



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Herramienta para generación automática de aplicaciones web móviles

Danier Fresley Vanegas Becerra

Universidad Nacional de Colombia
Facultad de ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2020

Herramienta para generación automática de aplicaciones web móviles

Danier Fresley Vanegas Becerra

Trabajo de investigación presentado como requisito parcial para optar al título de:
Magíster en Ingeniería - Ingeniería de Sistemas y Computación

Director:

Msc. Henry Roberto Umaña Acosta

Línea de Investigación:

Ingeniería de Software

Grupo de Investigación:

Colectivo de Investigación en Ingeniería de Software (CoISWe)

Universidad Nacional de Colombia

Facultad de ingeniería, Departamento de Ingeniería de Sistemas e Industrial

Bogotá, Colombia

2020

A Dios, a mis hijos María José Vanegas Mercado y Danier Davit Vanegas Mercado, a mi esposa, Vita María Mercado Rodríguez, a mis madres Matilde del Socorro Becerra y Nilfa Leonor Vanegas Becerra más que el motor de la vida fueron parte importante de lo que hoy puedo presentar como trabajo de tesis, gracias a ellos por cada palabra de apoyo, gracias por cada momento en familia sacrificado para ser invertido en el desarrollo de la tesis; gracias por entender que el éxito demanda algunos sacrificios.

Agradecimientos

Mi más sincero agradecimiento al Msc. Henry Roberto Umaña Acosta, principal colaborador durante todo este proceso, quien, con su dirección, conocimientos, enseñanza y colaboración permitió el desarrollo de este trabajo.

También agradezco, a mis amigos que la vida me regalo, Wilman José Vega Castilla y Deivs de Jesús Martínez Acosta a quienes les debo su comprensión y apoyo incondicional durante este proceso.

Resumen

Actualmente, el acceso a sitios, páginas y aplicaciones web desde dispositivos móviles toma suma importancia debido a la necesidad de movilidad de los usuarios y de la variedad en capacidad y tecnología de los dispositivos.

La Web Móvil cuya principal cualidad es precisamente la movilidad, supone un reto para los desarrolladores de aplicaciones web móviles, que asumen el desafío de construir aplicaciones fundamentadas en esta cualidad, pero que también deben tener en cuenta la portabilidad entre sistemas y tecnologías, y la usabilidad, para facilitar acceso a la web desde los dispositivos móviles.

Este trabajo utiliza el desarrollo de software dirigido por modelos, presentando una metodología basada en dicho desarrollo, para construir una herramienta que permita la generación automática de aplicaciones web dirigidas a dispositivos móviles, Teniendo como prototipo referente, una aplicación que identifica las variaciones en los precios de la canasta familiar en supermercados y tiendas de barrios colombianos.

Esta implementación de referencia cubre los aspectos básicos de la mayoría de aplicaciones webs móviles más utilizadas: utilización de una arquitectura de componentes, acceso a múltiples bases de datos y utilización de un esquema de servicios para realizar operaciones.

Con esta base, la herramienta propuesta mejorará el tiempo de desarrollo de los programadores en la construcción de software para la Web móvil.

Palabras clave: Web Móvil, Desarrollo Dirigido por Modelos, Arquitectura Dirigida por Modelos, Lenguaje de Dominio Especifico, Transformación de modelos.

Abstract

Accessing to sites, pages and web applications from mobile devices right now is very important due to the need for user mobility and the variety in capacity and technology of the devices.

The mobile web, whose main quality is precisely mobility, is a challenge for developers of mobile web applications, who assume the challenge to build applications based on this quality, but which must also take into account interoperability between systems, technologies, and usability, to facilitate access to the web from mobile devices.

This work uses the model driven development, presenting a methodology based on that development, to build a tool that allows the automatic generation of web applications for mobile devices, having as a reference prototype, an application that identifies price variations of the family basket in supermarkets and stores in Colombian neighborhoods.

This reference implementation covers the basic aspects of most commonly used mobile web applications: using a component architecture, access to multiple databases and use of a service scheme to perform operations.

With this base, the proposed tool will improve the development time of programmers in the construction of software for the mobile Web.

Keywords: Mobile Web, Model-Driven Development, Model-Driven Architecture, Domain-Specific Language, Model transformation.

