 [/template]
```

La definición de la plantilla tiene como método el nombre del módulo, con el parámetro anApp de tipo App, el cual es el tipo del elemento del metamodelo que será leído en el módulo Acceleo. El elemento App es utilizado debido a que es el elemento superior en la jerarquía del metamodelo

de arquitectura y contiene todos los elementos del modelo. El listado 6.3 presenta el código fuente del módulo `main.mtl`.

Listado 6.3: Módulo `main.mtl` en Acceleo

```

1  [comment
    encoding = UTF-8 /]
2  [module main("http://danielsan.com/mpml/architecture")]
3  [import com::danielsan::pim::architecture::gen::kotlin::request::
4  generateFrameworkComponent/]
5  [import com::danielsan::pim::architecture::gen::kotlin::request::
6  generateAdaptersComponent/]
7  [import com::danielsan::pim::architecture::gen::kotlin::request::
8  generateConfigs /]
9  [import com::danielsan::pim::architecture::gen::kotlin::request::
10 generateUseCasesComponent /]
11 [import com::danielsan::pim::architecture::gen::kotlin::request::
12 generateEntitiesComponent /]
13 [import com::danielsan::pim::architecture::gen::kotlin::request::
14 peripherals::generateInputPeripheral /]
15 [import com::danielsan::pim::architecture::gen::kotlin::common::
16 QuerySelector /]
17 [template public main(anApp : App)]
18   [comment @main /]
19   [comment ***** /]
20   [comment Generate project basic structure ***** /]
21   [comment ***** /]
22   [comment Framework /]
23   [anApp.generateFrameworkComponent()/]
24   [anApp.generateConfigs()/]
25   [comment generate all the Clean Architecture components just
26   in the case there exists Controllers. Not only embedded
27   Peripherals /]
28   [if (anApp.getControllers() ->
29   union(anApp.getEmbeddedControllers()) -> size() > 0)]
30     [comment Adapters /]
31     [anApp.generateAdaptersComponent()/]
32     [comment UseCases /]
33     [anApp.generateUseCasesComponent()/]
34     [comment Entities /]
35     [anApp.generateEntitiesComponent()/]
36   [/if]
37   [comment ***** /]
38   [comment Generate the complete flow ***** /]
39   [comment ***** /]
40   [anApp.start.connect -> select(p |
41   p.oclIsKindOf(InputPeripheral)) -> collect(p |
42   p.oclAsType(InputPeripheral)) -> collect(p |
43   generateInputPeripheral(p, anApp))/]
44   [/template]

```

El objetivo de este módulo es lanzar la ejecución de los módulos que crean cada uno de los componentes de arquitectura limpia. Para ello, se hacen las siguientes operaciones:

- La línea 2 define el metamodelo fuente de la transformación, que es el metamodelo de arquitectura.
- Las líneas 3-9 importan los módulos de la transformación.
- Las líneas 16 y 17 usan el parámetro `anApp` para invocar los módulos `generateFrameworkComponent()` y `generateConfigs()`.
- La línea 26 tiene una estructura de condición que consulta si el modelo tiene algún `Controller` o `EmbeddedController`. En caso verdadero, las líneas 20-25 usan el parámetro `anApp` para realizar invocaciones a los módulos:
  - `generateAdaptersComponent()` para crear los adaptadores.
  - `generateUseCasesComponent()` para crear los casos de uso.
  - `generateEntitiesComponent` para crear las entidades.
- La línea 30 invoca el módulo `generateInputPeripheral(p,anApp)`, que genera el código para cada elemento `InputPeripheral`. En esta línea se realizan las siguientes tareas:

1. Obtiene el elemento de inicio relacionado con el elemento `anApp` y extrae el elemento conectado mediante la relación `connect` definida en el metamodelo de arquitectura.

```
1 | anApp.start.connect
```

2. Filtra los elementos de la relación `connect` que sean de tipo `InputPeripheral` mediante una expresión lambda con la función `oclIsKindOf()` en la operación `select()`.

```
1 | anApp.start.connect -> select(p |
2 | p.oclIsKindOf(InputPeripheral))
```

3. Convierte los objetos a `InputPeripheral` mediante una expresión lambda con la función `oclAsType()` en la operación `collect()`.

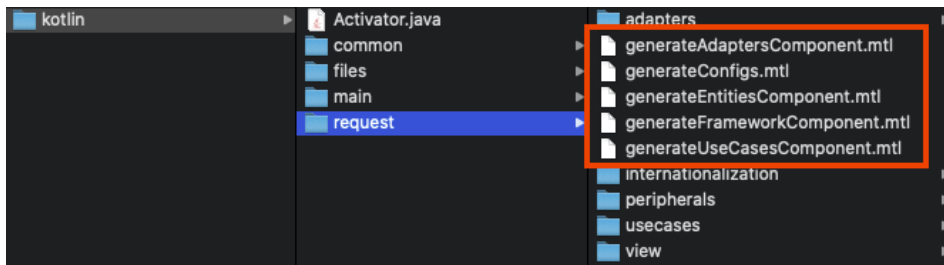
```
1 | anApp.start.connect -> select(p |
2 |   p.oclIsKindOf(InputPeripheral)) -> collect(p |
3 |     p.oclAsType(InputPeripheral))
```

4. Invoca `generateInputPeripheral(p, anApp)` con dos parámetros. El primero de tipo `InputPeripheral` y el segundo de tipo `App` utilizando una expresión lambda en la operación `collect()`.

```
1 | anApp.start.connect -> select(p |
2 |   p.oclIsKindOf(InputPeripheral)) -> collect(p |
3 |     p.oclAsType(InputPeripheral)) -> collect(p |
4 |       generateInputPeripheral(p, anApp))
```

Todos estos módulos que generan cada uno de los componentes de arquitectura limpia están dentro del directorio `request`, como se presenta en la figura 24.

**Figura 24.** Módulos de los generadores de arquitectura limpia con Acceleo



Fuente: elaboración propia

Todos los módulos siguientes se generan de la misma forma. Sin embargo, se presenta una diferencia que es la complejidad de las validaciones que deben tener los módulos. Por ejemplo, en el listado 6.4 está el código del módulo `generateAdaptersComponent()` cuando se realiza una validación para generar el contrato de puerto `IUserInterface`, solo en el caso de que exista elementos `Screen`.

#### Listado 6.4: Módulo `generateAdaptersComponent.mtl` en Acceleo

```
1 | [comment encoding = UTF-8 /]
2 | [module generateAdaptersComponent(
   |   "http://danielsan.com/mpml/architecture")]
```

