MDD+

Mejorando el Proceso de Desarrollo de Software: Propuesta basada en MDD[1]

Nro. Referencia: 14-INV-056

Investigador principal: Luca Cernuzzi, Universidad Católica "Nuestra Señora de la Asunción"

Propuesta MDD para la persistencia de datos en aplicaciones nativas orientadas a teléfonos móviles inteligentes

Autor del Documento: Manuel Núñez, Universidad Católica "Nuestra Señora de la Asunción"

2 de diciembre de 2016

Asunción, Paraguay

[1]Este proyecto es financiado por el CONACYT a través del programa PROCIENCIA con recursos del Fondo para la Excelencia de la Educación e Investigación (FEEI) del FONACIDE.

Índice

1. Introducción

El auge de las aplicaciones móviles y la facilidad con la que hoy en día uno puede acceder a un teléfono móvil inteligente[2], o *smartphone* en inglés, promueve a que año tras año vaya creciendo el número de usuarios de éstos dispositivos; y de la misma forma, se produce el aumento del volumen de ventas en el mercado incentivados por la demanda y por la economía que mueve este sector tecnológico. Sin duda, este sector es de gran importancia y no está libre de problemas, sino en constante cambio y evolución[3]. Enfocar nuestro estudio aquí es de fundamental importancia, por su relevancia en la actualidad,en especial al sector informático.

La variedad de teléfonos móviles inteligentes disponibles hoy día en el mercado es muy amplia. El número de sistemas operativos no es tan elevado, pero las diferentes versiones de un mismo sistema operativo incrementan la variabilidad con la que se debe lidiar al momento de desarrollar aplicaciones. Cada sistema operativo móvil presenta un Entorno de Desarrollo Integrado (IDE - *Integrated Development Environment*) propio, con un diseño de interfaz diferente; así también, el manejo de los recursos de *hardware* y de los datos en las aplicaciones [8] suele realizarse de manera diferente en cada sistema. Este ambiente fragmentado es uno de los principales retos para los desarrolladores de aplicaciones móviles en la actualidad, donde la variedad de plataformas[3] origina este fenómeno conocido como **fragmentación** [19] [10] [16] [15].

Ante tanta variedad de problemas ocasionados por la fragmentación en diferentes aspectos (*hardware*, interfaz, manejo de datos, entre otros), en el marco de desarrollo de este trabajo, nos centraremos en un aspecto en particular: **la persistencia de datos en los teléfonos móviles**.

Toda aplicación necesita almacenar o retener datos, ya sea de forma permanente o temporal. Las aplicaciones móviles no cuentan con un disco duro ni con una conexión permanente a datos alojados en un servidor, sino que almacenan datos que pueden ser sincronizados luego cuando se cuente con una conexión [5]. Estos datos pueden ser guardados temporal (en caché) o permanentemente en el dispositivo (mediante archivos y bases de datos). Pueden ser de distintos tipos: documentos, respaldo de archivos, media (imágenes, música y vídeo), entre otros. Su origen puede variar, desde una simple configuración de la aplicación que el usuario realiza, a datos precargados en un formulario de la aplicación. De igual forma, los datos pueden provenir de un servidor remoto (por ejemplo, mediante una consulta a un servicio), los cuales pueden ser manipulados localmente [11], guardados en caché, y así estar disponibles aún cuando no se cuente con una conexión a red, pudiendo ser sincronizados luego [4]. Si se desea trabajar con datos que necesitan ser manipulados localmente, o se cuenta con gran cantidad de datos, el mecanismo más utilizado y tradicional, son las bases de datos; pero, existen otros métodos que mencionaremos a continuación.

Como vimos, son varios los escenarios en los que se necesita almacenar datos en las aplicaciones móviles. La fragmentación trae consigo que cada plataforma presente diferentes opciones de persistencia, y a su vez, maneje de forma diferente los distintos mecanismos de persistencia, dificultando esta tarea. Son varias las opciones y mecanismos de almacenamiento, como ser: i) HTML5 (`http://www.w3.org/TR/html5/`) y sus métodos específicos de persistencia dentro de las aplicaciones web; ii) almacenamiento local con archivos y bases de datos integradas como SQLite (`http://www.sqlite.org/`); y, iii) almacenamiento externo con servicios en la nube (como Dropbox y Google Drive), y dispositivos de almacenamiento externo (por ejemplo, tarjetas SD). Hablamos también de mecanismos de almacenamiento temporales, como el almacenamiento tipo pares clave - valor, muy utilizados dentro de la aplicación para guardar datos de configuración, datos de sesión y para pasar datos entre pantallas y aplicaciones [9] [5] [8].

La fragmentación incrementa el tiempo de desarrollo y el costo de mantenimiento de las aplicaciones; tener esto en cuenta al momento de desarrollar una aplicación móvil es muy importante debido a que se debe determinar el alcance que tendrá la aplicación (es decir, si se desea una aplicación para una plataforma específica o para varias) [10]. Para lograr el desarrollo multiplataforma existen varios enfoques: i) el desarrollo web móvil, ahorra esfuerzo de desarrollo, pero con resultados pobres en cuanto rendimiento e interfaz [20] [19]; y, ii) el desarrollo nativo, implica mayor tiempo y esfuerzo de desarrollo, pero arroja mejores resultados en apariencia y aprovechamiento de recursos del dispositivo[20]. Este enfoque es el más afectado por el fenómeno de la fragmentación [10], razón por la cual nos centraremos en los problemas asociados a este enfoque.

Siguiendo el marco investigativo en el que se desarrolla nuestro trabajo, analizaremos la adopción del Desarrollo Dirigido por Modelos (MDD - *Model Driven Development*) como herramienta de solución para minimizar la brecha de heterogeneidad que supone la fragmentación, y de esa forma reducir el esfuerzo de desarrollo de las aplicaciones móviles nativas entre plataformas.