Contenido

	Pág.
Resumen	IX
Lista de figuras.....	XIII
Lista de cuadros.....	XIV
1. Introducción	17
1.1 Objetivo General	18
1.2 Objetivos específicos	18
1.3 Esquema del documento.....	18
2. Conceptos relacionados.....	20
2.1 Mejores prácticas para el desarrollo de aplicaciones Web Móviles	20
2.1.1 Mantener Economía en el uso de red	20
2.1.2 Dar libertad a los usuarios	20
2.1.3 Recuerde los principios del diseño Web	21
2.1.4 Diseñar aplicaciones flexibles	21
2.1.5 Aprovechar las características de los móviles.....	21
2.1.6 Optimizar el tiempo de respuesta	21
2.2 Desarrollo de software dirigido por modelos.....	21
2.3 Arquitectura dirigida por modelos (ADM).....	22
2.4 Lenguaje de dominio específico	22
2.5 Transformación de modelos	23
2.6 Generación de código	24
2.7 Planteamientos significativos sobre desarrollo de software dirigido por modelos (DSDM) para aplicaciones móviles	25
3. Elementos del diseño de la herramienta propuesta	27
3.1 Dominio del problema	27
3.2 Diagrama de contexto	27
3.3 Descripción de alcance	28
3.4 Requerimientos funcionales del problema de dominio específico.....	28
3.5 Lenguajes de programación, herramientas y marcos de trabajo que apoyarán el desarrollo de la metodología.....	29
3.5.1 Lenguajes de programación.....	29
3.5.2 Marcos de trabajo.....	30
4. Construcción de la herramienta para generación automática de aplicaciones web móvil dirigido por modelos.....	31
4.1 Metodología de trabajo.....	31

4.2	Implementación de referencia	33
4.3	Análisis de componentes	34
4.3.1	Componentes genéricos.....	34
4.3.2	Componentes frecuentes	38
4.3.3	Componentes individuales	39
4.4	Metamodelo	40
4.4.1	Metamodelo del dominio	41
4.4.2	Metamodelo de la arquitectura	42
4.4.3	Metamodelo de tecnología	43
4.5	Diseño del DSL.....	44
4.5.1	Dominio.....	45
4.5.2	Arquitectura.....	46
4.5.3	Tecnología	48
4.6	Gramática del lenguaje específico de dominio (DSL) en Xtext	49
4.7	Desarrollo de transformaciones	50
4.7.1	Definición de las transformaciones	50
4.8	Generación del Editor	74
4.9	Plantillas de generación de código automática	76
5.	Evaluación.....	77
5.1	Análisis conceptual	77
5.2	Prueba de concepto.....	77
5.2.1	Lenguaje específico de dominio	78
5.2.2	Transformaciones.....	78
5.2.3	Generación de código	78
5.3	Análisis práctico.....	78
6.	Conclusiones y Trabajo futuro	81
A.	Anexo: Gramática del lenguaje Xtext	83
B.	Anexo: Código fuente del lenguaje específico de dominio generado.....	89
	Bibliografía	94

Lista de figuras

	Pág.
Figura 3-1. Diagrama de contexto. (Elaboración propia).....	28
Figura 3-2. Diagrama de descomposición. (Elaboración propia).....	29
Figura 4-1. Metodología para el desarrollo de la herramienta, iniciando en la parte superior con el ítem 1, siguiendo a la derecha con el ítem 2, realizando luego el ítem 3 y finaliza en ítem 4, el cual indica que se debe iterar todo el proceso. (Elaboración propia)	32
Figura 4-2. Metodología de trabajo, inicia en la parte superior con el ítem 1, continuando a la derecha, finalizando con el ítem 6. (Elaboración propia).....	33
Figura 4-3. Arquitectura de la implementación de referencia. (Elaboración propia)	34
Figura 4-4. Proyecto AppPrecio en Ionic. (Elaboración propia).....	35
Figura 4-5. Carpeta principal donde se almacena. (Elaboración propia).....	36
Figura 4-6. Archivos dentro de la carpeta app. (Elaboración propia)	37
Figura 4-7. Elementos principales del metamodelo. (Elaboración propia).....	41
Figura 4-8. Metamodelo del dominio. (Elaboración propia).....	42
Figura 4-9. Metamodelo de la arquitectura. (Elaboración propia)	43
Figura 4-10. Metamodelo de la tecnología. (Elaboración propia).....	44
Figura 4-11. Árbol de sintaxis concreta general. (Elaboración propia)	44
Figura 4-12. Árbol de sintaxis concreta del dominio. (Elaboración propia).....	45
Figura 4-13. Árbol de sintaxis concreta de la arquitectura. (Elaboración propia).....	46
Figura 4-14. Árbol de sintaxis concreta de la Arquitectura – <i>CampoEntrada</i> . (Elaboración propia).....	47
Figura 4-15. Árbol de sintaxis concreta de la Arquitectura – Botón. (Elaboración propia).....	47
Figura 4-16. Árbol de sintaxis concreta de la Arquitectura – <i>BotonOpcionItem</i> . (Elaboración propia)	48
Figura 4-17. Árbol de sintaxis concreta de la Arquitectura – Arreglo. (Elaboración propia)	48
Figura 4-18. Árbol de sintaxis concreta de la Arquitectura – Listado. (Elaboración propia)	48
Figura 4-19. Árbol de sintaxis concreta de la tecnología. (Elaboración propia).....	49
Figura 4-20. Pruebas del nuevo DSL.....	75
Figura 4-21. Asistente para ver el metamodelo gráfico.....	75
Figura 4-22. Prueba del metamodelo grafico a partir de la creación con texto.....	76