```

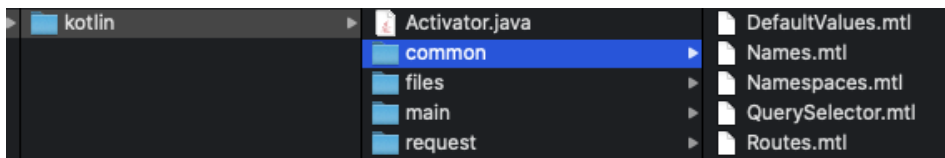
3  [import com::danielsan::pim::architecture::gen::kotlin::files::
4  adapters::port::DS::DS/]
5  [import com::danielsan::pim::architecture::gen::kotlin::files::
6  adapters::port::input::IUserInterface/]
7  [import com::danielsan::pim::architecture::gen::kotlin::files::
8  adapters::port:: AdaptersFactory/]
9  [import com::danielsan::pim::architecture::gen::kotlin::files::
10 adapters:: AdaptersModuleGradle/]
11 [import com::danielsan::pim::architecture::gen::kotlin::common::
12 QuerySelector/]
13 [template public generateAdaptersComponent(anApp : App)]
14 [anApp.AdaptersModuleGradle()/]
15 [if (anApp.getScreens() -> size() > 0)]
16 [anApp.IUserInterface()/]
17 [/if]
18 [anApp.AdaptersFactory()/]
19 [/template]

```

## Directorio common

En el directorio common están los módulos utilitarios que son útiles para todo el proyecto. La figura 25 presenta los archivos que contiene este directorio.

Figura 25. Módulos utilitarios



Fuente: elaboración propia

- `DefaultValues.mtl`. Este módulo tiene la definición de los valores por defecto que se asignarán a las variables en función del tipo de datos detectado como:
  - para `String`.
  - 0 para `Int`.
  - 0.0 para `Double`.
  - `LocalDate.now()` para `LocalDate`.
  - `null` para `Any`.

Esta utilidad es idónea para crear todas las entidades y las entidades DTO usadas en los puertos para comunicar diferentes componentes.

- `Names.mt1`. Este módulo centraliza todos los estándares de nombres utilizados en la creación de archivos. Por ejemplo, si recibe una entidad llamada `Persona` y tiene que crear un `Controller`, entonces se añadirá esta palabra como sufijo del nombre de la entidad y devuelve el nombre `PersonaController`. Esta nomenclatura de nombres se aplica en cada elemento de clase que se necesite crear. El proyecto incluye 45 estándares. Algunos de estos son los siguientes:
  - Elementos de arquitectura limpia:
    - Añade el sufijo `Controller` para el controlador.
    - Añade el sufijo `DS` al puerto de salida del adaptador.
    - Añade el prefijo `List` y el sufijo `Presenter` para el `List Presenter` que se utiliza para mostrar una lista de elementos.
    - Añade el prefijo `Register` y el sufijo `Presenter` para el `RegisterPresenter` que se utiliza para vincular un formulario que recibirá información del usuario.
  - Archivos específicos de `Android Framework`:
    - Convertir a minúsculas y añade el sufijo `_fragment` para el fragmento *Layout XML Android file*.
  - Archivos específicos de periféricos:
    - Toma el nombre de la entidad, la primera letra en mayúscula y el sufijo `NFCReader` para el periférico `NFCReader`.
    - Toma el nombre de la entidad, coloca mayúscula en la primera letra y el sufijo `MotionSensor` para el periférico `MotionSensor`.
    - Toma el nombre de la entidad, coloca mayúscula en la primera letra y añade el sufijo `PositionSensor` para el periférico `PositionSensor`.

- `Namespaces.mtl`. En este archivo se centralizan todos los espacios de nombres utilizados en la aplicación Android generada.
- `Routes.mtl`. Este módulo se apoya en el módulo `Namespaces.mtl` para generar las rutas en las que se van a colocar los archivos. Incluye 87 reglas en total.
- `QuerySelector.mtl`. Este módulo incluye reglas para navegar por el modelo de arquitectura desde el elemento raíz de la aplicación. En los casos en que se realice una modificación en el metamodelo de arquitectura y, por tanto, el modelo fuente de esta transformación también lo haga, entonces el código que busca elementos se colocará únicamente en este módulo.

## Directorio files

Este directorio contiene los módulos que generan los archivos reales. Estos módulos se agrupan en las siguientes categorías:

- `files/entities`. Este directorio contiene los módulos que generarán el componente interno de arquitectura limpia:
  - `EntitiesModuleGradle.mtl`. El archivo gradle es el archivo de configuración utilizado por el gestor de dependencias de Android.
  - `Entity.mtl`. Este módulo genera la clase de lógica de negocio relacionada con las entidades que se extraerán del metamodelo de arquitectura. Esta clase aplica las reglas de lógica de negocio y si alguna de ellas no se cumple lanza una excepción de tipo `IllegalArgumentException`.
- `files/adapters`. Este directorio incluye los archivos específicos relacionados con el componente Adapter:
  - `AdaptersModuleGradle.mtl`. Este módulo genera el archivo gradle que se requiere para el proyecto *Class Library* y la dependencia correspondiente al componente `useCases`.

- `controllers/Controller.mtl`. Este módulo recibe una referencia del elemento `Controller` del metamodelo de arquitectura y con ella genera el código fuente del archivo `Controller`.
- `gateways/Gateway.mtl`. Este módulo recibe una referencia `Gateway` del metamodelo de arquitectura y con ella genera el código fuente del archivo `Gateway`.
- `presenters/PresenterList.mtl`. Este módulo recibe una referencia del elemento `PresenterList` del metamodelo de arquitectura y con ella genera el código fuente del archivo `PresenterList`.
- `presenters/PresenterRegister.mtl`. Este módulo recibe una referencia del elemento `PresenterRegister` del metamodelo de arquitectura y con ella genera el archivo `PresenterRegister`.
- `port/AdaptersFactory.mtl`. Crea las clases concretas que implementan las interfaces que se utilizarán como contratos para comunicarse con otros componentes.
- `port/DS/DS.mtl`. Este módulo genera la entidad `DTO` que se utilizará para comunicar los datos del componente `Adapter` a los demás componentes.
- `port/Tnput.mtl`. Este módulo contiene la generación de los contratos que usará el componente `Framework` para utilizar los servicios expuestos por el componente `Adapter`.
- `files/usecases`. Este directorio contiene los `UseCases` y es el componente intermedio entre las entidades y los adaptadores:
  - `UseCasesModuleGradle.mtl`. Este módulo genera el archivo `gradle` para este proyecto de *Class Library*.
  - `UseCaseList.mtl`. Este módulo genera la clase concreta `Use Case` que se utiliza para mostrar una lista de resultados en el periférico `Framework`.

- `UseCaseRegister.mtl`. Este módulo genera el operador Use Case que se utiliza para guardar operaciones en la fuente de datos.
- `files/framework`. Este directorio se genera el componente más externo de la arquitectura limpia. Es el único que es específico para la plataforma Android:
  - `framework/dataAccess/DAO.mtl`. Utilizando el patrón DAO (*Data Access Object*), se definen las clases para realizar todas las operaciones con cualquier fuente de datos requerida por un periférico de entrada o salida. Las definidas en el metamodelo de arquitectura son las `RealTimeDatabase`, `NFCReader` y `NFCWriter`.
  - Los siguientes módulos generan un archivo de clase por cada `Peripheral`.
    - `framework/environmentSensor/EnvironmentSensor.mtl`
    - `framework/motionSensor/MotionSensor.mtl`
    - `framework/nfc/NFCReader.mtl`
    - `framework/nfc/NFCWriter.mtl`
    - `framework/positionSensor/PositionSensor.mtl`
  - El directorio `framework/port` contiene la generación de la definición de la clase DTO en el módulo `FrameworkDTO.mtl` y el módulo `FrameworkFactory.mtl` genera la clase que construirá las definiciones concretas utilizadas en este componente.
  - El directorio `framework/ui/extensions` contiene dos utilidades para la interfaz de usuario:
    - `EditTextExtensions.mtl`. Este módulo permite que la aplicación Android generada muestre unos mensajes amigables en forma de UI que presentan los mensajes de validación. Esto está conectado con toda la arquitectura.

- `ViewExtensions.mtl`. Este módulo vincula el elemento de interfaz de usuario con el `ViewModel`.
- `ui/view/fragment/`. Este directorio contiene los módulos que generarán los diferentes tipos de fragmentos Android que se requieran para la aplicación, dependiendo del caso.
- La aplicación Android generada utiliza componentes de la arquitectura Android de Jetpack<sup>1</sup> que proporciona un elemento `androidx.lifecycle.LiveData`<sup>2</sup> que utiliza el patrón *Observer* para conectar con la fuente de datos y detectar cualquier actualización de la fuente de datos. Esto contiene una clase que se extiende a `androidx.lifecycle.ViewModel`<sup>3</sup>, la cual es responsable de la preparación y gestión de los datos para la interfaz de usuario. En el directorio `ui/viewModel` se encuentran los módulos para llevar a cabo la generación de estas clases Kotlin:
  - `BaseViewModelFactory.mtl`. Este módulo genera una fábrica creadora de `ViewModel`.
  - Con base en los conceptos `UseCaseList`, `UseCaseRegister`, `PresenterList` y `PresenterRegister`, el módulo `ListViewModel.mtl` genera un archivo para devolver a los componentes Framework los datos de las fuentes de datos, mientras que el módulo `RegisterViewModel.mtl` genera un archivo para almacenar y modificar datos en las fuentes de datos.
  - El módulo `ui/widgets/CustomAlertDialog.mtl` genera una utilidad UI para mostrar ventanas de diálogo personalizadas para las aplicaciones generadas que se utilizan en varios casos para avisar al usuario acerca de mensajes relacionados con los comportamientos de los periféricos.

<sup>1</sup> <https://developer.android.com/topic/architecture>

<sup>2</sup> <https://developer.android.com/topic/libraries/architecture/livedata>

<sup>3</sup> <https://developer.android.com/topic/libraries/architecture/viewmodel>

## Capítulo 7. Validación

La cadena de transformación de modelos es validada con tres casos de estudio. Cada caso requirió de la creación del modelo de dominio conforme con el metamodelo de dominio MPML, la ejecución de la transformación de modelo a modelo y la ejecución de la transformación de modelo a texto. Finalmente, el resultado que corresponde al código fuente Kotlin para Android, es ejecutado en el IDE Android Studio usando un *Android Virtual Device* y un dispositivo móvil físico con sistema operativo Android 11.

### Caso de estudio GPS

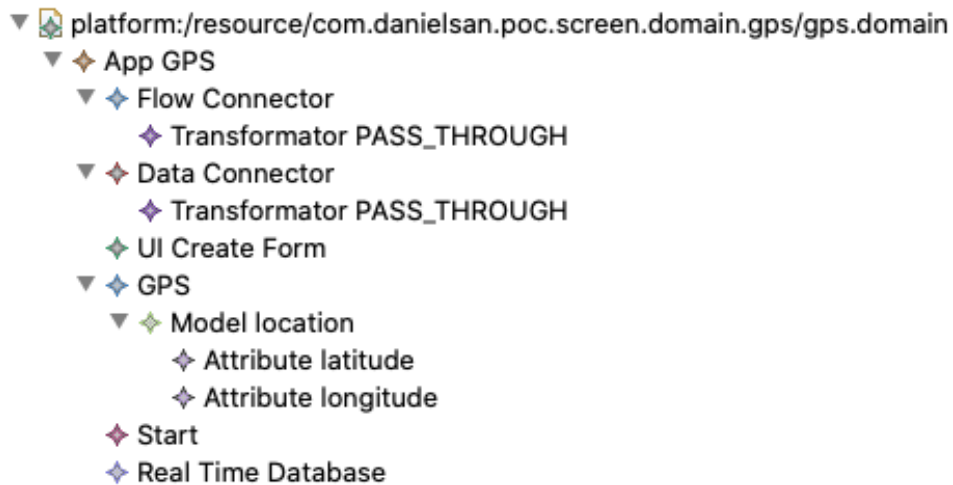
Este caso de estudio tiene como objetivo crear un modelo de dominio que modele una app que captura la localización GPS durante unos periodos de tiempo y la almacena en Firestore Firebase.

#### Modelo de dominio

El modelo de dominio creado para este caso de estudio se presenta mediante el editor nativo de EMF en la figura 26. Sin embargo, la representación gráfica del modelo de dominio usando el editor gráfico MPML se presenta en la figura 27. El modelo de dominio cuenta con los siguientes elementos:

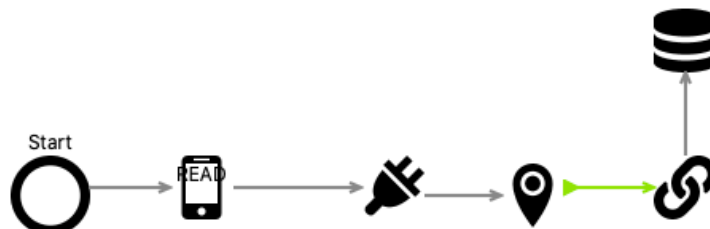
- **Start.** Elemento con el cual inicia el flujo de la aplicación.
- **UIElement.** Este elemento es una interfaz de usuario Screen de tipo READ.

**Figura 26.** Caso de estudio GPS-modelo de dominio



Fuente: elaboración propia

**Figura 27.** Caso de estudio GPS-versión gráfica del modelo de dominio



Fuente: elaboración propia

- FlowConnector. Este elemento permite el flujo de datos entre el UIElement y el siguiente elemento que es GPS.
- GPS. Es el elemento que captura datos del periférico GPS del dispositivo y tiene un modelo de localización con coordenadas de latitud y longitud.
- DataConnector. Este elemento permite el flujo de almacenamiento de datos entre el GPS y el elemento de persistencia que es RealTime Database.

- **RealTimeDatabase.** Este elemento permite el almacenamiento de la información capturada por el periférico GPS en una base de datos NoSQL de tiempo real Firestore.

Debido a que la localización tiene una salida que incluye latitud, longitud y la naturaleza de Firestore, la cual es una base de datos de tipo documental, no es necesario crear una estructura previa, simplemente se trata de crear una colección. En este caso, la transformación crea una colección en Firestore llamada `location`, en la que se almacenarán los documentos con los campos `latitude` y `longitude`.

### **Transformación de modelo a modelo**

Con base en el modelo de dominio conforme con el metamodelo de dominio MPML, la ejecución de la transformación ATL genera el modelo de arquitectura presentado en la figura 28. Para guardar la localización en la `RealTimeDatabase` se generan todos los componentes de arquitectura limpia con su respectivo `Controller` como punto de acceso. Asimismo, el componente GPS embebido con su respectivo `EmbeddedConnector`.

### **Transformación de modelo a texto**

Con base en el modelo de arquitectura conforme con el metamodelo de arquitectura, la ejecución de la transformación *Acceleo* identifica que el modelo tiene un `Screen` que no tiene un modelo relacionado, pero está conectado directamente a un `EmbeddedPeripheral` que en este caso es el GPS. Entonces, la transformación *Acceleo* inserta el código fuente para obtener la localización del elemento GPS en el `Fragment` de Android relacionado con el elemento UI, que en este caso es un `Form`. Además, la transformación genera los elementos de arquitectura limpia desde el GPS al `RealTimeDatabase`.

La aplicación generada tiene solo un elemento `Screen` con un botón, como se presenta en la figura 29. Cuando el usuario pulsa el botón, entonces la aplicación lee la localización desde el dispositivo y dicha información de localización se almacena en la base de datos en tiempo real Firestore.

**Figura 28.** Caso de estudio GPS-modelo de arquitectura



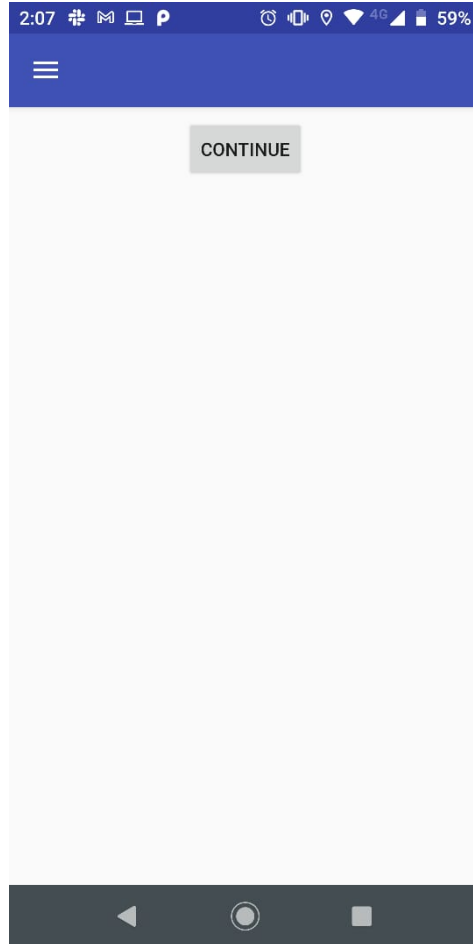
Fuente: elaboración propia

Finalmente, las capturas se almacenan correctamente en la base de datos, como se presenta en la figura 30.

## Caso de estudio sensor

La plataforma Android cuenta con varios sensores, como los de ambiente. Este caso de estudio genera una aplicación que usa el sensor de ambiente TYPE\_LIGHT, que captura la iluminación ambiente y es muy utilizado para la calibración del brillo de la pantalla. Así, este caso de estudio captura información de este sensor de ambiente y la almacena en una base de datos de Firestore.

**Figura 29.** Caso de estudio GPS-impresión de pantalla de la aplicación generada



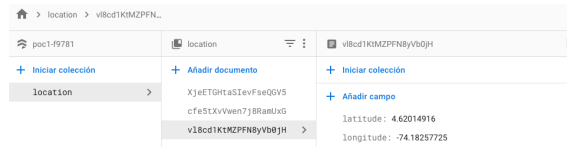
**Fuente:** elaboración propia

## Modelo de dominio

El modelo de dominio creado para este caso de estudio se presenta mediante el editor gráfico MPML en la figura 31. El modelo de dominio cuenta con los siguientes elementos:

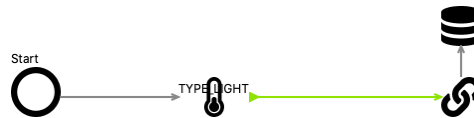
- Start. Elemento con el cual inicia el flujo de la aplicación.

**Figura 30.** Caso de estudio GPS-impresión de pantalla de Firestore



Fuente: elaboración propia

**Figura 31.** Caso de estudio Sensor-versión gráfica del modelo de dominio



Fuente: elaboración propia

- **EnvironmentSensor.** Es el elemento que captura datos del sensor que captura la intensidad de luz del ambiente.
- **DataConnector.** Este elemento permite el flujo de almacenamiento de datos entre el EnvironmentSensor y el elemento de persistencia que es RealTimeDatabase.
- **RealTimeDatabase.** Este elemento permite almacenar la información capturada por el sensor de ambiente en una base de datos de tiempo real Firestore.

## Transformación de modelo a modelo

Con base en el modelo de dominio conforme con el metamodelo de dominio MPML, la ejecución de la transformación ATL genera el modelo de arquitectura presentado en la figura 32. Para guardar la información sobre la intensidad de la luz en la RealTimeDatabase se generan todos los componentes de arquitectura limpia con su respectivo Controller como punto de acceso.

**Figura 32.** Caso de estudio Sensor-modelo de arquitectura



Fuente: elaboración propia

## Transformación de modelo a texto

Con base en el modelo de arquitectura conforme con el metamodelo de arquitectura, la ejecución de la transformación Acceleo identifica que el modelo no tiene un `Screen`, entonces, la transformación genera un UI simple; es decir, una pantalla vacía, solo para capturar las lecturas del sensor de ambiente. Entonces, la transformación inserta este código en el `Fragment` relacionado con el elemento UI. Además, genera los elementos de arquitectura limpia para la `RealTimeDatabase`.

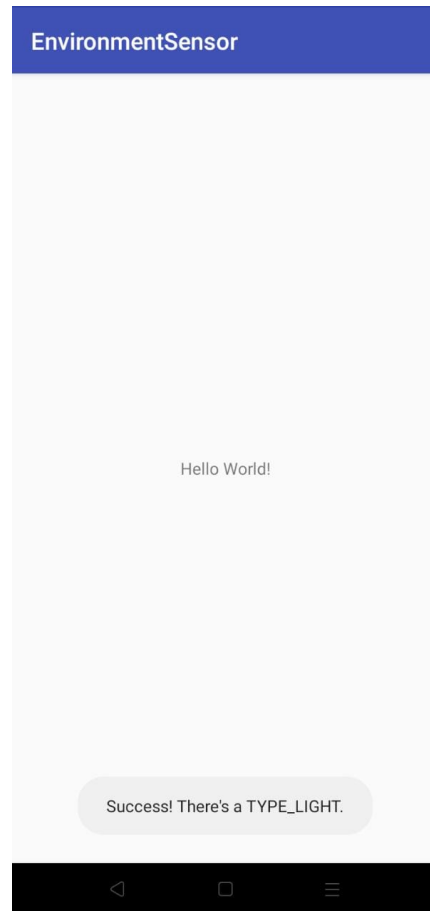
La aplicación generada tiene solo una pantalla con un fragmento en blanco, como se presenta en la figura 33. Una vez que la aplicación se inicia, el sensor lee la intensidad de la luz y almacena esa información en la base de datos en tiempo real `Firestore`.

Finalmente, las capturas se almacenan en la base de datos correctamente, como se presenta en la figura 34.

## Caso de estudio NFC

La tecnología de identificación por radiofrecuencia (RFID) permite transmitir operaciones de datos digitales (lectura/escritura) en tarjetas RFID, etiquetas o dispositivos (Weinstein, 2005).

**Figura 33.** Caso de estudio Sensor-impresión de pantalla de la aplicación generada

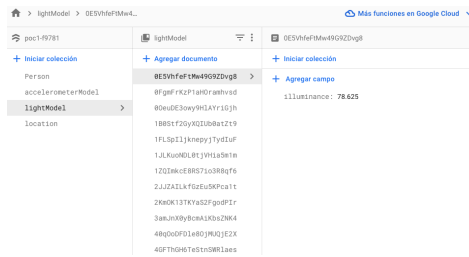


**Fuente:** elaboración propia

Uno de los usos típicos de esta tecnología es registrar el acceso concedido a una persona a un lugar acercando una tarjeta RFID a un dispositivo lector. Este caso de estudio tiene como objetivo crear dos aplicaciones:

1. La primera aplicación permite escribir en una tarjeta RFID el id y el nombre del usuario.
2. La segunda aplicación leerá esta información de la tarjeta RFID y la registrará en una base de datos.

**Figura 34.** Caso de estudio Sensor-impresión de pantalla de Firestore

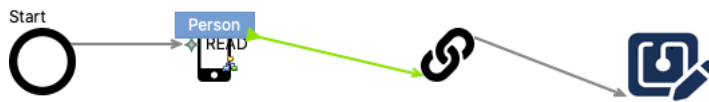


Fuente: elaboración propia

## Primera aplicación

El modelo de dominio creado para la primera aplicación de este caso de estudio se presenta mediante el editor gráfico MPML en la figura 35.

**Figura 35.** Caso de estudio NFC-modelo de dominio de la primera aplicación



Fuente: elaboración propia

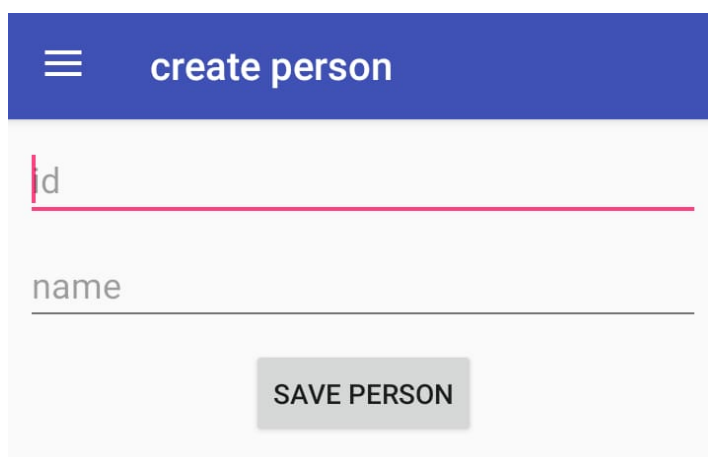
El modelo de dominio cuenta con los siguientes elementos:

- **Start.** Elemento con el cual inicia el flujo de la aplicación.
- **UIElement.** Este elemento es una interfaz de usuario Screen de tipo READ.
- **DataConnector.** Este elemento permite el flujo de almacenamiento de datos entre el UIElement y el elemento NFCWriter. Este conector tiene un transformador con una operación de tipo JSON\_ENCODE.
- **NFCWriter.** Este elemento permite el almacenamiento de la información capturada por el formulario en una tarjeta RFID.

Después de ejecutar la transformación de modelo a modelo y la transformación de modelo a texto, la aplicación generada tiene solo una pantalla con un formulario, como se presenta en la figura 36. Este formulario incluye un campo de texto id, un campo de texto name y un botón, los cuales

permiten introducir los datos que se desean almacenar en la tarjeta RFID. Después de hacer clic en el botón, la aplicación muestra una pantalla en blanco que corresponde a una `AppCompatActivity` que tendrá el código del periférico `NFCWriter`. Esta acción se realiza debido a que el *callback* del `NFCWriter` está muy acoplado a la referencia del `Activity` en Android. Esta característica rompe la arquitectura limpia, y es por eso que en el metamodelo de arquitectura se modela como un `EmbeddedComponent`.

**Figura 36.** Caso de estudio NFC-impresión de pantalla de la primera aplicación



**Fuente:** elaboración propia

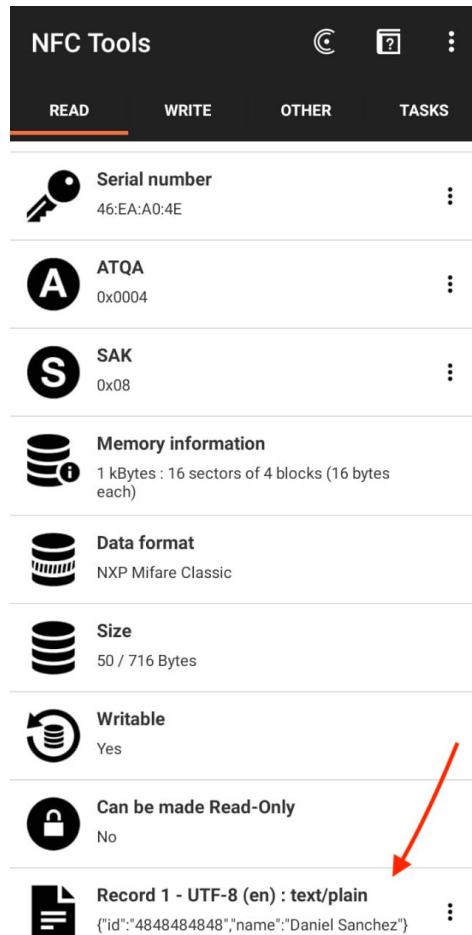
Después de escribir el valor en la tarjeta RFID, se puede confirmar en una aplicación externa que los datos se almacenaron en la tarjeta RFID en formato JSON, como se muestra en la figura 37.

## Segunda aplicación

El modelo de dominio creado para la segunda aplicación de este caso de estudio se presenta mediante el editor gráfico MPML en la figura 38. El modelo de dominio cuenta con los siguientes elementos:

- **Start.** Elemento con el cual inicia el flujo de la aplicación.
- **NFCReader.** Este elemento permite la lectura de la información almacenada en una tarjeta RFID.

**Figura 37.** Caso de estudio NFC-datos escritos en la tarjeta RFID



**Fuente:** elaboración propia

- **DataConnector.** Este elemento permite el flujo de almacenamiento de datos entre el NFCReader y el elemento RealTimeDatabase. Este conector tiene un transformador con una operación de tipo JSON\_DECODE.
- **RealTimeDatabase.** Este elemento permite el almacenamiento de la información almacenada en una tarjeta RFID en una base de datos de tiempo real Firestore.

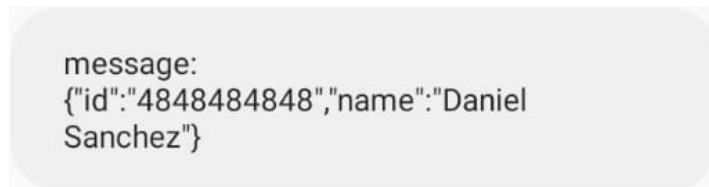
**Figura 38.** Caso de estudio NFC-modelo de dominio de la segunda aplicación



Fuente: elaboración propia

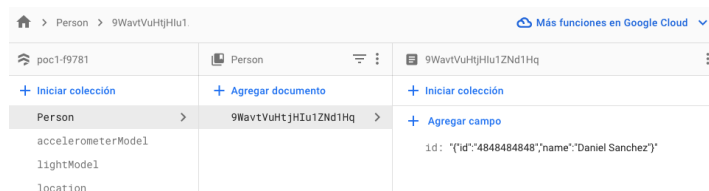
Después de ejecutar la transformación de modelo a modelo y la transformación de modelo a texto, la aplicación generada tiene solo un Activity con una pantalla en blanco que permite la lectura de la tarjeta RFID cuando es acercada al dispositivo móvil. Al hacer la lectura, la aplicación muestra un mensaje en la UI con los datos que hay en la tarjeta, como se presenta en la figura 39. Posteriormente, la aplicación almacena los datos almacenados en la tarjeta RFID en la base de datos en tiempo real Firestore, como se presenta en la figura 40.

**Figura 39.** Caso de estudio NFC-mensaje de lectura de tarjeta RFID



Fuente: elaboración propia

**Figura 40.** Caso de estudio NFC-impresión de pantalla de Firestore



Fuente: elaboración propia

# Capítulo 8. Discusión

## Síntesis

Este libro de investigación propone una solución para el desarrollo ágil de *software* para dispositivos móviles en el contexto de la ingeniería dirigida por modelos. Esta solución está enmarcada en el desarrollo de una cadena de transformación de modelos para generar una aplicación de *software* móvil para la plataforma Android que permita la configuración de los periféricos del dispositivo, con base en la construcción de un solo modelo que abstraer los requerimientos del dominio.

Este libro de investigación presenta los siguientes resultados:

- Revisión de la literatura relacionada con MDE presentada en el capítulo 1, la cual fue el fundamento para la creación de la cadena de transformación de modelos presentada en el capítulo 2 y para la creación del metamodelo de dominio MPML, que tiene como fin ofrecer al modelador una abstracción en el dominio de aplicaciones móviles. Así, con base en el metamodelo de dominio MPML, el modelador tiene los recursos para crear un modelo de dominio que satisfaga los requerimientos de la aplicación que desea generar.
- Creación del metamodelo de dominio MPML presentado en el capítulo 3 y el metamodelo de arquitectura presentado en el capítulo 4, debido a que estos metamodelos especifican los elementos del dominio y los elementos de la arquitectura que interactúan mediante la cadena de transformación de modelos, para generar un

modelo de arquitectura con base en un modelo de dominio creado por el modelador.

- Creación de la transformación modelo a modelo en lenguaje ATL presentado en el capítulo 5 y la creación de la transformación de modelo a texto en lenguaje Acceleo presentado en el capítulo 6. Desde el punto de vista de la cadena de transformación de modelos, estas transformaciones son conformes con el metamodelo de ATL y el metamodelo de Acceleo, respectivamente. Así, estas transformaciones dan como resultado aplicaciones de *software* para la plataforma Android.
- Creación de varios casos de estudio presentados en el capítulo 7, los cuales usan diferentes periféricos, como GPS, NFC y sensores. Estos casos de estudio se basaron en la creación del modelo de dominio y en la ejecución de las transformaciones incluidas en la cadena de transformación de modelos, generando para cada caso de estudio una aplicación de *software* para plataforma Android completamente funcional.
- Creación del DSML denominado MPML, que es basado del metamodelo de dominio MPML presentado en el capítulo 3. El editor del DSML permite al modelador crear el modelo de dominio, especificando el flujo de datos entre los componentes modelados para la generación de una aplicación de *software* en una plataforma Android específica.

## Trabajo futuro

Con base en la solución presentada en este libro de investigación, posibles investigaciones futuras basadas en cadenas de transformación de modelos que permitan el desarrollo ágil de *software* son las siguientes:

- Aplicaciones multiplataforma. Debido a que el metamodelo de arquitectura es independiente de la plataforma y que fue construido siguiendo los conceptos y principios de arquitectura limpia, es posible la creación de una nueva transformación de modelo a texto que

genere archivos de texto plano con código fuente en otros lenguajes (por ejemplo, Objective C) para otras plataformas como iOS.

- Integración con servicios. El enfoque principal de la cadena de transformación de modelos está determinado por la creación de aplicaciones para el manejo de periféricos de dispositivos móviles. Entonces, es posible extender el metamodelo de dominio con el fin de ofrecer a las aplicaciones móviles acceso a diversos servicios, como Google maps, autenticación de terceros (por ejemplo, Google, Outlook, Github), reconocimiento facial, entre otros. Incluso, algunos de esos servicios podrían interactuar con ciertos periféricos. Por ejemplo, el servicio de Google maps podría integrarse con el GPS.



## Capítulo 9. Conclusiones

Este capítulo discute las principales contribuciones del trabajo de investigación que se presentó en este libro.

- **Cadena de transformación de modelos**

La construcción de cadenas de transformación de modelos en el contexto de MDE es una estrategia muy bien aceptada en la academia e industria para contribuir con el desarrollo eficiente de productos de *software*.

En este trabajo de investigación, se ha propuesto una cadena de transformación de modelos específicamente para el desarrollo ágil de aplicaciones móviles en la plataforma Android con lenguaje de programación Kotlin. Las aplicaciones móviles generadas por la cadena de transformación de modelos están enfocadas en la configuración y uso de los periféricos que incluyen los dispositivos móviles. Mediante varios casos de estudio, se ha demostrado que el resultado de la ejecución de la cadena de transformación es correcto y que el esfuerzo para la generación de la aplicación final reside en la creación del modelo de dominio. Por tanto, la cadena de transformación de modelos es una herramienta idónea para la creación de *software*, ya que enfoca los esfuerzos en el análisis y modelamiento del negocio, en vez de enfocar los esfuerzos en el desarrollo de código fuente. Sin embargo, las aplicaciones generadas por la cadena de transformación de modelos son extensibles, lo cual permite desarrollar servicios específicos basados de los componentes generados.

El desarrollo de la cadena de transformación de modelos fue un proceso iterativo e incremental. Cada decisión tomada para cumplir con los requerimientos de las aplicaciones de *software* generadas para la plataforma Android ha demandado un análisis que da como consecuencia la actualización de alguno de los componentes de la cadena de transformación de modelos; es decir, el metamodelo de dominio, el metamodelo de arquitectura, la transformación de modelo a modelo y la transformación de modelo a texto. Esta característica es natural en el contexto de ingeniería de *software* y, además, implica que la cadena de transformación de modelos es extensible favoreciendo su evolución.

### ■ Metamodelos

Por un lado, el metamodelo de dominio permite al modelador la creación de un modelo conceptual que satisface los requerimientos del negocio bajo estudio, el cual es denominado modelo de dominio. Por tanto, el modelo de dominio es conforme con el metamodelo de dominio y es el insumo de entrada para la ejecución de la cadena de transformación de modelos. Por otro lado, el metamodelo de arquitectura incluye los conceptos modelados en el metamodelo de dominio, más los conceptos que abstraen la arquitectura de la aplicación (por ejemplo, arquitectura limpia).

En este trabajo de investigación, se ha propuesto una metodología para la creación del metamodelo de dominio. Esta metodología presenta un marco básico que hasta el momento se ha utilizado con éxito en la creación del metamodelo de dominio. Asimismo, fue posible validar la eficacia de la metodología, porque genera una taxonomía escalable de los elementos del dominio y ayuda con la definición inicial del metamodelo. Además, la creación del metamodelo de dominio se ha creado con el fin de abstraer el dominio, pero también con el fin de crear un DSML que ofrece un editor gráfico, el cual permite a los modeladores crear el modelo de dominio de manera simple y ágil e incluso sin conocimientos de desarrollo de *software*. El DSML creado se denomina

MPML, por tanto, el metamodelo de dominio en este proyecto se denomina metamodelo de dominio MPML. Este DSML provee elementos gráficos alineados con los conceptos definidos en el metamodelo de dominio MPML. De esta manera, con base en el DSML, se ha demostrado que crear los modelos de dominio es una tarea simple para un modelador. Asimismo, el editor gráfico tiene la capacidad de validar los modelos alertando al modelador sobre errores gramaticales y semánticos. Esta función del editor es muy relevante ya que asegura que un modelo esté en un estado correcto y deseado para ser usado como insumo de la cadena de transformación de modelos.

Además, también se ha creado un metamodelo de arquitectura, el cual es basado en el metamodelo de dominio e incluye los conceptos de arquitectura limpia con el fin de que los modelos de arquitectura conformes con el metamodelo de arquitectura sean correctos independientemente de la plataforma (por ejemplo, Android o iOS). La decisión de crear el metamodelo de arquitectura alineado con los conceptos de arquitectura limpia ha sido muy apropiada y relevante, porque permite que los modelos de arquitectura incluyan elementos que satisfacen la separación de preocupaciones y los principios de desarrollo de *software*.

#### ■ Transformaciones

Por un lado, la transformación de modelo a modelo permite tomar un modelo origen conforme con un metamodelo origen para trabajar con cada uno de sus elementos y con base en ellos generar un modelo destino conforme con un metamodelo destino. Por otro lado, la transformación de modelo a texto permite tomar un modelo origen conforme con un metamodelo origen y generar archivos de texto plano, donde dicho texto normalmente corresponde con código fuente. Por tanto, los archivos de texto plano deben ser conformes con un lenguaje de programación.

En este trabajo de investigación, se ha creado una transformación de modelo a modelo usando el lenguaje de transformación ATL que lee el modelo de dominio que es conforme con el metamodelo de dominio MPML y genera un modelo de arquitectura conforme con el metamodelo de arquitectura. La decisión de usar ATL para la transformación de modelo a modelo ha sido bastante acertada debido a que ATL incluye todas las herramientas necesarias para la construcción y ejecución de la transformación. Asimismo, debido a que ATL se basa en reglas, la transformación puede incluir algoritmos reutilizables para varios elementos del modelo de dominio.

Además, también se ha creado una transformación de modelo a texto usando el lenguaje de transformación Acceleo, que lee el modelo de arquitectura y genera código fuente en lenguaje Kotlin para aplicaciones móviles en plataforma Android. Acceleo se basa en módulos que permiten desacoplar la transformación en componentes unitarios. Esta característica es de gran utilidad debido a que esta transformación requiere una gran cantidad de desarrollo porque necesita un algoritmo específico por cada archivo de texto plano que se requiere generar en la aplicación final.

# Referencias

- Amrani, M., Combemale, B., Lúcio, L., Selim, G., Dingel, J., Vangheluwe, H., Cordy, J. R. *et al.* (2015). Formal verification techniques for model transformations: A tridimensional classification. *The Journal of Object Technology*, 14(3), 1-1.
- Arango, E. C. y Loaiza, O. L. (2021). SCRUM Framework Extended with Clean Architecture Practices for Software Maintainability. *Computer Science On-line Conference*, 667-681.
- Benouda, H., Azizi, M., Moussaoui, M. y Esbai, R. (2017). Automatic code generation within MDA approach for cross-platform mobiles apps. *2017 First International Conference on Embedded & Distributed Systems (EDiS)*, 1-5.
- Bettini, L. (2016). *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd.
- Bézivin, J. (2005). On the unification power of models. *Software and Systems Modeling*, 4(2), 171-188.
- Bézivin, J., Gérard, S., Muller, P.-A. y Rioux, L. (2003). MDA components: Challenges and Opportunities. *Workshop on Metamodelling for MDA*, 23-41.
- Bruel, J.-M., Combemale, B., Guerra, E., Jézéquel, J.-M., Kienzle, J., De Lara, J., Mussbacher, G., Syriani, E. y Vangheluwe, H. (2020). Comparing and classifying model transformation reuse approaches across metamodels. *Software and Systems Modeling*, 19(2), 441-465.
- Budinsky, F., Ellersick, R., Steinberg, D., Grose, T. J. y Merks, E. (2004). *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional.

- Chaudhary, H. A. A., Guevara, I., John, J., Singh, A., Ghosal, A., Pesch, D. y Margaria, T. (2022). Model-driven engineering in digital thread platforms: A practical use case and future challenges. *Leveraging Applications of Formal Methods, Verification and Validation. Practice: 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part IV*, 195-207.
- Di Sipio, C., Di Rocco, J., Di Ruscio, D. y Nguyen, D. P. T. (2021). A low-code tool supporting the development of recommender systems. *Fifteenth ACM Conference on Recommender Systems*, 741-744.
- Fowler, M. (2012). *Patterns of Enterprise Application Architecture: Pattern Enterprisepr Applica Arch.* Addison-Wesley.
- Fowler, M. (2015). PresentationDomainDataLayering.
- Frankel, S. I., Miller, J., Mischkinsky, J., Mukerji, J., Siegel, J., Soley, R., Tyndal-Biscoe, S., Uhl, A., Watson, A. y Wood, B. (2001). Model Driven Architecture (MDA).
- Ghezzi, C., Jazayeri, M., Mandrioli, D. et al. (2004). *Software qualities and principles*. CRC Press.
- Gotseva, D., Tomov, Y. y Danov, P. (2019). Comparative study Java vs. Kotlin. *2019 27th National Conference with International Participation (Telecom)*, 86-89.
- Greenfield, J. y Short, K. (2003). Software factories: assembling applications with patterns, models, frameworks and tools. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 16-27.
- Harrison, W. y Ossher, H. (1993). Subject-oriented programming: a critique of pure objects. *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, 411-428.
- Hawking, S. y Mlodinow, L. (2010). *The grand design*. Random House LLC.
- Holderer, K. (2022). Single responsibility principle: software design. *POGIL Activity Clearinghouse*, 3(4).
- Iung, A., Carbonell, J., Marchezan, L., Rodrigues, E., Bernardino, M., Basso, F. P. y Medeiros, B. (2020). Systematic mapping study on domain-specific language development tools. *Empirical Software Engineering*, 25, 4205-4249.

- Jemerov, D. y Isakova, S. (2017). *Kotlin in action*. Simon & Schuster.
- Jouault, F., Allilaire, F., Bézivin, J. y Kurtev, I. (2008). ATL: A model transformation tool. *Science of computer programming*, 72(1-2), 31-39.
- Kahani, N., Bagherzadeh, M., Cordy, J. R., Dingel, J. y Varró, D. (2019). Survey and classification of model transformation tools. *Software and Systems Modeling*, 18(4), 2361-2397.
- Kardas, G., Demirezen, Z. y Challenger, M. (2010). Towards a DSML for semantic web enabled multi-agent systems. *Proceedings of the International Workshop on Formalization of Modeling Languages*, 1-5.
- Kaschek, R. H. (2008). On the evolution of conceptual modeling. *Dagstuhl Seminar Proceedings*. <https://drops.dagstuhl.de/storage/16dagstuhl-seminar-proceedings/dsp-vol08181/DagSemProc.08181.5/DagSemProc.08181.5.pdf>
- Kelly, S. y Tolvanen, J.-P. (2021). Collaborative modelling and metamodelling with MetaEdit+. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 27-34.
- Kim, W. Y., Son, H. S., Kim, J. S. y Kim, R. Y. C. (2011). Adapting model transformation approach for android smartphone application. *Advanced Communication and Networking: Third International Conference, ACN 2011, Brno, Czech Republic, August 15-17, 2011. Proceedings*, 421-429.
- Lindland, O. I., Sindre, G. y Solvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software*, 11(2), 42-49.
- Ludewig, J. (2003). Models in software engineering—an introduction. *Software and Systems Modeling*, 2(1), 5-14.
- Martin, R. C. (2000). Design principles and design patterns. *Object Mentor*, 1(34), 597.
- Martin, R. C. (2018). *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall.
- Mernik, M., Heering, J. y Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4), 316-344.
- Moskala, M. y Wojda, I. (2017). *Android Development with Kotlin*. Packt Publishing Ltd.

- Muller, P.-A., Fondement, F., Baudry, B. y Combemale, B. (2012). Modeling modeling modeling. *Software and Systems Modeling*, 11(3), 347-359.
- Nozik, A. (2019). Kotlin language for science and Kmath library. *AIP Conference Proceedings*, 2163, 40004.
- Oakes, B. J., Troya, J., Lúcio, L. y Wimmer, M. (2018). Full contract verification for ATL using symbolic execution. *Software and Systems Modeling*, 17(3), 815-849.
- Object Management Group. (2015). *OMG Meta Object Facility (MOF) Core Specification*. Open Management Group.
- Open Management Group. (2005). *Unified modeling language: Superstructure*.
- Parr, T. y Fisher, K. (2011). LL (\*) the foundation of the ANTLR parser generator. *ACM Sigplan Notices*, 46(6), 425-436.
- Rachovski, T., Hadzhikoleva, S., Hadzhikolev, E. y Lengerov, A. (2022). Using clean architecture principles to improve the design and implementation of the mobiles online platform. *Proceedings of Second International Conference on Sustainable Expert Systems*, 11-19.
- Roussopoulos, N. y Karagiannis, D. (2009). Conceptual modeling: past, present and the continuum of the future. En *Conceptual Modeling: Foundations and Applications* (pp. 139-152). Springer.
- Rumbaugh, J., Jacobson, I. y Booch, G. (2004). *The Unified Modeling Language Reference Manual*. Pearson Higher Education.
- Sánchez, D. y Flórez, H. (2018). Model driven engineering approach to manage peripherals in mobile devices. *International Conference on Computational Science and Its Applications*, 353-364.
- Sánchez, D., Rojas, A. E. y Flórez, H. (2022). Towards a clean architecture for android apps using model transformations. *IAENG International Journal of Computer Science*, 49(1), 270-278.
- Schmidt, D. (1996). Programming language semantics. *ACM Computing Surveys (CSUR)*, 28(1), 265-267.
- Schmidt, D. (2006). Model-driven engineering. *IEEE Computer*, 39(2), 25-31.
- Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5), 26-32.
- Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5), 19-25.

- Sousa, G. C., Costa, F. M., Clarke, P. J. y Allen, A. A. (2012). Model-driven development of DSML execution engines. *Proceedings of the 7th Workshop on Models@ run. time*, 10-15.
- Stachowiak, H. (1973). *General model theory*. Springer.
- Steinberg, D., Budinsky, F., Merks, E. y Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education.
- Van Deursen, A. y Klint, P. (2002). Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1), 1-17.
- Weinstein, R. (2005). RFID: a technical overview and its application to the enterprise. *IT Professional*, 7(3), 27-33.



# Anexo A. Metamodelo de dominio

El listado A.1 presenta el código fuente en formato XML del metamodelo de dominio MPML.

Listado A.1: Metamodelo de dominio XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="domain"
5   nsURI="http://danielsan.com/mpml/domain" nsPrefix="domain">
6   <eClassifiers xsi:type="ecore:EClass" name="App">
7     <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
8       lowerBound="1" eType="ecore:EDatatype
9       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
10    <eStructuralFeatures xsi:type="ecore:EAttribute" name="namespace"
11      lowerBound="1" eType="ecore:EDatatype
12      http://www.eclipse.org/emf/2002/Ecore#//EString"/>
13    <eStructuralFeatures xsi:type="ecore:EReference" name="connectors"
14      upperBound="-1" eType="#//Connector" containment="true"/>
15    <eStructuralFeatures xsi:type="ecore:EReference" name="components"
16      upperBound="-1" eType="#//MPMLComponent" containment="true"/>
17    <eStructuralFeatures xsi:type="ecore:EReference" name="start"
18      eType="#//Start" containment="true"/>
19    <eStructuralFeatures xsi:type="ecore:EReference" name="datastorages"
20      upperBound="-1" eType="#//DataStorage" containment="true"/>
21  </eClassifiers>
22  <eClassifiers xsi:type="ecore:EClass" name="FlowConnector"
23    eSuperTypes="#//Connector">
24    <eStructuralFeatures xsi:type="ecore:EReference" name="connect"
25      lowerBound="1" eType="#//MPMLComponent"/>
26  </eClassifiers>
27  <eClassifiers xsi:type="ecore:EEnum" name="DataOperation">
28    <eLiterals name="PASS_THROUGH"/>
29    <eLiterals name="MERGE" value="1"/>
```

```

30     <eLiterals name="CONCAT" value="2"/>
31     <eLiterals name="JSON_ENCODE" value="3"/>
32     <eLiterals name="JSON_DECODE" value="4"/>
33 </eClassifiers>
34 <eClassifiers xsi:type="ecore:EClass" name="MPMLComponent"
35   abstract="true">
36   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
37     eType="ecore:EDatatype
38     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
39 </eClassifiers>
40 <eClassifiers xsi:type="ecore:EClass" name="MPMLInputComponent"
41   abstract="true" eSuperTypes="#//MPMLComponent">
42   <eStructuralFeatures xsi:type="ecore:EReference" name="connect"
43     eType="#//FlowConnector"/>
44   <eStructuralFeatures xsi:type="ecore:EReference" name="dataStore"
45     eType="#//DataConnector"/>
46 </eClassifiers>
47 <eClassifiers xsi:type="ecore:EClass" name="GPS"
48   eSuperTypes="#//MPMLInputComponent#//MPMLModeledComponent"/>
49 <eClassifiers xsi:type="ecore:EClass" name="MPMLOutputComponent"
50   abstract="true" eSuperTypes="#//MPMLComponent">
51   <eStructuralFeatures xsi:type="ecore:EReference" name="dataSource"
52     eType="#//DataConnector"/>
53 </eClassifiers>
54 <eClassifiers xsi:type="ecore:EClass" name="Camera"
55   eSuperTypes="#//MPMLInputComponent#//MPMLModeledComponent"/>
56 <eClassifiers xsi:type="ecore:EClass" name="Entity" abstract="true">
57   <eStructuralFeatures xsi:type="ecore:EReference" name="attributes"
58     upperBound="-1" eType="ecore:EAttribute" containment="true"/>
59   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
60     eType="ecore:EDatatype
61     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
62   <eStructuralFeatures xsi:type="ecore:EReference" name="validators"
63     upperBound="-1" eType="ecore:EReference" containment="true"/>
64 </eClassifiers>
65 <eClassifiers xsi:type="ecore:EClass" name="Vibrator"
66   eSuperTypes="#//MPMLOutputComponent#//MPMLModeledComponent"/>
67 <eClassifiers xsi:type="ecore:EClass" name="Transformator">
68   <eStructuralFeatures xsi:type="ecore:EAttribute" name="operation"
69     eType="ecore:EDatatype
70     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
71   <eStructuralFeatures xsi:type="ecore:EReference" name="transform"
72     upperBound="-1" eType="ecore:EReference" containment="true"/>
73 </eClassifiers>
74 <eClassifiers xsi:type="ecore:EClass" name="Attribute">
75   <eStructuralFeatures xsi:type="ecore:EReference" name="validatedBy"
76     eType="ecore:EReference"/>
77   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
78     eType="ecore:EDatatype
79     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
80   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
81     eType="ecore:EDatatype"/>