La idea que nos presenta MDD es describir un problema en un modelo y generar software a partir de esta representación [18] [13] [2]. Desarrollar aplicaciones que comparten las mismas funcionalidades y comportamiento, pero para diferentes plataformas, constituye un área adecuado para esta metodología [1] [15]. MDD trata el problema de la redundancia de tareas, reduciendo el esfuerzo de programación y los errores de codificación [6]. Así, el uso de

[2]Dispositivo móvil con avanzados recursos computacionales y equipado con tecnologías que facilitan el acceso a Internet, corren aplicaciones, son táctiles, poseen cámara y otros sensores, todos bajo un avanzado sistema operativo [14] [17].

[3]Una plataforma móvil comprende el recurso de *hardware* subyacente del dispositivo, la arquitectura sobre la que está basada, el sistema operativo, el Kit de Desarrollo de Software (SDK - *Software Developer Kit*) del proveedor y las librerías estándares [17]

metodologías basadas en MDD, frente las problemáticas presentadas previamente, puede suponer facilidades en el desarrollo de aplicaciones para teléfonos inteligentes [15].

Atendiendo a lo dicho anteriormente, la propuesta MoWebA [7] es un enfoque metodológico MDD para el desarrollo de aplicaciones web que adopta el esquema de Arquitectura Dirigido por Modelos (MDA - *Model Driven Architecture*). Esta metodología podría constituirse en una opción adecuada para el entorno móvil, gracias a su estructura por capas bien definida (mediante la cual se logra una clara separación de conceptos) y el modelado centrado en la navegación jerárquica orientada a funciones. En este sentido, esta propuesta proporciona un esquema más adecuado para el diseño de una navegación basada en la interacción del usuario o del contexto [7], al considerar que la manera en la cual la información es organizada y estructurada dentro del sistema no es necesariamente la misma en la que los usuarios acceden a ella. Las aplicaciones móviles son dirigidas por eventos, así que este diseño de navegación sería muy adecuado para el entorno móvil. De la misma forma, MoWebA propone un enriquecimiento de los modelos existentes en orden a considerar aspectos relacionados a la arquitectura final del sistema (por ejemplo RIAs, SOA, REST, entre otros), gracias a su Modelo Específico de la Arquitectura (ASM - *Architecture Specific Modelling*) [7]. Mediante este modelo se podría analizar la posibilidad de representar aplicaciones móviles, utilizando conceptos específicos de la arquitectura y plataforma móvil.

Como objetivo general nos planteamos desarrollar una propuesta MDD que contemple la persistencia en el desarrollo de aplicaciones móviles nativas. Para ello extenderemos MoWebA al ambiente móvil, realizando los ajustes necesarios para dotarla de la capacidad de considerar aspectos y opciones de persistencia durante el desarrollo de aplicaciones móviles.

El resto del presente documento presenta el diseño de la propuesta de extensión de MoWebA para el desarrollo de aplicaciones móviles y, ejemplificamos su uso mediante el modelo de una aplicación móvil.

2. MoWebA Mobile : Una Extensión de MoWebA para el Desarrollo de Aplicaciones Móviles

Existen aplicaciones cuya naturaleza es tratar y manejar datos. En ellos, el almacenamiento de datos a nivel local constituye un aspecto primario. Una aplicación móvil orientada a datos está en constante conexión con datos remotos, y debido a la naturaleza transitoria de los teléfonos móviles inteligentes, requieren de mecanismos para almacenar datos en caso de que no se cuente con esta conexión de red. Este trabajo se enfoca en el desarrollo dirigido por modelos de ese tipo de aplicaciones móviles. Específicamente, siguiendo la arquitectura de software por capas, nos centramos en una capa en particular: en la capa de acceso de datos (o capa de datos) [4] de una aplicación móvil[12].

Partimos de MoWebA, un enfoque MDD para el desarrollo de aplicaciones web, y mediante la extensión de sus metamodelos y perfiles, definimos un ASM para móviles, contemplando las funcionalidades mencionadas.

Las siguientes subsecciones se distribuyen como sigue: en la subsección 2.1 hablamos de los aspectos de la aplicación que consideramos, detallamos qué extensiones realizamos a MoWebA (subsección 2.3) y el proceso de desarrollo que seguimos para la generación de las aplicaciones móviles (subsección 2.2). En la subsección 2.4 presentamos el ASM móvil, donde explicamos los elementos que constituyen y que abarcan las funcionalidades propuestas. Por último, presentamos un ejemplo de modelado de una aplicación móvil utilizando la propuesta (subsección 2.5).

2.1. Aspectos de la aplicación móvil a generar

Con este enfoque nos centramos en la capa de datos. Consideramos la persistencia de datos, a nivel local, y el diseño de los componentes de acceso a datos, para el cual identificamos los distintos proveedores o fuentes de datos en estos dispositivos.

Respecto a la persistencia de datos, identificamos distintos tipos de almacenamiento de datos en los teléfonos móviles inteligentes: base de datos, archivos y pares clave - valor. Considerando este aspecto introducimos los *persistentEntity* que, mediante el *PersistenceType*, nos permiten señalar específicamente qué tipo de almacenamiento estaríamos utilizando para nuestros datos. Las base de datos a generar son para el motor de base de datos SQL, SQLite [5].

Como mencionamos, existen distintas fuentes de datos que proporcionan datos a la aplicación móvil. Identificamos que las aplicaciones móviles pueden recibir datos mediante: comunicación con fuentes de datos externas (por ejemplo, servidores y base de datos externas), comunicación con fuentes de datos internas (sensores, como el giróscopo y el acelerómetro; y recursos de *hardware* específicos del dispositivo, como la cámara y el micrófono) y mediante la comunicación con otras aplicaciones. A fin de representar qué fuentes de datos consideramos manejar con nuestra

[4] En una arquitectura de software por capas, la capa de datos proporciona acceso a los datos alojados dentro de los límites del sistema, y a los datos expuestos por otros sistemas interconectados, quizás por medio de servicios. Se encarga de exponer interfaces de componentes, para que las demás capas puedan consumir [12]