Lista de cuadros

	Pág.
Cuadro 2-1. Propuestas en transformación de modelos (Elaboración propia basado en [19] [20] [21] [22] [24] [25] [27] [28] [29] [30] [31]).....	23
Cuadro 2-2. Enfoques en las definiciones de reglas funcionales (Elaboración propia basado en [19] [20] [21] [22] [24] [25] [27] [28] [29] [30] [31])	24
Cuadro 2-3. Lenguajes de transformación de modelo a texto. (Elaboración propia).....	24
Cuadro 2-4. Trabajos que aplican un enfoque basado en modelos. (Elaboración propia)	25
Cuadro 4-1. Componentes genéricos del proyecto. (Elaboración propia).....	36
Cuadro 4-2. Componentes genéricos de la carpeta src. (Elaboración propia).....	37
Cuadro 4-3. Componentes genéricos de la carpeta app. (Elaboración propia).....	38
Cuadro 4-4. Componentes frecuentes. (Elaboración propia).....	39
Cuadro 4-5. Componentes individuales. (Elaboración propia).....	40
Cuadro 4-6. Descripción de las transformaciones. (Elaboración propia)	50
Cuadro 4-7. Definición de las transformaciones – del “Modulo”.	51
Cuadro 4-8. Definición de las transformaciones – de los artefactos “CampoEntrada”, “titulo”, “CamposEntradas”, “compileBotones”, “Lista”.....	54
Cuadro 4-9. Definición de las transformaciones – de los artefactos “CampoEntrada”, “Boton”, “BotonOpcionItem”.	57
Cuadro 4-10. Definición de las transformaciones – de los artefactos “Componente”, “arreglos”, “interfaz”, “listado”, “lista”, “servicio”, “métodos”.	59
Cuadro 4-11. Definición de las transformaciones – del artefacto “addItem”.....	61
Cuadro 4-12. Definición de las transformaciones – del artefacto “loadItems”.....	62
Cuadro 4-13. Definición de las transformaciones – del artefacto “updateItem”.....	64
Cuadro 4-14. Definición de las transformaciones – del artefacto “deleteItem” del componente.	65
Cuadro 4-15. Definición de las transformaciones – del artefacto “servicio”.	68
Cuadro 4-16. . Definición de las transformaciones – del artefacto “add” del servicio.....	69
Cuadro 4-17. Definición de las transformaciones – del artefacto “getAll” del servicio	70
Cuadro 4-18. Definición de las transformaciones – del artefacto “update” del servicio	72
Cuadro 4-19. Definición de las transformaciones – del artefacto “delete” del servicio	74
Cuadro 5-1. Identificación de líneas de código fuente frente a las generadas de manera automática.	79

1.Introducción

Las aplicaciones móviles, son programas informáticos para ser instalados y ejecutados en dispositivos inteligentes, por lo general se pueden descargar en plataformas destinadas para su distribución, como la Google Play, del sistema operativo Android; App Store, del sistema operativo iOS, entre otros.

La pluralidad de plataformas a las que van dirigidas estas aplicaciones, requieren un esfuerzo adicional en capacidades tecnológicas y procedimentales, para que estas aplicaciones puedan, con cierto grado de calidad, reflejarse de manera transparente en cada una de las plataformas para las que se destinan.

Una forma de reducir estos esfuerzos es a través de la estrategia de la Aplicación Web Móvil, que enfoca el desarrollo de aplicaciones con soporte en múltiples plataformas, pero centrado en el objetivo de ejecutar dichas aplicaciones en los navegadores actuales para esas plataformas, es decir, la normalización en el navegador web.