```

```

81     <eStructuralFeatures xsi:type="ecore:EAttribute" name="options"
82     upperBound="-1" eType="ecore:EDatatype
83     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
84     <eStructuralFeatures xsi:type="ecore:EAttribute" name="defaultValue"
85     eType="ecore:EDatatype
86     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
87   </eClassifiers>
88   <eClassifiers xsi:type="ecore:EClass" name="Validator">
89     <eStructuralFeatures xsi:type="ecore:EAttribute" name="isRequired"
90     eType="ecore:EDatatype
91     http://www.eclipse.org/emf/2003/XMLType#//Boolean"/>
92     <eStructuralFeatures xsi:type="ecore:EReference" name="sizelengths"
93     upperBound="-1" eType="#//SizeLength" containment="true"/>
94   </eClassifiers>
95   <eClassifiers xsi:type="ecore:EEnum" name="DataType">
96     <eLiterals name="STRING" value="1"/>
97     <eLiterals name="LONG_STRING" value="3"/>
98     <eLiterals name="EMAIL" value="4"/>
99     <eLiterals name="PASSWORD" value="5"/>
100    <eLiterals name="PASSWORD_NUMERIC" value="6"/>
101    <eLiterals name="PHONE" value="7"/>
102    <eLiterals name="POSTAL_ADDRESS" value="8"/>
103    <eLiterals name="TIME" value="9"/>
104    <eLiterals name="DATE" value="10"/>
105    <eLiterals name="INTEGER_NUMBER" value="11"/>
106    <eLiterals name="DECIMAL_NUMBER" value="10"/>
107    <eLiterals name="SIGNED_NUMBER" value="11"/>
108    <eLiterals name="MULTIPLE_OPTIONS" value="12"/>
109    <eLiterals name="BITMAP" value="13"/>
110  </eClassifiers>
111  <eClassifiers xsi:type="ecore:EEnum" name="Comparator">
112    <eLiterals name="GREATER_THAN"/>
113    <eLiterals name="LESS_THAN" value="1"/>
114    <eLiterals name="EQUALS_TO" value="2"/>
115  </eClassifiers>
116  <eClassifiers xsi:type="ecore:EClass" name="SizeLength">
117    <eStructuralFeatures xsi:type="ecore:EAttribute" name="comparator"
118    eType="#//Comparator"/>
119    <eStructuralFeatures xsi:type="ecore:EAttribute" name="size"
120    eType="ecore:EDatatypehttp://www.eclipse.org/emf/2002/Ecore#//EInt"/>
121  </eClassifiers>
122  <eClassifiers xsi:type="ecore:EClass" name="Start">
123    <eStructuralFeatures xsi:type="ecore:EReference" name="connect"
124    lowerBound="1" upperBound="-1" eType="#//MPMLComponent"/>
125  </eClassifiers>
126  <eClassifiers xsi:type="ecore:EClass" name="MotionSensor">
127    eSuperTypes="#//MPMLInputComponent#//MPMLModeledComponent">
128    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
129    eType="#//MotionSensorType"/>
130  </eClassifiers>
131  <eClassifiers xsi:type="ecore:EClass" name="NFCWriter">