[5] `https://sqlite.org/`

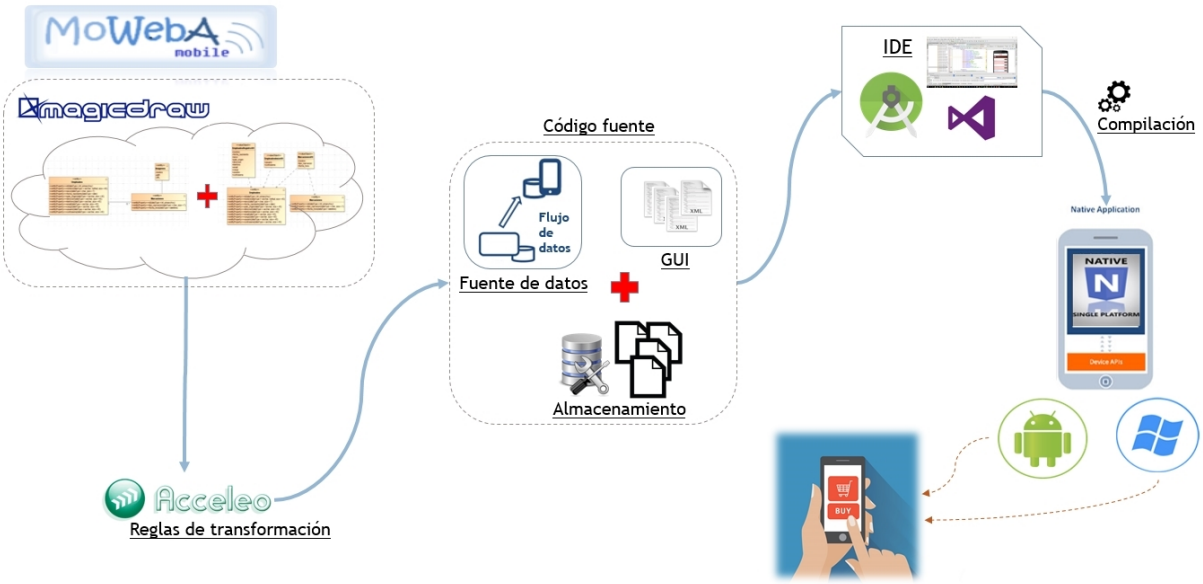

Figura 1: Esquema general de la propuesta

aplicación, introducimos las siguientes interfaces: *WebServiceInterface*, *HardwareDeviceInterface* y *MobileAppDataInterface*, para la representación de las fuentes externas, internas y la comunicación con otras aplicaciones, respectivamente.

La aplicación generada es completa, puede ser instalada en el dispositivo móvil y utilizarse sin mayores inconvenientes, pero constituye una aplicación DEMO con el objetivo de probar las funcionalidades generadas para cada uno de los aspectos que estuvimos viendo. Para testear las principales funcionalidades de cada uno de los métodos de almacenamiento indicados, generamos formularios que nos permiten cargar los datos que definimos para cada entidad, y realizar las distintas operaciones disponibles (operaciones *CRUD*: Crear, Leer, Actualizar y Borrar). Respecto a los proveedores de datos, se tiene una pantalla dedicada a verificar los valores arrojados por cada uno de los sensores y probar los recursos de *hardware* del dispositivo (como el GPS y la cámara) indicados. Además de esta DEMO, proporcionamos *helpers*, funciones para el manejo de cada tipo de almacenamiento bien definidas, disponibles para facilitar las tareas del programador.

2.2. Proceso de desarrollo

En la figura 1 observamos el esquema general del proceso de desarrollo que seguimos para generar las aplicaciones móviles orientadas a datos. Partimos del modelado con la herramienta MagicDraw [6] utilizando el ASM Mobile que proponemos, enfocados en las funcionalidades ya mencionadas. Luego, a modo de generar código a partir de los modelos realizados, los exportamos en formato XMI (XML de Intercambio de Metadatos) para utilizarlo en Acceleo [7]. Aquí definimos reglas de transformación, basados en el modelo importado, lo que nos permite generar el código fuente final de la aplicación móvil. El código fuente generado abarca los aspectos principales de la propuesta: la persistencia local de datos y el diseño de la fuente de datos para la aplicación. Además, a fin de generar una aplicación completa y utilizable, generamos una interfaz que nos permite probar las principales funcionalidades generadas.

A fin de obtener la aplicación nativa, el código generado pasa por un proceso de compilación utilizando la herramienta de diseño de la plataforma objetivo. En nuestro caso generamos tanto para Android como para Windows Phone, por lo que utilizando Android Studio [8] y Visual Studio [9], respectivamente, podremos compilar el código importado y generar la aplicación nativa para cada plataforma.

[6] https://www.nomagic.com/products/magicdraw.html
[7] http://www.eclipse.org/acceleo/
[8] https://developer.android.com/studio/index.html?hl=es-419
[9] https://www.visualstudio.com/es/