Para desarrollar este tipo de aplicaciones, se utilizan marcos de trabajo como *Angular*, desarrollado en el lenguaje *TypeScript*, y paquetes de desarrollo de software como *Ionic*, basados en Angular, que agiliza el desarrollo y despliegue de aplicaciones web móviles, optimiza el rendimiento y permite la reutilización de componentes en el desarrollo.

El objetivo del presente trabajo es construir una herramienta para generación automática de aplicaciones web móviles por medio de un Lenguaje específico para integrar los marcos de trabajo *Ionic* y *Angular*, utilizando una metodología de desarrollo dirigido por modelos, para ofrecer mayor flexibilidad al desarrollador, descartando detalles irrelevantes en la implementación y orientándose al desarrollo con las mejores prácticas en la construcción de aplicaciones Web Móvil.

1.1 Objetivo General

El objetivo principal de este trabajo es construir una herramienta para generación automática de aplicaciones web móviles.

1.2 Objetivos específicos

Los objetivos específicos del presente trabajo son los siguientes:

- Implementar las mejores prácticas para las aplicaciones Web Móviles.
- Desarrollar los componentes para la herramienta de desarrollo de aplicaciones Web Móviles a través de MDD (Desarrollo Dirigido por Modelos) con Xtext (marco de trabajo de código abierto para el desarrollo de lenguajes de programación) y un Lenguaje específico de dominio (DSL, por sus siglas en inglés).
- Demostrar la herramienta con una prueba de concepto.

1.3 Esquema del documento

Este documento está conformado por tres fases que se describen a continuación.

En la fase 1 (capítulo 2) se establecen las pautas para la implementación de las mejores prácticas en el desarrollo de las aplicaciones móviles, según las recomendaciones de la W3C (World Wide Web Consortium). De igual manera se determina en esta fase la relación de conceptos que envuelven el Desarrollo dirigido por modelos, enfocado en el Desarrollo Web móvil.

En la fase 2 (capítulo 3 y capítulo 4), se establecen los elementos del diseño de la herramienta propuesta, y se define la metodología para el desarrollo de la aplicación Web Móvil a través del desarrollo dirigido por modelos.

El capítulo 3 se enfoca en los elementos de diseño para la construcción de la herramienta propuesta, que comprende la selección del problema de dominio específico y la elección de las herramientas y marcos de trabajo para el apoyo de la construcción.

El capítulo 4 introduce los elementos de la metodología dirigida por modelos, y el desarrollo de las siguientes actividades para la implementación de la metodología:

- Realización de la implementación de referencia mediante el uso de las tecnologías seleccionadas.
- Identificación y clasificación del código generado en el lenguaje de programación seleccionado.
- Creación del lenguaje de metamodelo.
- Generación de artefactos del lenguaje de dominio específico.
- Prueba y validación del lenguaje de dominio.

En la fase 3 (capítulo 5) se realiza una demostración de la metodología propuesta mediante una prueba de concepto funcional.

2. Conceptos relacionados

2.1 Mejores prácticas para el desarrollo de aplicaciones Web Móviles

Una aplicación Web Móvil es un programa desarrollado con tecnología Web para dispositivos móviles que cuenten con una conexión a Internet y un navegador. El consorcio internacional WWW (en inglés World Wide Web Consortium – W3C) define las recomendaciones y estándares que aseguren el crecimiento de la Web a largo plazo, entre los que se encuentra las buenas prácticas en aplicaciones Web Móvil. Este documento realiza unas recomendaciones para el desarrollo de aplicaciones Web en dispositivos móviles [42][48].

Los aspectos dentro de las buenas prácticas en aplicaciones Web Móvil de la W3C, que se tienen en cuenta son las detalladas a continuación, se describe la implementación de cada una de estas prácticas para la elaboración de la implementación de referencia.

2.1.1 Mantener Economía en el uso de red

La persistencia de los datos se realiza en el navegador del mismo dispositivo móvil, con la implementación del marco de trabajo de almacenamiento *Ionic* sin conexión (en inglés, *Ionic Offline Storage*), sin la necesidad de hacer uso de Internet, adicionalmente se aplica el desarrollo de la aplicación Web de una sola página, permitiendo cargar los recursos que componen la página de manera dinámica según lo requiera la aplicación o la intervención del usuario.