```

```

132 eSuperTypes="//DataStorage"/>
133 <eClassifiers xsi:type="ecore:EClass" name="NFCReader"
134 eSuperTypes="//MPMLInputComponent//MPMLModeledComponent"/>
135 <eClassifiers xsi:type="ecore:EClass" name="RealTimeDatabase"
136 eSuperTypes="//DataStorage"/>
137 <eClassifiers xsi:type="ecore:EEnum" name="UpdateOperation">
138   <eLiterals name="UPDATE" value="1"/>
139   <eLiterals name="DELETE" value="2"/>
140 </eClassifiers>
141 <eClassifiers xsi:type="ecore:EClass" name="DataStorage"
142 abstract="true"/>
143 <eClassifiers xsi:type="ecore:EClass" name="DataConnector"
144 eSuperTypes="//Conecator">
145   <eStructuralFeatures xsi:type="ecore:EReference" name="connect"
146     eType="//DataStorage"/>
147 </eClassifiers>
148 <eClassifiers xsi:type="ecore:EClass" name="Conecator" abstract="true">
149   <eStructuralFeatures xsi:type="ecore:EReference" name="transform"
150     lowerBound="1" eType="//Transformator" containment="true"/>
151 </eClassifiers>
152 <eClassifiers xsi:type="ecore:EClass" name="UIList"
153 eSuperTypes="//MPMLOutputComponent//MPMLModeledComponent"/>
154 <eClassifiers xsi:type="ecore:EClass" name="UIcreateForm"
155 eSuperTypes="//MPMLModeledComponent//MPMLInputComponent"/>
156 <eClassifiers xsi:type="ecore:EClass" name="PositionSensor"
157 eSuperTypes="//MPMLInputComponent//MPMLModeledComponent">
158   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
159     eType="//PositionSensorType"/>
160 </eClassifiers>
161 <eClassifiers xsi:type="ecore:EEnum" name="MotionSensorType">
162   <eLiterals name="TYPE_ACCELEROMETER"/>
163   <eLiterals name="TYPE_ACCELEROMETER_UNCALIBRATED" value="1"/>
164   <eLiterals name="TYPE_GRAVITY" value="2"/>
165   <eLiterals name="TYPE_GYROSCOPE" value="3"/>
166   <eLiterals name="TYPE_GYROSCOPE_UNCALIBRATED" value="4"/>
167   <eLiterals name="TYPE_LINEAR_ACCELERATION" value="5"/>
168   <eLiterals name="TYPE_ROTATION_VECTOR" value="6"/>
169   <eLiterals name="TYPE_SIGNIFICANT_MOTION" value="7"/>
170   <eLiterals name="TYPE_STEP_COUNTER" value="8"/>
171   <eLiterals name="TYPE_STEP_DETECTOR" value="9"/>
172 </eClassifiers>
173 <eClassifiers xsi:type="ecore:EEnum" name="PositionSensorType">
174   <eLiterals name="TYPE_GAME_ROTATION_VECTOR"/>
175   <eLiterals name="TYPE_GEOMAGNETIC_ROTATION_VECTOR" value="1"/>
176   <eLiterals name="TYPE_MAGNETIC_FIELD" value="2"/>
177   <eLiterals name="TYPE_MAGNETIC_FIELD_UNCALIBRATED" value="3"/>
178   <eLiterals name="TYPE_ORIENTATION" value="4"/>
179   <eLiterals name="TYPE_PROXIMITY" value="5"/>
180 </eClassifiers>
181 <eClassifiers xsi:type="ecore:EClass" name="EnvironmentSensor"
182 eSuperTypes="//MPMLInputComponent//MPMLModeledComponent">

```

```

183     <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
184       eType="#//EnvironmentSensorType"/>
185   </eClassifiers>
186   <eClassifiers xsi:type="ecore:EEnum" name="EnvironmentSensorType">
187     <eLiterals name="TYPE_AMBIENT_TEMPERATURE"
188       literal="TYPE_AMBIENT_TEMPERATURE"/>
189     <eLiterals name="TYPE_LIGHT" value="1"/>
190     <eLiterals name="TYPE_PRESSURE" value="2"/>
191     <eLiterals name="TYPE_RELATIVE_HUMIDITY" value="3"/>
192     <eLiterals name="TYPE_TEMPERATURE" value="4"/>
193   </eClassifiers>
194   <eClassifiers xsi:type="ecore:EClass" name="Model"
195     eSuperTypes="#//Entity"/>
196   <eClassifiers xsi:type="ecore:EClass" name="MPMLModeledComponent"
197     abstract="true">
198     <eStructuralFeatures xsi:type="ecore:EReference" name="model"
199       eType="#//Model" containment="true"/>
200   </eClassifiers>
201   <eClassifiers xsi:type="ecore:EClass" name="PhoneCaller"
202     eSuperTypes="#//MPMLOutputComponent#//MPMLModeledComponent"/>
203   <eClassifiers xsi:type="ecore:EClass" name="SMSSender"
204     eSuperTypes="#//MPMLOutputComponent#//MPMLModeledComponent"/>
205 </ecore:EPackage>

```



# Anexo B. Metamodelo de arquitectura

El listado B.1 presenta el código fuente en formato XML del metamodelo de arquitectura.