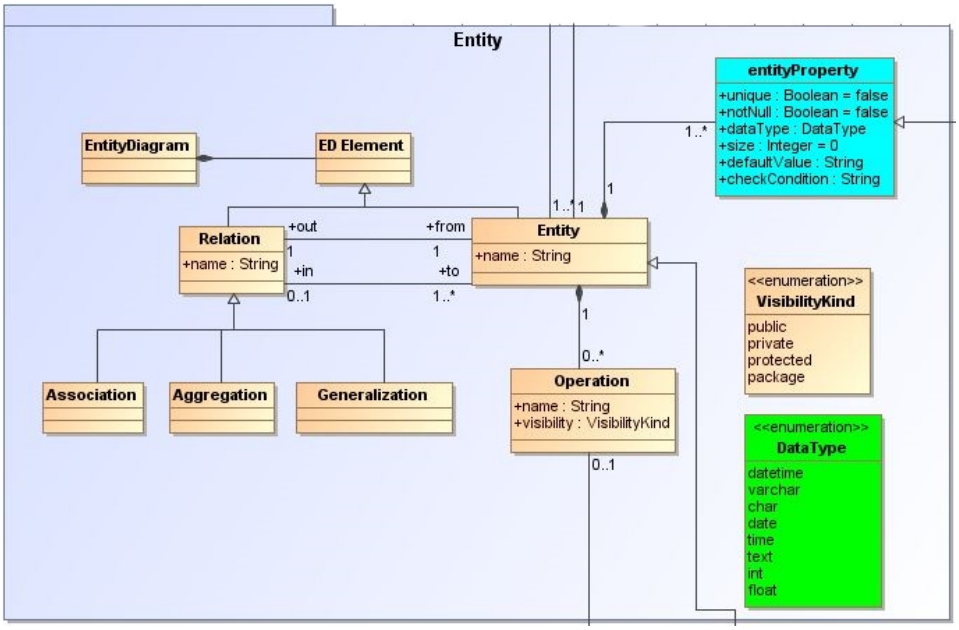

Figura 2: Metamodelo extendido de entidades. *En color salmón, se resaltan las clases que no han sido alteradas de su estado original; en azul, las clases que han sido modificadas; y por último, en verde, las nuevas clases agregadas.*

2.3. Extensiones al PIM de MoWebA: Diagrama de Entidades

Enfocados en el desarrollo de aplicaciones móviles dirigido por modelos, y siguiendo el marco investigativo en el que se desarrolla este trabajo, partimos de MoWebA. MoWebA es un enfoque MDD para el desarrollo de aplicaciones web, que gracias a su estructura por capas bien definida (mediante la cual se logra una clara separación de conceptos) y el modelado centrado en la navegación jerárquica orientada a funciones, consideramos adecuado para el ambiente móvil.

A fin de definir la estructura y las relaciones estáticas entre las clases identificadas en el dominio del problema [7], MoWebA cuenta con el diagrama de entidades. Extendimos este diagrama a fin de proporcionar los elementos necesarios para modelar el diagrama conceptual de una base de datos.

La extensión propuesta se puede observar en la figura 2 (metamodelo de entidades extendido) y en la figura 3 (perfil de entidades extendido). Con la intención de facilitar la comprensión de los cambios realizados, las clases en el metamodelo y en el perfil de entidades extendidos han sido coloreados, utilizando la siguiente convención de colores: en color salmón se encuentran las clases que no han sido alteradas de su forma original; en azul, las que han sido modificadas ya sea mediante la agregación, actualización o eliminación de alguna de sus propiedades; y por último, en verde, aquellas clases nuevas adheridas.

Para detallar los aspectos propuestos y extendidos, utilizamos el metamodelo extendido de entidades. Como vemos en la figura 2, se realizaron dos tipos de modificaciones: una extensión de las propiedades de una entidad (*EntityProperty*), y se agregaron los tipos de datos específicos que pueden tener estas propiedades (*DataType*). La intención es mapear cada entidad (*Entity*) con una tabla de la base de datos, por lo que podemos definir los campos de la tabla y especificar: el tipo de dato (*DataType*), cuyos valores pueden ser *datetime*, *varchar*, *char*, *date*, *time*, *text*, *int* y *float*; un tamaño máximo (*size*), aplicado a los tipos *char*, *varchar* e *int*; restringir que el campo sea nulo (*notNull*); indicar que la tabla tenga un valor único (*unique*), o un valor por defecto (*defaultValue*) y condiciones que deben cumplirse (*checkCondition*).

2.4. ASM Mobile

Luego de presentar las extensiones que realizamos al PIM de MoWebA, continuamos a explicar las extensiones realizadas para definir las aplicaciones móviles con las funcionalidades definidas: la persistencia local de datos y el diseño de los componentes para el acceso a fuente de datos. En la figura 4 observamos el metamodelo de nuestra propuesta para el desarrollo de aplicaciones móviles nativas enfocadas en la capa de datos a partir de MoWebA. La figura 5 muestra el correspondiente perfil de la propuesta. En ambas figuras podemos resaltar los siguiente: se resaltan dos aspectos en particular, la persistencia de datos (dentro del cuadro con líneas punteadas: *PersistentData*) y los proveedores de datos (dentro del cuadro con líneas punteadas: *Data provider*). Dentro de los proveedores de datos, encontramos tres secciones distinguibles: en rojo, distinguimos que otras aplicaciones funcionan como proveedores

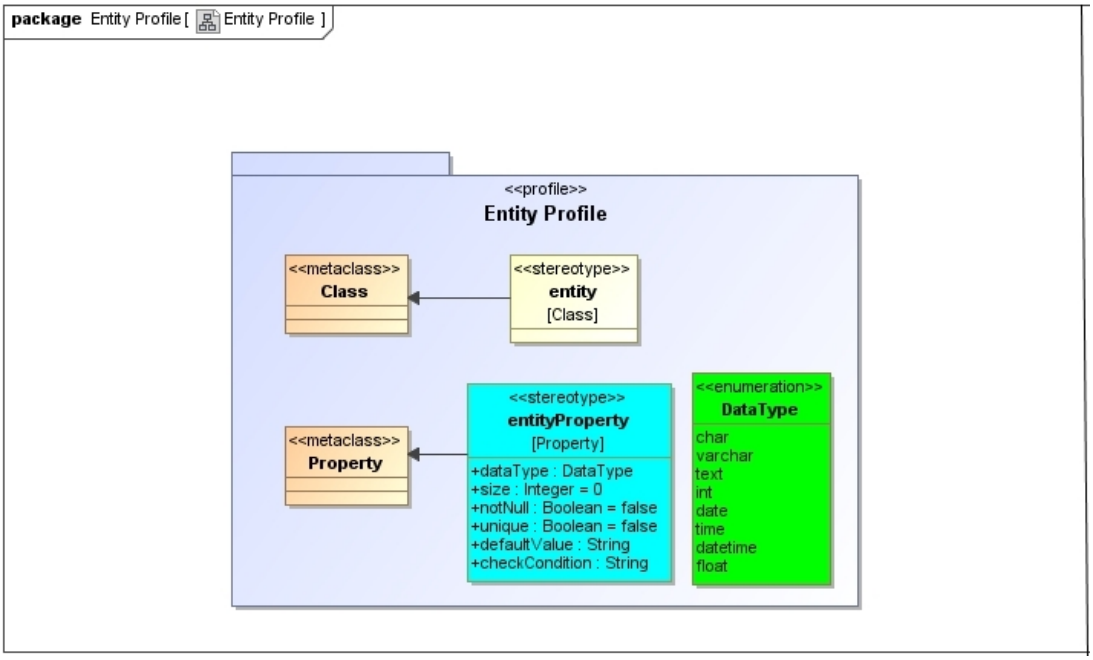

Figura 3: Perfil extendido de entidades. *En color salmón, se resaltan las clases que no han sido alteradas de su estado original; en azul, las clases que han sido modificadas; y por último, en verde, las nuevas clases agregadas.*