2.1.2 Dar libertad a los usuarios

Al usuario final de la aplicación de referencia se le permitirá realizar todas las acciones básicas de la base de datos (Crear, Leer, Actualizar, Borrar), informando al usuario el resultado y proceso de cada funcionalidad. Cuando la aplicación se abre, lo primero que realiza es consultar la información almacenada permitiendo al usuario recordar los datos persistentes.

2.1.3 Recuerde los principios del diseño Web

La aplicación de referencia se apoyará en el marco de trabajo Ionic, permitiendo que la aplicación funcione en múltiples plataformas (iOS, Android) incluyendo los principios de diseño Web centrado en la experiencia del usuario final.

2.1.4 Diseñar aplicaciones flexibles

El marco de trabajo *Ionic* permitirá crear la aplicación de referencia con alta calidad y un alto rendimiento utilizando tecnología web (HTML, CSS y JavaScript). Integrando otro marco de trabajo como Angular (implementa el lenguaje de programación TypeScript) permitirá la compatibilidad de la aplicación con los diferentes navegadores web y plataformas específicas más utilizadas como iOS y Android.

2.1.5 Aprovechar las características de los móviles

Los marcos de trabajo después del proceso de compilar (esto es, compilar de un lenguaje de alto nivel a otro lenguaje de alto nivel), crean artefactos en tecnología web para móviles como HTML5, CSS3, permitiendo aprovechar características de los dispositivos como el tamaño de la pantalla y almacenamiento de datos locales según la persistencia soportada por el navegador.

2.1.6 Optimizar el tiempo de respuesta

La optimización del tiempo de respuesta se ve reflejada en el patrón de diseño de carga diferida (en inglés, *lazy loading*) cargando los componentes cuando son requeridos, permitiendo optimizar el tiempo de carga de la aplicación.

2.2 Desarrollo de software dirigido por modelos

El desarrollo de software dirigido por modelos es un enfoque para la industria del software el cual implementa los modelos como parte importante del desarrollo de artefactos. Un modelo en el desarrollo de software es la representación más abstracta y concreta (Lenguajes específicos de dominio) de los requerimientos del sistema, donde se realizan una serie de transformaciones para llegar a un nivel de detalle deseado [5].

2.3 Arquitectura dirigida por modelos (ADM)

La arquitectura dirigida por modelos (*Model Driven Architecture* - MDA) es un enfoque de la *Object Management Group* para implementar el desarrollo de software dirigido por modelos [7], [8]. Su objetivo es aislar las especificaciones del sistema (diseño de la arquitectura) de su funcionalidad, independiente de su ejecución en cualquier plataforma tecnológica.

2.4 Lenguaje de dominio específico

Al momento de modelar con texto las instrucciones son realizadas con el lenguaje específico de dominio (en inglés *Domain-Specific Language* (DSL)). A diferencia de los lenguajes de propósito general (Java, .NET, entre otros) que son utilizados para muchas cosas, el DSL es el indicado para resolver un problema de dominio específico (un problema particular). La sintaxis concreta del DSL puede ser diseñada con lenguaje de metamodelo textual como Xtext, el cual contiene una colección de sentencias textuales con semántica y sintaxis definida [2].

La estructura de las sentencias del lenguaje de dominio específico debe ser definido por medio de un metamodelo. La semántica del lenguaje se define por medio de una semántica bien definida. El DSL es un lenguaje de un alto nivel de abstracción dejando a un lado los detalles de implementación de bajo nivel y las singularidades de la plataforma de implementación específica.

La ventaja del DSL es que extrae, captura, encapsula, estructura y reutiliza los conocimientos y habilidades del dominio, permitiendo que las soluciones se expresen a nivel de abstracción del dominio del problema. La desventaja es que el experto del dominio define el DSL, dificultando la modificación del lenguaje para quienes no tengan conocimiento específico del dominio.

2.5 Transformación de modelos

En la construcción del lenguaje específico del dominio (DSL) se tiene un modelo con las descripciones del sistema y un alto nivel de abstracción, buscando llegar a un mayor nivel de detalle con las tecnologías seleccionadas para la plataforma específica y realizar transformaciones de manera automática.

Estas transformaciones entre modelos, consisten en convertir un modelo en otro modelo del mismo sistema y permite conocer cómo evolucionan los elementos del sistema en el proceso de desarrollo y como se relacionan.

En la actualidad se encuentran numerosos marcos de trabajo según diferentes autores [18], como se puede observar en la Cuadro 2-1. Propuestas en transformación de modelos, con estándar propuestos por la OMG y basados en diferentes tecnologías.