Listado B.1: Metamodelo de arquitectura XMI

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="architecture"
5 nsURI="http://danielsan.com/mpml/architecture" nsPrefix="architecture">
6   <eClassifiers xsi:type="ecore:EClass" name="App">
7     <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
8       eType="ecore:EDatatype
9       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
10    <eStructuralFeatures xsi:type="ecore:EAttribute" name="namespace"
11      eType="ecore:EDatatype
12      http://www.eclipse.org/emf/2002/Ecore#//EString"/>
13    <eStructuralFeatures xsi:type="ecore:EReference" name="start"
14      lowerBound="1" upperBound="-1" eType="#//Start" containment="true"/>
15  </eClassifiers>
16  <eClassifiers xsi:type="ecore:EEnum" name="KotlinType">
17    <eLiterals name="STRING" literal="String"/>
18    <eLiterals name="DATE" value="2" literal="LocalDate"/>
19    <eLiterals name="INT" value="3" literal="Int"/>
20    <eLiterals name="DOUBLE" value="4" literal="Double"/>
21    <eLiterals name="BITMAP" value="4" literal="Any"/>
22  </eClassifiers>
23  <eClassifiers xsi:type="ecore:EEnum" name="FieldType">
24    <eLiterals name="TEXT"/>
25    <eLiterals name="EMAIL" value="1"/>
26    <eLiterals name="PASSWORD" value="2"/>
27    <eLiterals name="PASSWORD_NUMERIC" value="3"/>
28    <eLiterals name="PHONE" value="4"/>
29    <eLiterals name="POSTAL_ADDRESS" value="5"/>
```

```

30     <eLiterals name="MULTILINE" value="6"/>
31     <eLiterals name="TIME" value="7"/>
32     <eLiterals name="DATE" value="8"/>
33     <eLiterals name="NUMBER" value="9"/>
34     <eLiterals name="NUMBER_SIGNED" value="10"/>
35     <eLiterals name="NUMBER_DECIMAL" value="11"/>
36 </eClassifiers>
37 <eClassifiers xsi:type="ecore:EEnum" name="Comparator">
38     <eLiterals name="GREATER_THAN"/>
39     <eLiterals name="LESS_THAN" value="1"/>
40     <eLiterals name="EQUALS_TO" value="2"/>
41 </eClassifiers>
42 <eClassifiers xsi:type="ecore:EEnum" name="Operation">
43     <eLiterals name="CREATE"/>
44     <eLiterals name="UPDATE" value="1"/>
45     <eLiterals name="DELETE" value="2"/>
46 </eClassifiers>
47 <eClassifiers xsi:type="ecore:EEnum" name="CRUDOperation">
48     <eLiterals name="CREATE"/>
49     <eLiterals name="READ" value="1"/>
50     <eLiterals name="UPDATE" value="2"/>
51     <eLiterals name="DELETE" value="3"/>
52 </eClassifiers>
53 <eClassifiers xsi:type="ecore:EEnum" name="DataOperation">
54     <eLiterals name="PASS_THROUGH"/>
55     <eLiterals name="MERGE" value="1"/>
56     <eLiterals name="CONCAT" value="2"/>
57     <eLiterals name="JSON_ENCODE" value="3"/>
58     <eLiterals name="JSON_DECODE" value="4" literal="JSON_DECODE"/>
59 </eClassifiers>
60 <eClassifiers xsi:type="ecore:EClass" name="RealTimeDatabase"
61 eSuperTypes="#//OutputPeripheral">
62     <eStructuralFeatures xsi:type="ecore:EAttribute" name="operation"
63     eType="#//CRUDOperation"/>
64 </eClassifiers>
65 <eClassifiers xsi:type="ecore:EClass" name="Peripheral" abstract="true">
66     <eAnnotations source="http://www.obeo.fr/dsl/dnc/archetype">
67         <details key="archetype" value="Description"/>
68     </eAnnotations>
69 </eClassifiers>
70 <eClassifiers xsi:type="ecore:EClass" name="UseCaseInputData"
71 eSuperTypes="#//Entity">
72     <eAnnotations source="http://www.obeo.fr/dsl/dnc/archetype">
73         <details key="archetype" value="Thing"/>
74     </eAnnotations>
75 </eClassifiers>
76 <eClassifiers xsi:type="ecore:EClass" name="UseCaseOutputData"
77 eSuperTypes="#//Entity">
78     <eAnnotations source="http://www.obeo.fr/dsl/dnc/archetype">
79         <details key="archetype" value="Thing"/>
80     </eAnnotations>

```

```

81 </eClassifiers>
82 <eClassifiers xsi:type="ecore:EClass" name="UseCaseInteractor">
83   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
84     eType="ecore:EDatatype
85       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
86   <eStructuralFeatures xsi:type="ecore:EReference" name="input"
87     lowerBound="1" eType="#//UseCaseInputData" containment="true"/>
88   <eStructuralFeatures xsi:type="ecore:EReference" name="output"
89     lowerBound="1" eType="#//UseCaseOutputData" containment="true"/>
90   <eStructuralFeatures xsi:type="ecore:EReference" name="gateway"
91     eType="#//Gateway"/>
92   <eStructuralFeatures xsi:type="ecore:EReference" name="presenter"
93     eType="#//Presenter"/>
94 </eClassifiers>
95 <eClassifiers xsi:type="ecore:EClass" name="Connector" abstract="true">
96   <eStructuralFeatures xsi:type="ecore:EReference" name="transform"
97     lowerBound="1" eType="#//Transformator" containment="true"/>
98 </eClassifiers>
99 <eClassifiers xsi:type="ecore:EClass" name="Controller"
100   eSuperTypes="#//Connector">
101   <eStructuralFeatures xsi:type="ecore:EReference" name="usecase"
102     lowerBound="1" eType="#//UseCaseInteractor"/>
103   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
104     eType="ecore:EDatatype
105       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
106   <eStructuralFeatures xsi:type="ecore:EReference" name="input"
107     lowerBound="1" eType="#//ControllerOutputData" containment="true"/>
108 </eClassifiers>
109 <eClassifiers xsi:type="ecore:EClass" name="Gateway"
110   eSuperTypes="#//Connector">
111   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
112     eType="ecore:EDatatype
113       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
114   <eStructuralFeatures xsi:type="ecore:EReference" name="output"
115     lowerBound="1" eType="#//GatewayInputData" containment="true"/>
116   <eStructuralFeatures xsi:type="ecore:EReference" name="peripheral"
117     lowerBound="1" eType="#//OutputPeripheral"/>
118 </eClassifiers>
119 <eClassifiers xsi:type="ecore:EClass" name="Transformator">
120   <eStructuralFeatures xsi:type="ecore:EAttribute" name="operation"
121     eType="#//DataOperation"/>
122   <eStructuralFeatures xsi:type="ecore:EReference" name="transform"
123     upperBound="-1" eType="#//Transformator" containment="true"/>
124 </eClassifiers>
125 <eClassifiers xsi:type="ecore:EClass" name="GatewayInputData"
126   eSuperTypes="#//Entity">
127   <eAnnotations source="http://www.obeo.fr/dsl/dnc/archetype">
128     <details key="archetype" value="Thing"/>
129   </eAnnotations>
130 </eClassifiers>
131 <eClassifiers xsi:type="ecore:EClass" name="ControllerOutputData"

```

```

132 eSuperTypes="#//Entity">
133   <eAnnotations source="http://www.obeo.fr/dsl/dnc/archetype">
134     <details key="archetype" value="Thing"/>
135   </eAnnotations>
136 </eClassifiers>
137 <eClassifiers xsi:type="ecore:EClass" name="Start">
138   <eStructuralFeatures xsi:type="ecore:EReference" name="connect"
139     lowerBound="1" upperBound="-1" eType="#//Peripheral"/>
140 </eClassifiers>
141 <eClassifiers xsi:type="ecore:EClass" name="Spinner"
142 eSuperTypes="#//BindedWidget"/>
143 <eClassifiers xsi:type="ecore:EClass" name="TextField"
144 eSuperTypes="#//BindedWidget">
145   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
146     eType="#//FieldType"/>
147 </eClassifiers>
148 <eClassifiers xsi:type="ecore:EClass" name="Form"
149 eSuperTypes="#//Container">
150   <eStructuralFeatures xsi:type="ecore:EReference" name="widgets"
151     upperBound="-1" eType="#//Widget" containment="true"/>
152   <eStructuralFeatures xsi:type="ecore:EAttribute" name="operation"
153     eType="#//Operation"/>
154 </eClassifiers>
155 <eClassifiers xsi:type="ecore:EClass" name="Screen"
156 eSuperTypes="#//InputPeripheral">
157   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
158     eType="ecore:EDataType
159     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
160   <eStructuralFeatures xsi:type="ecore:EAttribute" name="title"
161     eType="ecore:EDataType
162     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
163   <eStructuralFeatures xsi:type="ecore:EReference" name="containers"
164     upperBound="-1" eType="#//Container" containment="true"/>
165 </eClassifiers>
166 <eClassifiers xsi:type="ecore:EClass" name="SaveButton"
167 eSuperTypes="#//UnBindedWidget"/>
168 <eClassifiers xsi:type="ecore:EClass" name="BindedWidget"
169 abstract="true" eSuperTypes="#//Widget">
170   <eStructuralFeatures xsi:type="ecore:EReference" name="bind"
171     lowerBound="1" eType="#//Attribute"/>
172 </eClassifiers>
173 <eClassifiers xsi:type="ecore:EClass" name="List"
174 eSuperTypes="#//Container"/>
175 <eClassifiers xsi:type="ecore:EClass" name="Container" abstract="true"/>
176 <eClassifiers xsi:type="ecore:EClass" name="CancelButton"
177 eSuperTypes="#//UnBindedWidget"/>
178 <eClassifiers xsi:type="ecore:EClass" name="Widget" abstract="true">
179   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
180     eType="ecore:EDataType
181     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
182 </eClassifiers>

```

```

183 <eClassifiers xsi:type="ecore:EClass" name="UnBindedWidget"
184 abstract="true" eSuperTypes="#//Widget">
185   <StructuralFeatures xsi:type="ecore:EAttribute" name="text"
186     eType="ecore:EDatatype
187     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
188 </eClassifiers>
189 <eClassifiers xsi:type="ecore:EClass" name="Label"
190 eSuperTypes="#//UnBindedWidget"/>
191 <eClassifiers xsi:type="ecore:EClass" name="OutputPeripheral"
192 abstract="true" eSuperTypes="#//Peripheral">
193   <StructuralFeatures xsi:type="ecore:EReference" name="input"
194     eType="#//GatewayInputData"/>
195 </eClassifiers>
196 <eClassifiers xsi:type="ecore:EClass" name="InputPeripheral"
197 abstract="true" eSuperTypes="#//Peripheral">
198   <StructuralFeatures xsi:type="ecore:EReference" name="output"
199     eType="#//ControllerOutputData"/>
200   <StructuralFeatures xsi:type="ecore:EReference" name="controller"
201     eType="#//Controller"/>
202   <StructuralFeatures xsi:type="ecore:EReference" name="viewmodel"
203     eType="#//ViewModel"/>
204   <StructuralFeatures xsi:type="ecore:EReference"
205     name="embeddedconnector" eType="#//EmbeddedConnector"/>
206 </eClassifiers>
207 <eClassifiers xsi:type="ecore:EClass" name="Presenter">
208   <StructuralFeatures xsi:type="ecore:EAttribute" name="name"
209     eType="ecore:EDatatype
210     http://www.eclipse.org/emf/2002/Ecore#//EString"/>
211   <StructuralFeatures xsi:type="ecore:EReference" name="viewmodel"
212     eType="#//ViewModel"/>
213   <StructuralFeatures xsi:type="ecore:EReference" name="peripheral"
214     eType="#//Peripheral"/>
215 </eClassifiers>
216 <eClassifiers xsi:type="ecore:EClass" name="ViewModel"
217 eSuperTypes="#//Entity"/>
218 <eClassifiers xsi:type="ecore:EClass" name="NFCReader"
219 eSuperTypes="#//InputPeripheral"/>
220 <eClassifiers xsi:type="ecore:EClass" name="NFCWriter"
221 eSuperTypes="#//EmbeddedPeripheral"/>
222 <eClassifiers xsi:type="ecore:EClass" name="EmbeddedPeripheral"
223 abstract="true">
224   <StructuralFeatures xsi:type="ecore:EReference" name="inputModel"
225     eType="#//ViewModel"/>
226 </eClassifiers>
227 <eClassifiers xsi:type="ecore:EClass" name="EmbeddedConnector"
228 eSuperTypes="#//Connector">
229   <StructuralFeatures xsi:type="ecore:EReference" name="connect"
230     eType="#//EmbeddedPeripheral"/>
231 </eClassifiers>
232 <eClassifiers xsi:type="ecore:EClass" name="PhoneCaller"
233 eSuperTypes="#//EmbeddedPeripheral"/>

```