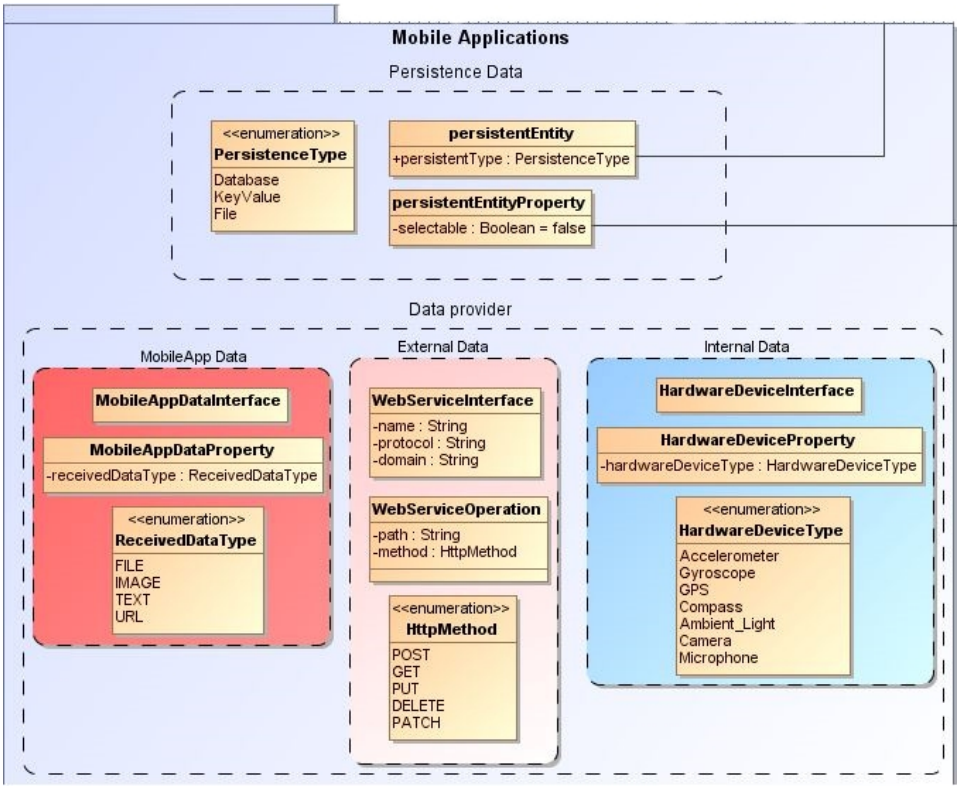

Figura 4: Metamodelo MoWebA Mobile. Distinguimos dos secciones en particular: la persistencia de datos (*Persistent Data*) y los proveedores de datos (*Data Provider*). Dentro de los proveedores de datos, distinguimos tres tipos de proveedores: en rojo, otras aplicaciones como fuente de datos (*MobileAppData*); en rosado, las fuentes externas de datos (*External Data*); y por último, en azul claro, las fuentes internas de datos (*Internal Data*).

de datos (dentro del cuadro con líneas punteadas: *MobileAppData*); en rosado, identificamos las fuentes externas de datos (dentro del cuadro con líneas punteadas: *External Data*); y por último, en azul claro, las fuentes internas de