Lenguajes de transformación de modelo (frameworks)	Propuesto por OMG	basado en MOF	Utiliza XML	UML	Lenguaje de acción AL	Incluye
QVT: (Query/View/Transformations)	x	x				UMLX
JMI: (Java Metadata Interface)		x	x	x		XMI (XML Metadata Interchange)
UML Action Semantics				x	x	Lenguaje de acción AL
GMT: (Generative Modeling Tools)						ATL (ATLAS Transformation Language), VIATRA2 y UMLX
xUML: (eXecutable UML)				x	x	xMUL=UML- "Debilidad Semántica"+AL
VIATRA: (Visual Automated model Transformations)						Soportado por VIATRA2
GREAT: (Graph Rewriting and Transformations)						
XDE: (IBM Rational Rose XDE Modeler)				x		
Stratego						Stratego/XT
BOTL: (The Transformation Language)						
ArcStyler		x	x	x		JMI
Codagen Architect			x	x		
AToM3				x		
KMTL						KMF (Kent Modelling Framework)

Cuadro 2-1. Propuestas en transformación de modelos (Elaboración propia basado en [19] [20] [21] [22] [24] [25] [27] [28] [29] [30] [31])

Las técnicas de transformación difieren en cada uno de los marcos de trabajo, como se puede observar en el cuadro 2-2, donde se presentan los enfoques en las definiciones de reglas funcionales, de importancia en modelos para el desarrollo de software.

Lenguajes de transformación de modelo (frameworks)			Enfoques			
	MDA	DSDM	Manipulación directa	Grafos	Relacionales	híbridos
QVT: (Query/View/Transformations)	x				x	
JMI: (Java Metadata Interface)	x		x			
GMT: (Generative Modeling Tools)		x				
xUML: (eXecutable UML)				x		
VIATRA: (Visual Automated model Transformations)				x		
GREAT: (Graph Rewriting and Transformations)				x		
XDE: (IBM Rational Rose XDE Modeler)						x
BOTL: (The Transformation Language)				x		
ArcStyler	x					
Codagen Architect	x					
AToM3				x		
KMTL		x		x		

Cuadro 2-2. Enfoques en las definiciones de reglas funcionales (Elaboración propia basado en [19] [20] [21] [22] [24] [25] [27] [28] [29] [30] [31])

2.6 Generación de código

Para que un modelo genere como salida texto, es necesario de lenguajes de generación de artefacto tipo texto (En el Cuadro 2-3 se presenta un listado de este tipo de lenguajes), con la finalidad de producir código fuente para aplicaciones de manera automática. El lenguaje que más se implementa es aquel que se apoyado en plantilla (fuente: OMG)

Denominación del lenguaje
<i>Xpand</i>
<i>Aceleo</i>
<i>JET</i>
<i>AndroMDA</i>
<i>MOFScript</i>
<i>Xtend</i>

Cuadro 2-3. Lenguajes de transformación de modelo a texto. (Elaboración propia)

2.7 Planteamientos significativos sobre desarrollo de software dirigido por modelos (DSDM) para aplicaciones móviles

En la literatura se encuentran trabajos de diferentes autores que plantean problemática, solución y ventajas de la implementación como puede verse en la Cuadro 2-4. Trabajos que aplican un enfoque basado en modelos.

AUTORES	PROBLEMA PLANTEADO	SOLUCIÓN	VENTAJA
Charaf, H. [32]	Mayor esfuerzo para el desarrollo de software	La arquitectura específica de dominio proporciona un lenguaje basado en modelos y Modelado (DSML)	Introduce MDA en el desarrollo del software
Según Forstner, B. [33]	Diversidad de plataformas	Visual Modeling y Transformación del sistema (STVM)	Independencia de la plataforma
Choi, Y. [34]	Software de aplicaciones móviles poco estructurado	El marco (framework) se define después de PIM	Definir una arquitectura común para los dispositivos móviles
Jones, V. [35]	Los sistemas móviles de salud pueden ampliar el sistema informático	Metodología aplica un enfoque basado en modelos	Acercar los servicios al paciente en cualquier momento y en cualquier lugar
Khalifa, M.; Verner, J.M. [36]	Factores que afectan el uso de métodos de desarrollo de productos	Mejora de procesos de calidad	Producto de calidad
Jong-Won Ko [37]	Aplicaciones móviles para cada plataforma	Paradigma desarrollo de software MDA y herramientas de transformación de los modelos, tales como la UMT, MTL, ATL	Prueba en el proceso de desarrollo de software en la fase de diseño
Lettner, M. [38]	Alta calidad en el software desarrollado cuando se trata de dispositivos integrados (embedded devices)	Enfoque formal como la arquitectura dirigida por modelo (MDA)	Generación de código de alta calidad para el software de un teléfono móvil de bajo costo
Minovic, M. [39]	Desarrollar interfaces de usuario para los juegos educativos	Enfoque basado en modelos	Demostración práctica del marco de aplicación
Stoyanov, S. [40]	Dispositivo móvil basado en el desarrollo de arquitecturas de sistemas e-learning inteligentes	El enfoque adopta las ideas sugeridas por la especificación MDA de la OMG	Contenidos electrónicos para los usuarios equipados con dispositivos inalámbricos
Kun Yang [41]	Metodología de descubrimiento de servicios nuevos para la próxima generación de sistemas móviles	Modelo basado en el descubrimiento de servicios o MBSD	Generar descripción del servicio de forma automática y precisa