```

234 <eClassifiers xsi:type="ecore:EClass" name="SMSSender"
235 eSuperTypes="#//EmbeddedPeripheral"/>
236 <eClassifiers xsi:type="ecore:EClass" name="Vibrator"
237 eSuperTypes="#//EmbeddedPeripheral"/>
238 <eClassifiers xsi:type="ecore:EClass" name="GPS"
239 eSuperTypes="#//EmbeddedPeripheral#//InputPeripheral"/>
240 <eClassifiers xsi:type="ecore:EClass" name="MotionSensor"
241 eSuperTypes="#//InputPeripheral">
242   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
243     eType="#//MotionSensorType"/>
244 </eClassifiers>
245 <eClassifiers xsi:type="ecore:EEnum" name="MotionSensorType">
246   <eLiterals name="TYPE_ACCELEROMETER"/>
247   <eLiterals name="TYPE_ACCELEROMETER_UNCALIBRATED" value="1"/>
248   <eLiterals name="TYPE_GRAVITY" value="2"/>
249   <eLiterals name="TYPE_GYROSCOPE" value="3"/>
250   <eLiterals name="TYPE_GYROSCOPE_UNCALIBRATED" value="4"/>
251   <eLiterals name="TYPE_LINEAR_ACCELERATION" value="5"/>
252   <eLiterals name="TYPE_ROTATION_VECTOR" value="6"/>
253   <eLiterals name="TYPE_SIGNIFICANT_MOTION" value="7"/>
254   <eLiterals name="TYPE_STEP_COUNTER" value="8"/>
255   <eLiterals name="TYPE_STEP_DETECTOR" value="9"/>
256 </eClassifiers>
257 <eClassifiers xsi:type="ecore:EClass" name="PositionSensor"
258 eSuperTypes="#//InputPeripheral">
259   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
260     eType="#//PositionSensorType"/>
261 </eClassifiers>
262 <eClassifiers xsi:type="ecore:EEnum" name="PositionSensorType">
263   <eLiterals name="TYPE_GAME_ROTATION_VECTOR"/>
264   <eLiterals name="TYPE_GEOMAGNETIC_ROTATION_VECTOR" value="1"/>
265   <eLiterals name="TYPE_MAGNETIC_FIELD" value="2"/>
266   <eLiterals name="TYPE_MAGNETIC_FIELD_UNCALIBRATED" value="3"/>
267   <eLiterals name="TYPE_ORIENTATION" value="4"/>
268   <eLiterals name="TYPE_PROXIMITY" value="5"/>
269 </eClassifiers>
270 <eClassifiers xsi:type="ecore:EClass" name="EnvironmentSensor"
271 eSuperTypes="#//InputPeripheral">
272   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
273     eType="#//EnvironmentSensorType"/>
274 </eClassifiers>
275 <eClassifiers xsi:type="ecore:EEnum" name="EnvironmentSensorType">
276   <eLiterals name="TYPE_AMBIENT_TEMPERATURE"
277     literal="TYPE_AMBIENT_TEMPERATURE"/>
278   <eLiterals name="TYPE_LIGHT" value="1"/>
279   <eLiterals name="TYPE_PRESSURE" value="2"/>
280   <eLiterals name="TYPE_RELATIVE_HUMIDITY" value="3"/>
281   <eLiterals name="TYPE_TEMPERATURE" value="4"/>
282 </eClassifiers>
283 <eClassifiers xsi:type="ecore:EClass" name="Camera"
284 eSuperTypes="#//EmbeddedPeripheral#//InputPeripheral"/>

```

```

285 <eClassifiers xsi:type="ecore:EClass" name="Entity" abstract="true">
286   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
287     eType="ecore:EDatatype
288       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
289   <eStructuralFeatures xsi:type="ecore:EReference" name="attributes"
290     upperBound="-1" eType="#//Attribute" containment="true"/>
291 </eClassifiers>
292 <eClassifiers xsi:type="ecore:EClass" name="Attribute">
293   <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
294     eType="ecore:EDatatype
295       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
296   <eStructuralFeatures xsi:type="ecore:EAttribute" name="readableName"
297     eType="ecore:EDatatype
298       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
299   <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
300     eType="#//KotlinType"/>
301   <eStructuralFeatures xsi:type="ecore:EReference" name="validator"
302     eType="#//Validator" containment="true"/>
303   <eStructuralFeatures xsi:type="ecore:EAttribute" name="options"
304     upperBound="-1" eType="ecore:EDatatype
305       http://www.eclipse.org/emf/2002/Ecore#//EString"/>
306 </eClassifiers>
307 <eClassifiers xsi:type="ecore:EClass" name="Validator">
308   <eStructuralFeatures xsi:type="ecore:EAttribute" name="isRequired"
309     eType="ecore:EDatatype
310       http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
311   <eStructuralFeatures xsi:type="ecore:EReference" name="sizeLength"
312     upperBound="-1" eType="#//SizeLength" containment="true"/>
313 </eClassifiers>
314 <eClassifiers xsi:type="ecore:EClass" name="SizeLength">
315   <eStructuralFeatures xsi:type="ecore:EAttribute" name="comparator"
316     eType="#//Comparator"/>
317   <eStructuralFeatures xsi:type="ecore:EAttribute" name="size"
318     eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
319 </eClassifiers>
320 </ecore:EPackage>

```



## Anexo C. Código fuente ATL

El listado C.1 presenta el código fuente de la transformación modelo a modelo en lenguaje ATL.

Listado C.1: Código fuente ATL

```
1 -- @path Architecture =
2 /com.danielsan.mpml.architecture/model/architecture.ecore
3 -- @path Domain = /com.danielsan.mpml.domain/model/domain.ecore
4 module Domain2Architecture;
5 create OUT : Architecture from IN : Domain;
6 -- *****
7 -- ***** Helpers *****
8 -- *****
9 helper context Domain!Attribute def: getData.TypeString(): String =
10   Set{self.type} -> select(t | t.toString() = "STRING" or t.toString() =
11     "LONG_STRING" or t.toString() = "EMAIL" or t.toString() = "PASSWORD" or
12     t.toString() = "PASSWORD_NUMERIC" or t.toString() = "PHONE" or
13     t.toString() = "POSTAL_ADDRESS" or t.toString() = "TIME") -> collect(t
14     | "STRING")
15   -> union(
16     Set{self.type} -> select(t | t.toString() = "DATE") -> collect(t |
17     t.toString())
18   )
19   -> union(
20     Set{self.type} -> select(t | t.toString() = "INTEGER_NUMBER" or
21     t.toString() = "SIGNED_NUMBER") -> collect(t | "INT")
22   )
23   -> union(
24     Set{self.type} -> select(t | t.toString() = "DECIMAL_NUMBER") ->
25     collect(t | "DOUBLE")
26   )
27   -> union(
28     Set{self.type} -> select(t | t.toString() = "BITMAP") -> collect(t |
29     "BITMAP")
30   ) -> first();
31
```

```

32 helper context Domain!Attribute def: getField.TypeString(): String =
33   Set{self.type} -> select(t | t.toString() = "STRING") -> collect(t |
34     "TEXT")
35   -> union(
36     Set{self.type} -> select(t | t.toString() = "LONG_STRING") ->
37     collect(t | "MULTILINE")
38   )
39   -> union(
40     Set{self.type} -> select(t | t.toString() = "EMAIL" or t.toString() =
41     "PASSWORD" or t.toString() = "PASSWORD_NUMERIC" or t.toString() =
42     "PHONE" or t.toString() = "POSTAL_ADDRESS" or t.toString() = "TIME"
43     or t.toString() = "DATE") -> collect(t | t.toString())
44   )
45   -> union(
46     Set{self.type} -> select(t | t.toString() = "INTEGER_NUMBER") ->
47     collect(t | "NUMBER")
48   )
49   -> union(
50     Set{self.type} -> select(t | t.toString() = "DECIMAL_NUMBER") ->
51     collect(t | "NUMBER_DECIMAL")
52   )
53   -> union(
54     Set{self.type} -> select(t | t.toString() = "SIGNED_NUMBER") ->
55     collect(t | "NUMBER_SIGNED")
56   ) -> first();
57 helper context Domain!Attribute def: getField():
58 Architecture!BindedWidget =
59   if self.type.toString() = "MULTIPLE_OPTIONS" then
60     thisModule.createSpinner(self)
61   else
62     thisModule.createTextField(self)
63   endif;
64 helper context Domain!MPMLInputComponent def: getOperation(): String =
65   if self.oclIsTypeOf(Domain!UICreateForm) then
66     "CREATE"
67   else
68     "EDIT"
69   endif;
70 helper context Domain!MPMLComponent def: getDataConnector():
71 Domain!DataConnector =
72   if self.oclIsKindOf(Domain!MPMLOutputComponent) then
73     self.dataSource
74   else
75     self.dataStore
76   endif;
77 -- *****
78 -- ***** Lazy Rules *****
79 -- *****
80 -- ***** Entities *****
81 lazy rule createSizeLength {
82   from

```

```

83     s: Domain!SizeLength
84   to
85     t: Architecture!SizeLength (
86       comparator <- s.comparator,
87       size <- s.size
88     )
89   }
90   lazy rule createValidator {
91     from
92       s: Domain!Validator
93     to
94       t: Architecture!Validator (
95         isRequired <- s.isRequired,
96         sizeLength <- s.sizeLengths -> collect(s |
97           thisModule.createSizeLength(s))
98       )
99   }
100   -- ***** /Entities *****
101   -- ***** UseCases *****
102   unique lazy rule createUseCaseInteractorFromFlowConnector {
103     from
104       sIn : Domain!MPMLInputOutputComponent
105     to
106       t : Architecture!UseCaseInteractor (
107         gateway <- Set{sIn.connect} -> select(con | con.oclIsUndefined() =
108           false) -> collect(con |
109             thisModule.createGateWayFromFlowConnector(con)) -> first(),
110         input <- thisModule.createUseCaseInputData(sIn.model),
111         name <- sIn.model.name.toString() + "UseCase",
112         output <- thisModule.createUseCaseOutputData(sIn.model),
113         presenter <- thisModule.createPresenter(sIn)
114       )
115   }
116   unique lazy rule createUseCaseInteractorFromDataConnector {
117     from
118       s : Domain!MPMLComponent
119     to
120       t : Architecture!UseCaseInteractor (
121         gateway <- thisModule.createGateWayFromDataConnector(s),
122         input <- thisModule.createUseCaseInputData(s.model),
123         name <- s.model.name.toString() + "UseCase",
124         output <- thisModule.createUseCaseOutputData(s.model),
125         presenter <- Set{s} -> select(s |
126           s.oclIsTypeOf(Domain!UICreateForm) or s.oclIsTypeOf(Domain!UIList))
127         -> collect(p | thisModule.createPresenter(p)) -> first() -- Just
128         generate the Presenter for the UI Components.
129       )
130   }
131   unique lazy rule createUseCaseInputAttribute {
132     from
133       s: Domain!Attribute

```

```

134   to
135     attribute: Architecture!Attribute (
136       name <- s.name,
137       options <- s.options,
138       readableName <- s.name,
139       type <- s.getDataTypeString(),
140       validator <- Set{s.validatedBy} -> select(v | v.ocllIsUndefined() =
141         false) -> collect(v | thisModule.createValidator(v)) -> first()
142     )
143   }
144   lazy rule createUseCaseInputData {
145     from
146       s: Domain!Output
147     to
148       entity: Architecture!UseCaseInputData (
149         name <- s.name,
150         attributes <- s.attributes -> collect(a |
151           thisModule.createUseCaseInputAttribute(a))
152       )
153   }
154   unique lazy rule createUseCaseOutputAttribute {
155     from
156       s: Domain!Attribute
157     to
158       attribute: Architecture!Attribute (
159         name <- s.name,
160         options <- s.options,
161         readableName <- s.name,
162         type <- s.getDataTypeString(),
163         validator <- Set{s.validatedBy} -> select(v | v.ocllIsUndefined() =
164           false) -> collect(v | thisModule.createValidator(v)) -> first()
165       )
166   }
167   lazy rule createUseCaseOutputData {
168     from
169       s: Domain!Output
170     to
171       entity: Architecture!UseCaseOutputData (
172         name <- s.name,
173         attributes <- s.attributes -> collect(a |
174           thisModule.createUseCaseOutputAttribute(a))
175       )
176   }
177   -- ***** /UseCases *****
178   -- ***** Controller *****
179   unique lazy rule createDefaultTransformator {
180     from
181       s: Domain!Connector
182     to
183       t: Architecture!Transformator (
184         operation <- "PASS_THROUGH"

```

```

185         -- transform <- IT IS NOT NECESSARY TO SET THIS ONE.
186     )
187 }
188 unique lazy rule createControllerFromDataConnector {
189     from
190         s : Domain!MPMLComponent
191     to
192         sm : Architecture!Controller (
193             input <- thisModule.createControllerOutputData(s.model),
194             name <- s.model.name.toString() + "Controller",
195             usecase <- thisModule.createUseCaseInteractorFromDataConnector(s),
196             transform <- thisModule.createDefaultTransformator(
197                 s.getDataConnector())
198         )
199 }
200 unique lazy rule createControllerFromFlowConnector {
201     from
202         sUI : Domain!MPMLInputComponent
203     to
204         sm : Architecture!Controller (
205             input <- thisModule.createControllerOutputData(sUI.model),
206             name <- sUI.model.name.toString() + "Controller",
207             usecase <-
208                 thisModule.createUseCaseInteractorFromFlowConnector(sUI),
209             transform <- thisModule.createDefaultTransformator(sUI.connect)
210         )
211 }
212 unique lazy rule createControllerAttribute {
213     from
214         s: Domain!Attribute
215     to
216         attribute: Architecture!Attribute (
217             name <- s.name,
218             options <- s.options,
219             readableName <- s.name,
220             type <- s.getDataTypeString(),
221             validator <- Set{s.validatedBy} -> select(v | v.ocIsUndefined() =
222                 false) -> collect(v | thisModule.createValidator(v)) -> first()
223         )
224 }
225 unique lazy rule createControllerOutputData {
226     from
227         s: Domain!Model
228     to
229         entity: Architecture!ControllerOutputData (
230             name <- s.name,
231             attributes <- s.attributes -> collect(a |
232                 thisModule.createControllerAttribute(a))
233         )
234 }
235 -- ***** /Controller *****

```

```

236
237 -- ***** Presenter *****
238 unique lazy rule createViewModel {
239     from
240         s: Domain!Output
241     to
242         entity: Architecture!ViewModel (
243             name <- s.name,
244             attributes <- s.attributes -> collect(a |
245                 thisModule.createViewModelAttribute(a))
246         )
247 }
248 unique lazy rule createViewModelAttribute {
249     from
250         s: Domain!Attribute
251     to
252         attribute: Architecture!Attribute (
253             name <- s.name,
254             options <- s.options,
255             readableName <- s.name,
256             type <- s.getDataTypeString(),
257             validator <- Set{s.validatedBy} -> select(v | v.ocIsUndefined() =
258                 false) -> collect(v | thisModule.createValidator(v)) -> first()
259         )
260 }
261 lazy rule createPresenter {
262     from
263         sIn : Domain!InputOutputPeripheral
264     to
265         t : Architecture!Presenter (
266             name <- sIn.model.name + "Presenter",
267             peripheral <- sIn,
268             viewModel <- thisModule.createViewModel(sIn.model)
269         )
270 }
271 -- ***** /Presenter *****
272 -- ***** Gateway *****
273 lazy rule createGatewayFromDataConnector {
274     from
275         s : Domain!MPMLComponent
276     to
277         t : Architecture!Gateway (
278             name <- s.model.name + "Gateway",
279             output <- thisModule.createGatewayInputData(s.model),
280             peripheral <- Set{s} -> select(p |
281                 p.getDataConnector().connect.ocIsTypeOf( Domain!NFCWriter)) ->
282                 collect(p | thisModule.createDataSourceNFCWriter(p)) -> union(
283                     Set{s} -> select(p | p.getDataConnector().connect.ocIsTypeOf(
284                         Domain!RealTimeDatabase)) -> collect(p |
285                         thisModule.createDataSourceRealTimeDatabase(p))
286                 ) -> first(),

```

```

287     transform <- Set{s.getDataConnector()} -> select(c |
288       c.oclIsUndefined() = false) -> collect(c | c.transform) -> first()
289   )
290 }
291 lazy rule createGateWayFromFlowConnector {
292   from
293     sCon : Domain!FlowConnector
294   to
295     t : Architecture!Gateway (
296       name <- sCon.connect.model.name + "Gateway",
297       output <- thisModule.createGatewayInputData(sCon.connect.model),
298       peripheral <- sCon.connect,
299       transform <- sCon.transform
300     )
301 }
302 unique lazy rule createGatewayAttribute {
303   from
304     s: Domain!Attribute
305   to
306     attribute: Architecture!Attribute (
307       name <- s.name,
308       options <- s.options,
309       readableName <- s.name,
310       type <- s.getDataTypeString(),
311       validator <- Set{s.validatedBy} -> select(v | v.oclIsUndefined() =
312         false) -> collect(v | thisModule.createValidator(v)) -> first()
313     )
314 }
315 unique lazy rule createGatewayInputData {
316   from
317     s: Domain!Output
318   to
319     entity: Architecture!GatewayInputData (
320       name <- s.name,
321       attributes <- s.attributes -> collect(a |
322         thisModule.createGatewayAttribute(a))
323     )
324 }
325 -- ***** /Gateway *****
326 -- ***** UI *****
327 lazy rule createList {
328   from
329     sUI : Domain!UIElementReader
330   to
331     sm : Architecture!List (
332     )
333 }
334 lazy rule createForm {
335   from
336     sUI : Domain!UICreateForm
337   to

```