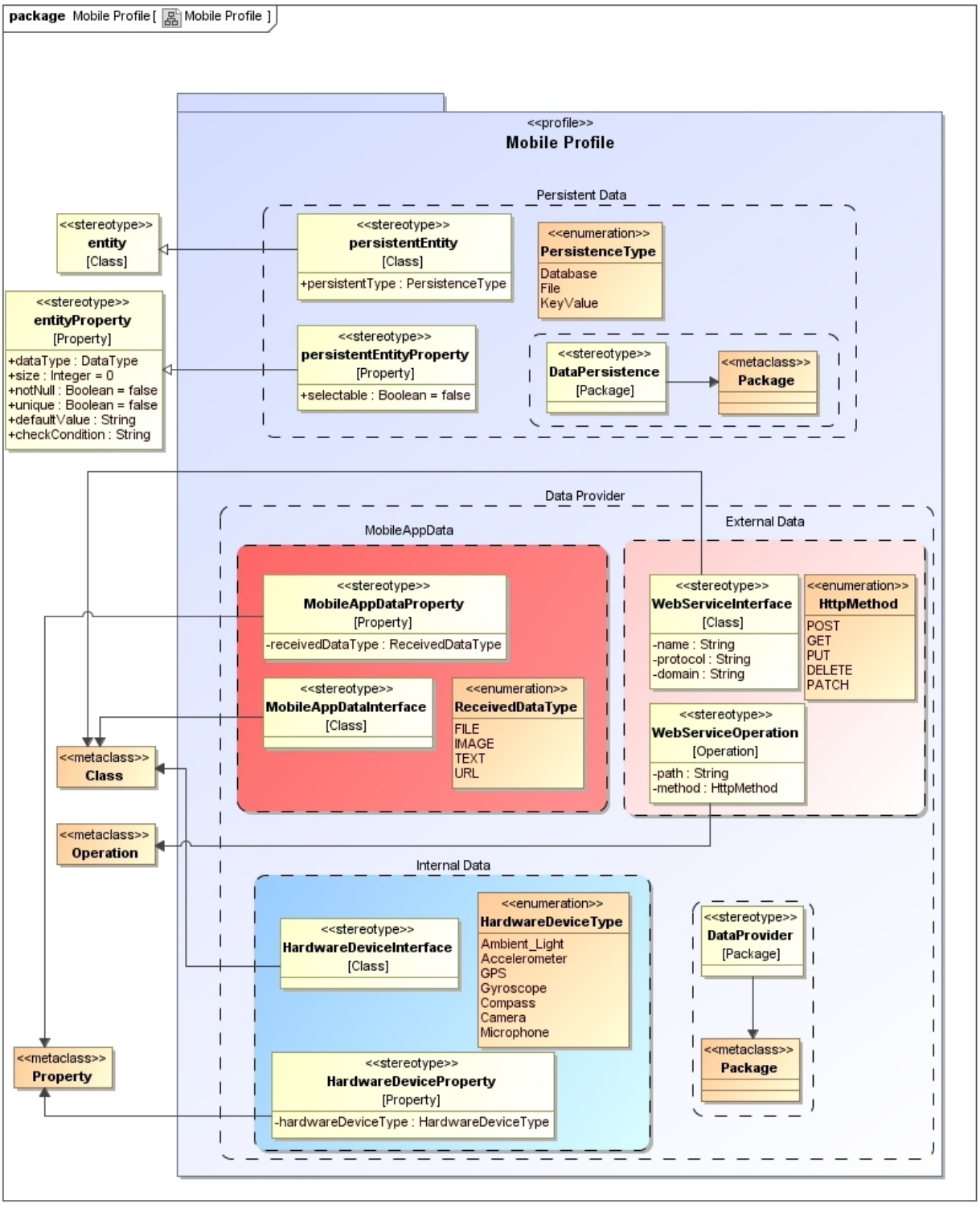

Figura 5: Perfil MoWebA Mobile. Distinguimos dos secciones en particular: la persistencia de datos (*Persistent Data*) y los proveedores de datos (*Data Provider*). Dentro de los proveedores de datos, distinguimos tres tipos de proveedores: en rojo, otras aplicaciones como fuente de datos (*MobileAppData*); en rosado, las fuentes externas de datos (*External Data*); y por último, en azul claro, las fuentes internas de datos (*Internal Data*).

datos (dentro del cuadro con líneas punteadas: *Internal Data*). A continuación pasamos a desarrollar cada uno de los aspectos mencionados.

2.4.1. Persistencia de datos

Para especificar el tipo de almacenamiento que deseamos utilizar en nuestra aplicación y así poder manejar nuestros datos, creamos dos clases: una entidad persistente (*PersistentEntity*) que constituye la extensión de su equivalente (*Entity*) del diagrama de entidades; y agregamos propiedades a estas entidades persistentes (*persistentEntityProperty*).

El *persistentEntity* permite generar modelos de datos, y gracias al valor etiquetado asociado, *persistentType*, podemos indicar el tipo de almacenamiento que deseamos utilizar, y pueden ser: *Database*, el cual nos permite generar una base de datos y utilizar el nombre de la entidad persistente como una tabla; *File*, nos permite guardar la entidad persistente en archivos, mediante funciones que facilitan el manejo de archivos (por ejemplo, leer, escribir y otros); y, por último, *KeyValue*, el cual nos permite almacenar la entidad persistente en pares clave - valor, para la cual proporcionamos funciones para el manejo de estos datos.

Respecto a *persistentEntityProperty*, agregamos la propiedad *selectable*. La misma nos permite indicar si un campo o atributo de la entidad persistente será usado o no, como valor clave de selección de datos para realizar las operaciones de borrado y actualización en las base de datos.

2.4.2. Proveedores de datos

Identificamos que una aplicación puede recibir datos de tres tipos de fuentes: interna, externa y mediante otras aplicaciones instaladas en el mismo dispositivo. A fin de representar este comportamiento, agregamos nuevos elementos, los cuales detallaremos a continuación.

Las fuentes externas proveen datos mediante la comunicación de la aplicación móvil con servidores o base de datos externas. Nos enfocamos en la comunicación mediante servicios web, específicamente servicios *Rest*. La clase *WebServiceInterface*, mediante valores etiquetados, permite definir la URL base de conexión: protocolo y dominio ([*protocol*]:[*domain*]). Con *WebServiceOperation* definimos las funciones o servicios en nuestra clase *WebServiceInterface*. El nombre de la función corresponde con el nombre del servicio, al cual, mediante valores etiquetados, le agregamos el método http que utiliza para la conexión (*method*) con posibles valores: *POST*, *GET*, *PUT*, *DELETE* y *PATCH*; y la ruta de acceso al servicio (*path*).

Los teléfonos móviles inteligentes cuentan con sensores y con recursos de *hardware* específicos para ciertas funciones como cámara, micrófono, entre otros, que proveen flujos de datos; consideramos esto como la fuente interna de datos. La clase *HardwareDeviceInterface* nos permite definir qué sensores y/o recursos de *hardware* propios del dispositivo utilizar mediante *HardwareDeviceProperty* y su valor etiquetado *hardwareDeviceType*. Identificamos una serie de opciones de sensores y recursos de *hardware*, comunes en los teléfonos móviles inteligentes: acelerómetro (*Accelerometer*), giróscopo (*Gyroscope*), *GPS*, brújula (*Compass*), sensor de luz (*AmbientLight*), cámara (*Camera*) y micrófono (*Microphone*).

Por último, la aplicación móvil puede comunicarse con otras aplicaciones instaladas en el mismo dispositivo móvil, puede tanto enviar o recibir datos. Nos interesan los datos que la aplicación puede recibir, para lo cual creamos la clase *MobileAppDataInterface*. Con el valor etiquetado *receivedDataType* de la propiedad (*MobileAppDataProperty*), seleccionamos qué tipo de dato será capaz de recibir y manejar la aplicación. Identificamos distintos tipos de datos comúnmente utilizados en el intercambio de datos entre aplicaciones, pueden ser (*ReceivedDataType*): *File*, para intercambio de archivos en general; *Image*, para recibir imágenes; *Text*, para recibir texto; y *Url*, para recibir enlaces (*links*) en general.