Cuadro 2-4. Trabajos que aplican un enfoque basado en modelos. (Elaboración propia)

Del anterior cuadro se puede observar que los autores introducen MDA en el desarrollo de software para plataformas móviles compatibles. Por otra parte, la arquitectura específica de dominio proporciona un lenguaje con una arquitectura común para la aplicación de software móvil. Los autores examinan los factores que afectan el uso de métodos de desarrollo de productos de calidad en apoyo con las herramientas de transformación.

La Arquitectura Dirigida por Modelos (Model Driven Architecture o MDA), es una propuesta de Desarrollo de software Dirigido por modelos definida por Object Management Group (OMG) [49].

3.Elementos del diseño de la herramienta propuesta

3.1 Dominio del problema

En Colombia el Departamento Nacional de Estadísticas (DANE) es una entidad que estudia el tema de la canasta familiar, compuesta aproximadamente por cuatrocientos elementos o bienes y servicios (arriendo, servicios públicos, alimentos) que son adquiridos de forma habitual por una familia para su sostenimiento. El DANE informa al consumidor las variaciones de los precios. La canasta familiar representa los bienes y servicios mínimos que requiere toda familia para subsistir en condiciones mínimas de calidad de vida, por lo que tiene una gran importancia en lo relativo al aspecto social. La canasta familiar se utiliza para el cálculo del Índice de Precios al Consumidor (IPC), el cual representa o mide el comportamiento del costo de vida, comportamiento que es medido en términos porcentuales. El IPC está construido para captar variaciones de precios de los bienes y servicios representativos en los hogares del país, es decir los incrementos de precio por unidad de producto

3.2 Diagrama de contexto

El diagrama de contexto que se muestra en la figura 3-1 define los establecimientos que suministran la información al sistema y las entidades que interactúan con él a la hora de realizar el IPC para informar al consumidor.

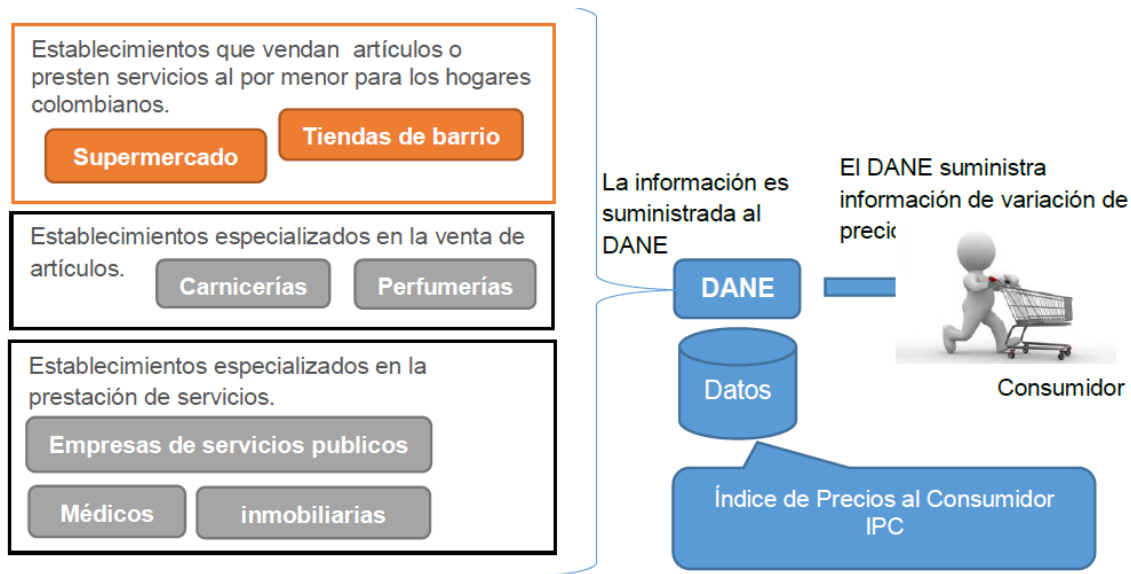


Figura 3-1. Diagrama de contexto. (Elaboración propia)