```

338     sm : Architecture!Form (
339         operation <- sUI.getOperation(),
340         widgets <- Set{sUI} -> select(cf | cf.model.ocIsUndefined() =
341             false) -> collect(cf | cf.model.attributes) -> collect(a |
342             a.getField())
343         -> union(
344             Set{ thisModule.createSaveButton(sUI) }
345         )
346     )
347 }
348 lazy rule createSaveButton {
349     from
350     sUI : Domain!UICreateForm
351     to
352     sm : Architecture!SaveButton (
353         name <- "btnSave",
354         text <- Set{sUI} -> select(s | s.model.ocIsUndefined() = false) ->
355         collect(s | "Save " + sUI.model.name) -> union(
356             Set{sUI} -> select(s | s.model.ocIsUndefined() = true) ->
357             collect(s | "Continue")
358         ) -> first()
359     )
360 }
361 lazy rule createSpinner {
362     from
363     a : Domain!Attribute
364     to
365     sm : Architecture!Spinner (
366         name <- "sp" + a.name,
367         bind <- thisModule.createViewModelAttribute(a)
368     )
369 }
370 lazy rule createTextField {
371     from
372     a : Domain!Attribute
373     to
374     sm : Architecture!TextField (
375         name <- "txt" + a.name,
376         type <- a.getFieldTypeString(),
377         bind <- thisModule.createViewModelAttribute(a)
378     )
379 }
380 -- ***** /UI *****
381 -- ***** RTD *****
382 lazy rule createDataSourceRealTimeDatabase {
383     from
384     sUI : Domain!UIElementReader
385     to
386     screen: Architecture!RealTimeDatabase (
387         input <- thisModule.createGatewayInputData(sUI.model),
388         operation <- "READ"

```

```

389     )
390 }
391 -- ***** /RTD *****
392 -- ***** Embedded Connector *****
393 lazy rule createEmbeddedConnectorFromDataConnector {
394     from
395         s: Domain!MPMLInputComponent
396     to
397         t: Architecture!EmbeddedConnector (
398             connect <- Set{s} -> select(p |
399                 p.getDataConnector().connect.oclIsTypeOf( Domain!NFCWriter)) ->
400                 collect(p | thisModule.createDataSourceNFCWriter(p)) -> first(),
401             transform <- s.getDataConnector().transform
402         )
403 }
404 lazy rule createEmbeddedConnectorFromFlowConnector {
405     from
406         s: Domain!MPMLInputComponent
407     to
408         t: Architecture!EmbeddedConnector (
409             connect <- Set{s} -> select(p |
410                 p.connect.connect.oclIsTypeOf(Domain!PhoneCaller)) -> collect(p |
411                 thisModule.createDataSourcePhoneCaller(p)) -> union(
412                     Set{s} -> select(p |
413                         p.connect.connect.oclIsTypeOf(Domain!SMSSender)) -> collect(p |
414                         thisModule.createDataSourceSMSSender(p))
415                     ) -> union(
416                         Set{s} -> select(p |
417                             p.connect.connect.oclIsTypeOf(Domain!Vibrator)) -> collect(p |
418                             thisModule.createDataSourceVibrator(p))
419                     ) -> union(
420                         Set{s} -> select(p | p.connect.connect.oclIsTypeOf(Domain!GPS)) -
421                         > collect(p | thisModule.createDataSourceGPS(p.connect.connect))
422                     ) -> union(
423                         Set{s} -> select(p |
424                             p.connect.connect.oclIsTypeOf(Domain!Camera)) -> collect(p |
425                             thisModule.createDataSourceCamera(p.connect.connect))
426                     ) -> first(),
427             transform <- s.connect.transform
428         )
429 }
430 -- ***** /Embedded Connector *****
431 -- ***** NFC *****
432 lazy rule createDataSourceNFCWriter {
433     from
434         s: Domain!MPMLInputComponent
435     to
436         t: Architecture!NFCWriter (
437             inputModel <- thisModule.createViewModel(s.model)
438         )
439 }

```

```

440 -- ***** /NFC *****
441 -- ***** PhoneCaller *****
442 lazy rule createDataSourcePhoneCaller {
443   from
444     s: Domain!MPMLInputComponent
445   to
446     t: Architecture!PhoneCaller (
447       inputModel <- thisModule.createViewModel(s.model)
448     )
449 }
450 -- ***** /PhoneCaller *****
451 -- ***** SMSSender *****
452 lazy rule createDataSourceSMSSender {
453   from
454     s: Domain!MPMLInputComponent
455   to
456     t: Architecture!SMSSender (
457       inputModel <- thisModule.createViewModel(s.model)
458     )
459 }
460 -- ***** /SMSSender *****
461 -- ***** Vibrator *****
462 lazy rule createDataSourceVibrator {
463   from
464     s: Domain!MPMLInputComponent
465   to
466     t: Architecture!Vibrator (
467       inputModel <- thisModule.createViewModel(s.model)
468     )
469 }
470 -- ***** /Vibrator *****
471 -- ***** GPS *****
472 lazy rule createDataSourceGPS {
473   from
474     s: Domain!MPMLInputComponent
475   to
476     t: Architecture!GPS (
477       controller <- Set{s} -> select(f | f.dataStore.oclIsUndefined() =
478         false) -> select(f | f.dataStore.connect.oclIsTypeOf(
479         Domain!RealTimeDatabase)) -> collect(f |
480         thisModule.createControllerFromDataConnector(f)) -> first(),
481       inputModel <- thisModule.createViewModel(s.model),
482       output <- Set{s} -> select(f | f.dataStore.oclIsUndefined() =
483         false) -> select(f | f.dataStore.connect.oclIsTypeOf(
484         Domain!RealTimeDatabase)) -> collect(f |
485         thisModule.createControllerOutputData(s.model)) -> first()
486       -- viewmodel <- IT IS ONLY FOR THE PRESENTER
487       -- embeddedconnector <- IN CASE THIS IS CONNECTED TO OTHER EMBEDDED
488       PERIPHERAL
489     )
490 }

```

```

491 -- ***** /GPS *****
492 -- ***** Camera *****
493 lazy rule createDataSourceCamera {
494     from
495         s: Domain!MPMLInputComponent
496     to
497         t: Architecture!Camera (
498             controller <- Set{s} -> select(f | f.dataStore.oclIsUndefined() =
499             false) -> select(f | f.dataStore.connect.oclIsTypeOf(
500             Domain!RealTimeDatabase)) -> collect(f |
501             thisModule.createControllerFromDataConnector(f)) -> first(),
502             inputModel <- thisModule.createViewModel(s.model),
503             output <- Set{s} -> select(f | f.dataStore.oclIsUndefined() =
504             false) -> select(f | f.dataStore.connect.oclIsTypeOf(
505             Domain!RealTimeDatabase)) -> collect(f |
506             thisModule.createControllerOutputData(s.model)) -> first()
507             -- viewmodel <- IT IS ONLY FOR THE PRESENTER
508             -- embeddedconnector <- IN CASE THIS IS CONNECTED TO OTHER EMBEDED
509             PERIPHERAL
510         )
511 }
512 -- ***** /Camera *****
513 -- ***** Rules *****
514 rule Transform2Transform {
515     from
516         s: Domain!Transformator
517     to
518         t: Architecture!Transformator (
519             operation <- s.operation,
520             transform <- s.transform
521         )
522 }
523 -- ***** /Rules *****
524 -- ***** UI *****
525 rule UICreateForm2ScreenForm {
526     from
527         sUI : Domain!UICreateForm
528     to
529         screen: Architecture!Screen (
530             containers <- Set{thisModule.createForm(sUI)},
531             -- ***** Clean Architecture flow *****
532             controller <- Set{sUI} -> select(f | f.dataStore.oclIsUndefined() =
533             false) -> select(f | f.dataStore.connect.oclIsTypeOf(
534             Domain!RealTimeDatabase)) -> collect(f |
535             thisModule.createControllerFromDataConnector(f)) -> first(),
536             output <- Set{sUI} -> select(f | f.dataStore.oclIsUndefined() =
537             false) -> select(f | f.dataStore.connect.oclIsTypeOf(
538             Domain!RealTimeDatabase)) -> collect(f |
539             thisModule.createControllerOutputData(sUI.model)) -> first(),
540
541     -- ***** Embedded Peripherals *****

```

```

542     embeddedconnector <- Set{sUI} -> select(f |
543     f.dataStore.oclIsUndefined() = false) -> collect(p |
544     thisModule.createEmbeddedConnectorFromDataConnector( p)) -> union(
545     Set{sUI} -> select(f | f.connect.oclIsUndefined() = false) ->
546     collect(p | thisModule.createEmbeddedConnectorFromFlowConnector(
547     p))
548   ) -> first(),
549   -- Set{sUI} -> select(f | f.dataStore.oclIsUndefined() = false) -
550   > select(f | f.dataStore.connect.oclIsTypeOf(Domain!NFCWriter)) -
551   > collect(f |
552     thisModule.createEmbeddedConnectorFromDataConnector( f)) ->
553     first(),
554   -- or f.connect.connect.oclIsTypeOf(Domain!PhoneCaller)
555   name <- Set{sUI} -> select(s | s.model.oclIsUndefined() = false) ->
556   collect(s | s.getOperation().toString().toLowerCase() +
557   s.model.name.toLowerCase()) -> union(
558     Set{sUI} -> select(s | s.model.oclIsUndefined() = true) ->
559     collect(s | "welcome")
560   ) -> first(),
561   title <- Set{sUI} -> select(s | s.model.oclIsUndefined() = false) -
562   > collect(s | s.getOperation().toString().toLowerCase() + " " +
563   s.model.name.toLowerCase()) -> union(
564     Set{sUI} -> select(s | s.model.oclIsUndefined() = false) ->
565     collect(s | "Welcome")
566   ) -> first(),
567   viewmodel <- Set{sUI} -> select(s | s.model.oclIsUndefined() =
568   false) -> collect(s | thisModule.createViewModel(s.model)) ->
569   first()
570 )
571 }
572 rule U IList2ScreenList {
573   from
574     sUI: Domain!U IList
575   to
576     screen: Architecture!Screen (
577       containers <- Set{thisModule.createList(sUI)},
578       controller <- thisModule.createControllerFromDataConnector(sUI),
579       -- input <-
580       name <- "read" + sUI.model.name.toLowerCase(),
581       -- output <-
582       title <- "read " + sUI.model.name.toLowerCase(),
583       viewmodel <- thisModule.createViewModel(sUI.model)
584     )
585 }
586 -- ***** /UI *****
587 -- ***** NFC *****
588 rule NFCReader2NFCReader {
589   from
590     s: Domain!NFCReader
591   to
592     t: Architecture!NFCReader (

```

```

593     controller <- thisModule.createControllerFromDataConnector(s),
594     output <- thisModule.createControllerOutputData(s.model),
595     viewmodel <- thisModule.createViewModel(s.model)
596   )
597 }
598 -- ***** /NFC *****
599 -- ***** Sensors *****
600 rule MotionSensor2MotionSensor {
601   from
602     s: Domain!MotionSensor
603   to
604     t: Architecture!MotionSensor (
605       controller <- thisModule.createControllerFromDataConnector(s),
606       output <- thisModule.createControllerOutputData(s.model),
607       viewmodel <- thisModule.createViewModel(s.model),
608       type <- s.type
609     )
610 }
611 rule PositionSensor2PositionSensor {
612   from
613     s: Domain!PositionSensor
614   to
615     t: Architecture!PositionSensor (
616       controller <- thisModule.createControllerFromDataConnector(s),
617       output <- thisModule.createControllerOutputData(s.model),
618       viewmodel <- thisModule.createViewModel(s.model),
619       type <- s.type
620     )
621 }
622 rule EnvironmentSensor2EnvironmentSensor {
623   from
624     s: Domain!EnvironmentSensor
625   to
626     t: Architecture!EnvironmentSensor (
627       controller <- thisModule.createControllerFromDataConnector(s),
628       output <- thisModule.createControllerOutputData(s.model),
629       viewmodel <- thisModule.createViewModel(s.model),
630       type <- s.type
631     )
632 }
633 -- ***** /Sensors *****
634 rule Start2Start {
635   from
636     s: Domain!Start
637   to
638     t: Architecture!Start (
639       connect <- s.connect
640     )
641 }
642 rule App2App {
643   from

```

```
644 |     s: Domain!App
645 |     to
646 |     t: Architecture!App (
647 |         name <- s.name,
648 |         namespace <- s.namespace,
649 |         start <- s.start
650 |     )
651 | }
```

## Anexo D. Código fuente Acceleo

El listado D.1 presenta el código fuente de la transformación modelo a texto en lenguaje Acceleo.

Listado D.1: Código fuente Acceleo

```
1 [module Routes("http://danielsan.com/mpml/architecture")]
2 [import com::danielsan::pim::architecture::gen::kotlin::common:: Names /]
3 [comment ***** ROOT Routes ***** /]
4 [query public getAppRoute() : String = "app/src/main/" /]
5 [query public getAppClassesRoute(anApp : App) : String =
6 getAppRoute().concat("java/").concat(anApp.namespace.
7 replaceAll("\\.", "/")).concat("/").concat(anApp.name).concat("/") /]
8 [query public getRESOURCESRoute() : String = getAppRoute().concat("res/")
9 /]
10 [query public getAdaptersRoute() : String = "adapters/src/main/" /]
11 [query public getAdaptersClassesRoute(anApp : App) : String =
12 getAdaptersRoute().concat("java/").concat(anApp.namespace.
13 replaceAll("\\.", "/")).concat("/") /]
14 [query public getUseCasesRoute() : String = "usecases/src/main/" /]
15 [query public getUseCasesClassesRoute(anApp : App) : String =
16 getUseCasesRoute().concat("java/").concat(anApp.namespace.
17 replaceAll("\\.", "/")).concat("/") /]
18 [query public getEntitiesRoute() : String = "entities/src/main/" /]
19 [query public getEntitiesClassesRoute(anApp : App) : String =
20 getEntitiesRoute().concat("java/").concat(anApp.namespace.
21 replaceAll("\\.", "/")).concat("/") /]
22 [comment ***** Resources Routes ***** /]
23 [query public getFrameworkUIColorsFileRoute() : String =
24 getRESOURCESRoute().concat("values/colors.xml") /]
25 [query public getFrameworkUIDimensFileRoute() : String =
26 getRESOURCESRoute().concat("values/dimens.xml") /]
27 [query public getFrameworkUIStringsFileRoute() : String =
28 getRESOURCESRoute().concat("values/strings.xml") /]
29 [query public getFrameworkUIStringsFileRoute(anAttribute: Attribute) :
30 String = getRESOURCESRoute().concat("values/").concat(
31 anAttribute.name).concat(".xml") /]
```

```

32 [query public getFrameworkUIStylesFileRoute() : String =
33 getRESOURCESRoute().concat("values/styles.xml") /]
34 [comment Layout /]
35 [query public getFrameworkUIActivityMainLayoutFileRoute() : String =
36 getRESOURCESRoute().concat("layout/").concat(
37 getFrameworkUIActivityMainLayoutName()).concat(".xml") /]
38 [query public getFrameworkUIAppBarMainLayoutFileRoute() : String =
39 getRESOURCESRoute().concat("layout/").concat(
40 getFrameworkUIAppBarMainLayoutName()).concat(".xml") /]
41 [query public getFrameworkUINavHeaderMainLayoutFileRoute() : String =
42 getRESOURCESRoute().concat("layout/").concat(
43 getFrameworkUINavHeaderMainLayoutName()).concat(".xml") /]
44 [comment Dimens /]
45 [query public getFrameworkUISideNavBarDimensFileRoute() : String =
46 getRESOURCESRoute().concat("drawable/").concat(
47 getFrameworkUISideNavBarDimensName()).concat(".xml") /]
48 [comment Menu /]
49 [query public getFrameworkUIActivityMainDrawerMenuFileRoute() : String =
50 getRESOURCESRoute().concat("menu/").concat(
51 getFrameworkUIActivityMainDrawerMenuName()).concat(".xml") /]
52 [comment Navigation /]
53 [query public getFrameworkNavigationNavGraphFileRoute() : String =
54 getRESOURCESRoute().concat("navigation/").concat(
55 getFrameworkNavigationNavGraphName()).concat(".xml") /]
56 [comment ***** Manifest Route ***** /]
57 [query public getFrameworkManifestFileRoute() : String =
58 getAppRoute().concat("AndroidManifest.xml") /]
59 [comment ***** Gradle Routes ***** /]
60 [query public getFrameworkProjectGradleRoute() : String = "build.gradle"
61 /]
62 [query public getFrameworkModuleGradleRoute() : String =
63 "app/build.gradle" /]
64 [query public getAdaptersModuleGradleRoute() : String =
65 "adapters/build.gradle" /]
66 [query public getUseCasesModuleGradleRoute() : String =
67 "usecases/build.gradle" /]
68 [query public getEntitiesModuleGradleRoute() : String =
69 "entities/build.gradle" /]
70 [comment ***** Clean Architecture Routes
71 ***** /]
72 [comment ***** Root Routes ***** /]
73 [comment adapters /]
74 [query public getAdaptersControllerRootRoute(anApp : App) : String =
75 getAdaptersClassesRoute(anApp).concat("adapters/controllers/") /]
76 [query public getAdaptersGatewayClassRootRoute(anApp : App) : String =
77 getAdaptersClassesRoute(anApp).concat("adapters/gateways/") /]
78 [query public getAdaptersPortRootRoute(anApp: App) : String =
79 getAdaptersClassesRoute(anApp).concat("adapters/port/") /]
80 [query public getAdaptersDSRootRoute(anApp : App) : String =
81 getAdaptersClassesRoute(anApp).concat("adapters/port/DS/") /]
82 [query public getAdaptersPortInputRootRoute(anApp: App) : String =

```