2.5. Modelo de una aplicación móvil

Presentamos un ejemplo de modelado de una aplicación móvil llamado ***e-market***, partiendo del ASM Mobile y enfocados en los aspectos mencionados.

La aplicación consiste en una tienda virtual que hace entregas a domicilio. Uno tiene disponible un catálogo de productos y toda la información necesaria de cada producto. Si se desea adquirir un producto en particular, se selecciona dicho producto y se agrega al carrito de compras. Una vez finalizado la selección de los productos, se finaliza la compra, con la posibilidad de indicar si se desea que la entrega se haga a domicilio o no. El usuario necesita estar registrado para ingresar a la aplicación. La aplicación debe estar disponible en todo momento, aún sin contar con una conexión a internet permanente (en modo *offline*).

En la figura 6 vemos el diseño de la estructura de datos de nuestra aplicación. Uno de los requisitos fue que la aplicación esté disponible aún sin conexión de red, por lo que utilizamos las entidades persistentes («*persistentEntity*») para indicar dónde almacenaremos los datos. Tenemos cinco entidades, cada una con propiedades estereotipadas con «*persistentEntityProperty*»: un producto (*Product*) tiene uno o mas proveedores (*Provider*), y a la vez puede tener cero o más imágenes asociadas. Tenemos un carrito (*ShoppingCart*) donde iremos almacenando los productos que se elijan, el cual puede estar compuesto de cero o más productos. Por último, tenemos a un usuario (*User*) donde

almacenamos los datos de registro a la aplicación. Como observamos, cada entidad tiene especificada un tipo de persistencia (*persistentType*): tanto los productos, el carrito y los proveedores serán almacenados en una base de datos local (*persistentType = Database*), por lo que éstos tienen definido un atributo con el valor etiquetado *selectable*. Las imágenes de cada producto serán almacenados en forma de archivos en una carpeta local del dispositivo (*persistentType = File*). La sesión del usuario, y sus datos, los almacenaremos en pares clave - valor (*persistentType = KeyValue*).

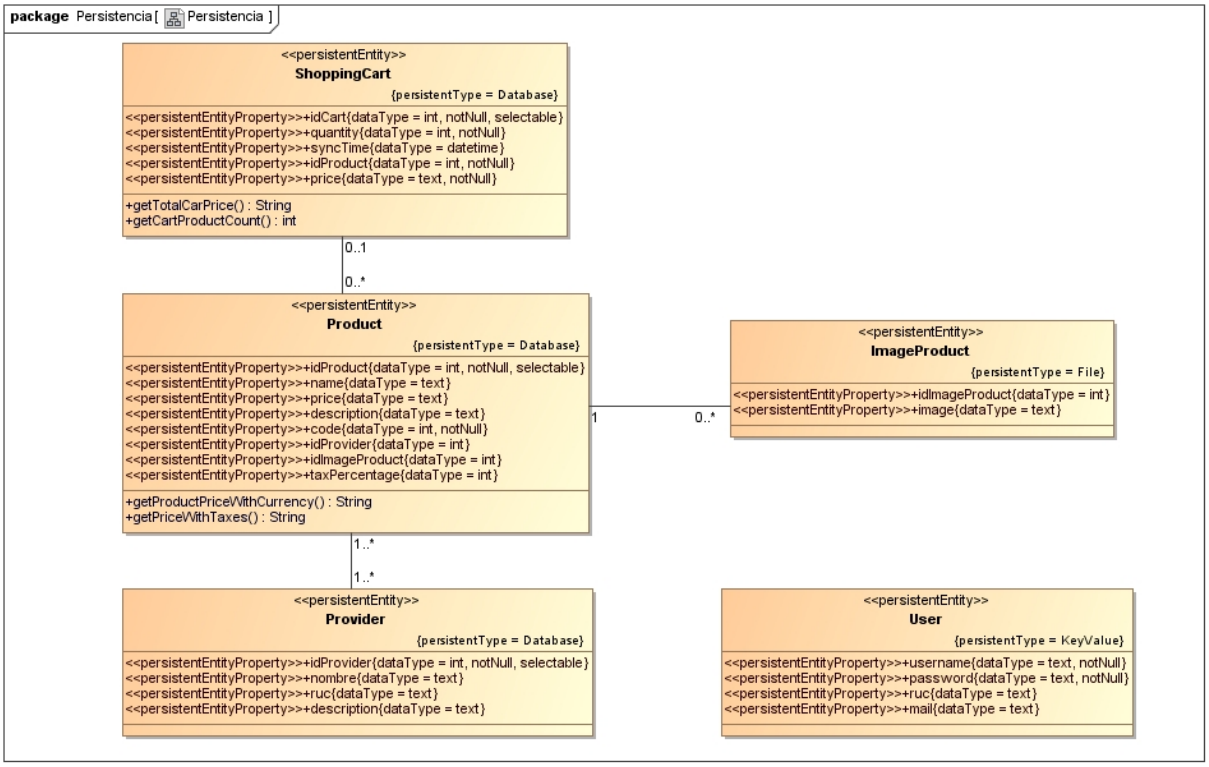

Figura 6: Modelo de la persistencia de datos de la aplicación ***e-market***

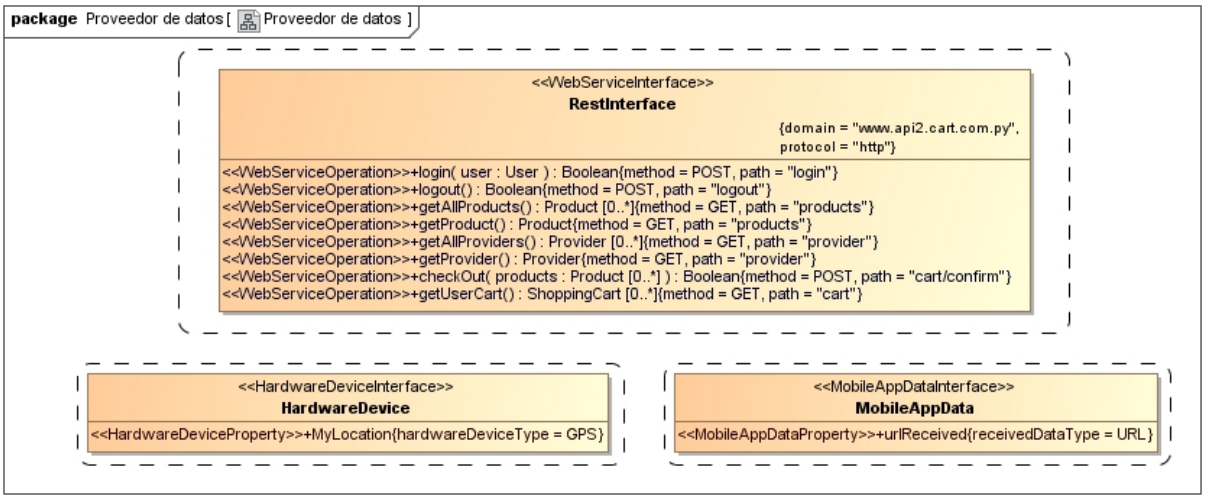

Figura 7: Modelo de los proveedores de datos de la aplicación ***e-market***

Para representar de dónde obtendremos estos datos, modelamos los proveedores de datos, como vemos en la figura 7. Tenemos la clase *RestInterface* con estereotipo «*WebServiceInterface*» donde definimos las interfaces de los servicios que estaríamos utilizando. Con valores etiquetados definimos la URL base: *domain* y *protocol*. Para representar los servicios, se usan las operaciones con el estereotipo «*WebServiceOperation*», donde usamos el nombre del mismo como nombre del servicio, y con valores etiquetados definimos a qué método http corresponde (*method*) y la ruta al servicio

(*path*). Por otra parte, creamos la clase *HardwareDevice* con estereotipo «*HardwareDeviceInterface*». Aquí definimos el atributo *MyLocation*, con estereotipo «HardwareDeviceProperty», e indicamos que estaríamos recibiendo los datos de localización del dispositivo mediante GPS (*hardwareDeviceType = GPS*) para obtener nuestra dirección en caso de la entrega a domicilio. Por último, la clase *MobileAppData* con estereotipo «*MobileDataInterface*», nos permite definir que la aplicación podrá ser capaz de recibir datos compartidos del tipo URL (*receivedType = URL*). Mediante esto, cualquier otra aplicación, podrá interactuar con esta aplicación y realizar determinadas acciones en respuesta a eso (por ejemplo, como usuario uno puede recibir una promoción de un determinado producto en el correo; al dar click al enlace, se abriría la aplicación y nos mostraría el producto listo para agregarlo a nuestro carrito). Esta es una funcionalidad extra que nos pareció interesante mostrar en el modelado.

3. Referencias

[1] F. T. Balagtas-Fernandez and H. Hussmann. Model-driven development of mobile applications. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE '08, pages 509–512, Washington, DC, USA, 2008. IEEE Computer Society.

[2] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[3] eMarketer. Global media intelligence report executive summary. `https://www.emarketer.com/public_media/docs/GMI-2015-ExecutiveSummary.pdf`, 2015.

[4] Joao André Lopes Ferreira and Alberto Rodrigues da Silva. Mobile cloud computing. *Open Journal of Mobile Computing and Cloud Computing*, 1(2), November 2014.

[5] Marin Fotache, Dragos Cogean, et al. Nosql and sql databases for mobile applications. case study: Mongodb versus postgresql. *Informatica Economica*, 17(2):41–58, 2013.

[6] S. Geiger-Prat, B. MarThín, S. España, and G. Giachetti. A gui modeling language for mobile applications. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 76–87, May 2015.

[7] Magalí González, Luca Cernuzzi, and Oscar Pastor. A Navigational Role-Centric Model Oriented Web Aproach - MoWebA. *International Journal of Web Engineering and Technology (IJWET)*, 11(1):29–67, 2016.

[8] Tor-Morten Grønli, Jonas Hansen, Gheorghita Ghinea, and Muhammad Younas. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 635–641. IEEE, 2014.

[9] Qusay H. Mahmoud, Shaun Zanin, and Thanh Ngo. Integrating mobile storage into database systems courses. In *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, pages 165–170, New York, NY, USA, 2012. ACM.

[10] Euler Horta Marinho and Rodolfo Ferreira Resende. Native and multiple targeted mobile applications. In *Computational Science and Its Applications–ICCSA 2015*, pages 544–558. Springer, 2015.

[11] C.M Melman. A generative approach for data synchronization between web and mobile applications. Master thesis, Electrical Engineering, Mathematics and Computer Science, 2014.

[12] Microsoft Patterns. *Microsoft Application Architecture Guide*. Microsoft Press, 2nd edition, 2009.

[13] Claudia Pons, Roxana Giandini, and Gabriela Perez. *Desarrollo de software dirigido por modelos : conceptos teoricos y su aplicacion practica*. Mc Graw Hill Educacion, 1era edition, 2010.

[14] Yonathan Aklilu Redda. Cross platform mobile applications development. *Norwegian University of Science and Technology, Norwegian University of Science and Technology, Norwegian University of Science and Technology, Master in Information Systems*, 2012.

[15] Alejandro Ribeiro and Airton R da Silva. Survey on cross-platforms and languages for mobile apps. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, pages 255–260. IEEE, 2012.

[16] André Ribeiro and Alberto Rodrigues da Silva. Xis-mobile: A dsl for mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1316–1323, New York, NY, USA, 2014. ACM.

[17] Andreas Sommer and Stephan Krusche. Evaluation of cross-platform frameworks for mobile applications. In *Software Engineering (Workshops)*, pages 363–376, 2013.

[18] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2006.

[19] Robert Ramírez Vique. Métodos para el desarrollo de aplicaciones móviles. *PID_00176755*, 2012.

[20] Spyros Xanthopoulos and Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA, 2013. ACM.