```

83 | getAdaptersClassesRoute(anApp).concat( "adapters/port/input/") [/]
84 | [query public getAdaptersPresenterClassRootRoute(anApp: App) : String =
85 | getAdaptersClassesRoute(anApp).concat( "adapters/presenters/") [/]
86 | [comment framework [/]
87 | [query public getFrameworkDataAccessRootRoute(anApp : App) : String =
88 | getAppClassesRoute(anApp).concat("framework/dataAccess/") [/]
89 | [query public getFrameworkPortRootRoute(anApp : App) : String =
90 | getAppClassesRoute(anApp).concat("framework/port/") [/]
91 | [query public getFrameworkUIExtensionsRootRoute(anApp : App) : String =
92 | getAppClassesRoute(anApp).concat("framework/ui/extensions/") [/]
93 | [query public getFrameworkViewRootRoute(anApp : App) : String =
94 | getAppClassesRoute(anApp).concat("framework/ui/view/") [/]
95 | [query public getFrameworkFragmentRootRoute(anApp : App) : String =
96 | getAppClassesRoute(anApp).concat( "framework/ui/view/fragments/") [/]
97 | [query public getFrameworkViewModelRootRoute(anApp : App) : String =
98 | getAppClassesRoute(anApp).concat("framework/ui/viewModel/") [/]
99 | [query public getFrameworkUIWidgetsRootRoute(anApp : App) : String =
100 | getAppClassesRoute(anApp).concat("framework/ui/widgets/") [/]
101 | [comment nfc [/]
102 | [query public getFrameworkNFCRootRoute(anApp : App) : String =
103 | getAppClassesRoute(anApp).concat("framework/nfc/") [/]
104 | [comment motionSensor [/]
105 | [query public getFrameworkMotionSensorRootRoute(anApp : App) : String =
106 | getAppClassesRoute(anApp).concat("framework/motionSensor/") [/]
107 | [comment positionSensor [/]
108 | [query public getFrameworkPositionSensorRootRoute(anApp : App) : String =
109 | getAppClassesRoute(anApp).concat( "framework/positionSensor/") [/]
110 | [comment environmentSensor [/]
111 | [query public getFrameworkEnvironmentSensorRootRoute(anApp : App) :
112 | String = getAppClassesRoute(anApp).concat(
113 | "framework/environmentSensor/") [/]
114 | [comment entities [/]
115 | [query public getEntitiesRootRoute(anApp: App) : String =
116 | getEntitiesClassesRoute(anApp).concat("entities/") [/]
117 | [comment usecases [/]
118 | [query public getUseCasesRootRoute(anApp : App) : String =
119 | getUseCasesClassesRoute(anApp).concat("usecases/") [/]
120 | [query public getUseCasesPortRootRoute(anApp : App) : String =
121 | getUseCasesClassesRoute(anApp).concat("usecases/port/") [/]
122 | [query public getUseCasesDTORootRoute(anApp: App) : String =
123 | getUseCasesClassesRoute(anApp).concat("usecases/port/DTO/") [/]
124 | [query public getUseCasesInputPortRootRoute(anApp: App) : String =
125 | getUseCasesClassesRoute(anApp).concat( "usecases/port/input/") [/]
126 | [query public getUseCasesOutputPortRootRoute(anApp: App) : String =
127 | getUseCasesClassesRoute(anApp).concat( "usecases/port/output/") [/]
128 | [comment ***** Class or Interface Routes
129 | ***** [/]
130 | [query public getFrameworkLayoutFileRoute(aScreen : Screen) : String =
131 | getRESOURCESRoute().concat("layout/").concat(aScreen.
132 | getFrameworkUILayoutName()).concat(".xml") [/]
133 | [query public getFrameworkListItemLayoutFileRoute(aScreen : Screen) :

```

```

134 String = getRESOURCESRoute().concat("layout/").concat(aScreen.
135 getFrameworkUIListItemLayoutName()).concat(".xml") /]
136 [comment adapters /]
137 [query public getAdaptersControllerClassRoute(anApp: App, aController :
138 Controller) : String = getAdaptersControllerRootRoute(anApp).concat(
139 getAdaptersControllerClassName(aController)).concat(".kt") /]
140 [query public getAdaptersGatewayClassRoute(anApp: App, aGateway: Gateway)
141 : String = getAdaptersGatewayClassRootRoute(anApp).concat(
142 getUseCasesGatewayClassName(aGateway)).concat(".kt") /]
143 [query public getAdaptersDSClassRoute(anApp: App, aControllerOutputData:
144 ControllerOutputData) : String = getAdaptersDSRootRoute(anApp).concat(
145 getAdaptersDSClassName(aControllerOutputData)).concat(".kt") /]
146 [query public getAdaptersControllerInterfaceRoute(anApp: App, aController
147 : Controller) : String = getAdaptersPortInputRootRoute(anApp).concat(
148 getAdaptersControllerInterfaceName(aController)).concat(".kt") /]
149 [query public getAdaptersDataAccessInterfaceRoute(anApp: App, anEntity:
150 Entity) : String = getAdaptersPortInputRootRoute(anApp).concat(
151 getAdaptersDAOInterfaceName(anEntity)).concat(".kt") /]
152 [query public getAdaptersUserInterfaceRoute(anApp: App) : String =
153 getAdaptersPortInputRootRoute(anApp).concat(
154 getAdaptersUserInterfaceInterfaceName()).concat(".kt") /]
155 [query public getAdaptersFactoryClassRoute(anApp: App) : String =
156 getAdaptersPortRootRoute(anApp).concat(
157 getAdaptersFactoryName()).concat(".kt") /]
158 [query public getAdaptersPresentersListClassRoute(anApp: App, aPresenter
159 : Presenter) : String = getAdaptersPresenterClassRootRoute(anApp).concat(
160 getAdaptersPresenterListClassName(aPresenter)).concat(".kt") /]
161 [query public getAdaptersPresentersRegisterClassRoute(anApp: App,
162 aPresenter: Presenter) : String =
163 getAdaptersPresenterClassRootRoute(anApp).concat(
164 getAdaptersPresenterRegisterClassName(aPresenter)).concat(".kt") /]
165 [comment framework /]
166 [query public getFrameworkDataAccessClassRoute(anApp: App,
167 aGatewayInputData : GatewayInputData) : String =
168 getFrameworkDataAccessRootRoute(anApp).concat(
169 getFrameworkDAOClassName(aGatewayInputData)).concat(".kt") /]
170 [query public getFrameworkFactoryClassRoute(anApp: App) : String =
171 getFrameworkDataAccessRootRoute(anApp).concat(
172 getFrameworkFactoryName()).concat(".kt") /]
173 [query public getFrameworkEditTextExtensionsClassRoute(aScreen : Screen,
174 anApp:App) : String = getFrameworkUIExtensionsRootRoute(anApp).concat(
175 getFrameworkUIEditTextExtensionsName()).concat(".kt") /]
176 [query public getFrameworkViewExtensionsClassRoute(aScreen : Screen,
177 anApp:App) : String = getFrameworkUIExtensionsRootRoute(anApp).concat(
178 getFrameworkUIViewExtensionsName()).concat(".kt") /]
179 [query public getFrameworkFragmentFileRoute(anApp: App, aScreen : Screen)
180 : String = getFrameworkFragmentRootRoute(anApp).concat(
181 getFrameworkUIFragmentClassName(aScreen)).concat(".kt") /]
182 [query public getFrameworkBaseViewModelFactoryRoute(aScreen : Screen,
183 anApp:App) : String = getFrameworkViewModelRootRoute(anApp).concat(
184 getFrameworkUIBaseViewModelFactoryName()).concat(".kt") /]

```

```

185 [query public getFrameworkListViewModelClassRoute(aCtrlOutputData :
186 ControllerOutputData, anApp:App) : String =
187 getFrameworkViewModelRootRoute(anApp).concat(
188 getFrameworkUIListViewModelClassName( aCtrlOutputData)).concat(".kt") /]
189 [query public getFrameworkRegisterViewModelClassRoute( aCtrlOutputData :
190 ControllerOutputData, anApp:App) : String =
191 getFrameworkViewModelRootRoute(anApp).concat(
192 getFrameworkUIRegisterViewModelClassName( aCtrlOutputData)).concat(".kt")
193 /]
194 [query public getFrameworkCustomAlertDialogFileRoute(anApp: App) : String
195 = getFrameworkUIWidgetsRootRoute(anApp).concat(
196 getFrameworkUICustomAlertDialogName()).concat(".kt") /]
197 [query public getFrameworkListAdapterFileRoute(aScreen : Screen,
198 anApp:App) : String = getFrameworkFragmentRootRoute(anApp).concat(
199 getFrameworkUIListAdapterClassName(aScreen)).concat(".kt") /]
200 [query public getFrameworkListViewHolderFileRoute(aScreen : Screen,
201 anApp:App) : String = getFrameworkFragmentRootRoute(anApp).concat(
202 getFrameworkUIListViewHolderClassName( aScreen)).concat(".kt") /]
203 [query public getFrameworkDTOClassRoute(anApp: App, aViewModel:
204 ViewModel) : String = getFrameworkPortRootRoute(anApp).concat(aViewModel.
205 getFrameworkDTOClassName()).concat(".kt") /]
206 [comment nfc /]
207 [query public getFrameworkNFCReaderClassRoute(anApp : App, aNFCReader:
208 NFCReader) : String = getFrameworkNFCRootRoute(anApp).concat(
209 getFrameworkNFCReaderClassName(aNFCReader)).concat(".kt") /]
210 [query public getFrameworkNFCWriterClassRoute(anApp : App, aNFCWriter:
211 NFCWriter) : String = getFrameworkNFCRootRoute(anApp).concat(
212 getFrameworkNFCWriterClassName(aNFCWriter)).concat(".kt") /]
213 [comment motionSensor /]
214 [query public getFrameworkMotionSensorClassRoute(anApp : App,
215 aMotionSensor: MotionSensor) : String =
216 getFrameworkMotionSensorRootRoute(anApp).concat(
217 getFrameworkMotionSensorClassName( aMotionSensor)).concat(".kt") /]
218 [comment positionSensor /]
219 [query public getFrameworkPositionSensorClassRoute(anApp : App,
220 aPositionSensor: PositionSensor) : String =
221 getFrameworkPositionSensorRootRoute(anApp).concat(
222 getFrameworkPositionSensorClassName( aPositionSensor)).concat(".kt") /]
223 [comment environmentSensor /]
224 [query public getFrameworkEnvironmentSensorClassRoute(anApp : App,
225 anEnvironmentSensor: EnvironmentSensor) : String =
226 getFrameworkEnvironmentSensorRootRoute(anApp).concat(
227 getFrameworkEnvironmentSensorClassName(
228 anEnvironmentSensor)).concat(".kt") /]
229 [comment entities /]
230 [query public getEntitiesRoute(anApp: App, anEntity: Entity) : String =
231 getEntitiesRootRoute(anApp).concat(getEntitiesClassName(
232 anEntity)).concat(".kt") /]
233 [comment usecases /]
234 [query public getUseCasesDTORoute(anApp: App, aUseCaseInputData:
235 UseCaseInputData) : String = getUseCasesDTORootRoute(anApp).concat(

```

```

236 | getUseCasesDTOClassName(aUseCaseInputData)).concat(".kt") /]
237 | [query public getUseCasesFactoryClassRoute(anApp: App) : String =
238 | getUseCasesPortRootRoute(anApp).concat(
239 | getUseCasesFactoryClassName()).concat(".kt") /]
240 | [query public getUseCasesListClassRoute(anApp: App, aUseCase:
241 | UseCaseInteractor) : String = getUseCasesRootRoute(anApp).concat(
242 | getUseCasesListClassName(aUseCase)).concat(".kt") /]
243 | [query public getUseCasesRegisterClassRoute(anApp: App, aUseCase:
244 | UseCaseInteractor) : String = getUseCasesRootRoute(anApp).concat(
245 | getUseCasesRegisterClassName(aUseCase)).concat(".kt") /]
246 | [query public getUseCasesListInterfaceRoute(anApp: App, aUseCase:
247 | UseCaseInteractor) : String =
248 | getUseCasesInputPortRootRoute(anApp).concat(
249 | getUseCasesListInterfaceName( aUseCase)).concat(".kt") /]
250 | [query public getUseCasesRegisterInterfaceRoute(anApp: App, aUseCase:
251 | UseCaseInteractor) : String =
252 | getUseCasesInputPortRootRoute(anApp).concat(
253 | getUseCasesRegisterInterfaceName( aUseCase)).concat(".kt") /]
254 | [query public getUseCasesGatewayInterfaceRoute(anApp: App, aGateway:
255 | Gateway) : String = getUseCasesOutputPortRootRoute(anApp).concat(
256 | getUseCasesGatewayInterfaceName(aGateway)).concat(".kt") /]
257 | [query public getUseCasesPresenterListInterfaceRoute(anApp: App,
258 | aPresenter: Presenter) : String =
259 | getUseCasesOutputPortRootRoute(anApp).concat(
260 | getUseCasesPresenterListInterfaceName( aPresenter)).concat(".kt") /]
261 | [query public getUseCasesPresenterRegisterInterfaceRoute(anApp: App,
262 | aPresenter: Presenter) : String =
263 | getUseCasesOutputPortRootRoute(anApp).concat(
264 | getUseCasesPresenterRegisterInterfaceName( aPresenter)).concat(".kt") /]

```

# Autores

## Héctor Arturo Flórez Fernández

Profesor titular de la Universidad Distrital Francisco José de Caldas, doctor en ingeniería de la Universidad de los Andes, magíster en ciencias de la información y las comunicaciones de la Universidad Distrital Francisco José de Caldas, magíster en gestión de organizaciones de la Universidad Militar Nueva Granada, ingeniero electrónico e ingeniero de sistemas de la Universidad El Bosque, matemático de la Fundación Universitaria Konrad Lorenz, investigador sénior de Minciencias Colombia.

✉ [haflorefz@udistrital.edu.co](mailto:haflorefz@udistrital.edu.co)

🆔 <https://orcid.org/0000-0002-5339-4459>

## Daniel Esteban Sánchez Rodríguez

Desarrollador sénior full stack en Iam Studio, profesor de cátedra de la Fundación Universitaria Konrad Lorenz, magíster en ciencias de la información y las comunicaciones e ingeniero en telemática de la Universidad Distrital Francisco José de Caldas.

✉ [desanchezr@correo.udistrital.edu.co](mailto:desanchezr@correo.udistrital.edu.co)

🆔 <https://orcid.org/0000-0001-9458-4155>

E2

E2

Este libro se terminó de imprimir en abril de 2025  
en XPRESS ESTUDIO GRÁFICO Y DIGITAL S.A.S, Bogotá, Colombia.