3.3 Descripción de alcance

La implementación de referencia que se plantea desarrollar es una aplicación Web móvil para identificar las variaciones de los precios de la canasta familiar en los supermercados y tiendas de barrios colombianos.

3.4 Requerimientos funcionales del problema de dominio específico.

El usuario podrá realizar un listado de los bienes que requiere comprar indicando de manera opcional el dinero que presupuesta para la compra. De manera automática el sistema le muestra los precios de los artículos, la variación si la tiene y el total especificando la cantidad de cada bien o artículo. El total de la compra permitirá identificar en cuál de las tiendas o supermercados más cercanos la compra puede ser de menor precio.

El usuario podrá de manera personal suministrar los precios que actualmente tienen los bienes de la canasta familiar, por supermercado y tienda de barrio que más frecuente. El registro de los precios se puede dar debido a que se identificó variación de precio o no se tenía registro del producto y su respectivo precio. La funcionalidad del sistema de información a desarrollar antes indicada se puede observar en el diagrama de descomposición de la figura 3-2.

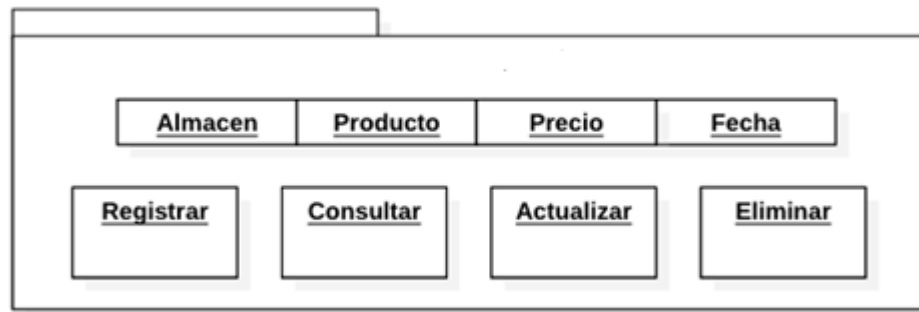


Figura 3-2. Diagrama de descomposición. (Elaboración propia)

3.5 Lenguajes de programación, herramientas y marcos de trabajo que apoyarán el desarrollo de la metodología

3.5.1 Lenguajes de programación

Los lenguajes de programación en que se basa la implementación de referencia son CSS, JavaScript y HTML. El lenguaje de diseño gráfico “Hojas de estilo en cascada” (CSS) definido para crear la presentación de un documento estructurado creado en un lenguaje de marcado, preferiblemente HTML. El lenguaje de marcado de hipertexto (enlaces que comunican a una página con otra) determina el contenido de la página Web, pero no su funcionalidad. HTML consiste en una serie de elementos que se utiliza para marcar diferentes partes del documento de manera anidada.

En cuanto a JavaScript, es un lenguaje de programación interpretado, que se define como multiparadigma: programación orientada a objetos, programación funcional, programación basada en prototipos; imperativo e interpretado. Es un lenguaje débilmente tipado. JavaScript es utilizado hoy en día del lado del cliente y del servidor.

Para el manejo de cifras se debe seleccionar la norma según el área de conocimiento de la tesis o trabajo de investigación.

3.5.2 Marcos de trabajo

Son marcos de trabajo con artefactos o módulos de software concretos en base a lo cual el sistema de información puede ser organizado, desarrollado y desplegado.

- *Angular*: Es un marco de trabajo de aplicaciones Web, mantenido por Google, es muy utilizado para crear y mantener aplicaciones Web de una sola página (SPA). El objetivo del marco de trabajo es que las aplicaciones basadas en navegadores cuenten con la capacidad de Modelo Vista Controlador (MVC). Funciona de la siguiente manera, el marco de trabajo lee el HTML de la página, sus elementos contienen atributos personalizados adicionales que obedecen a sus directivas. (Fuente: <https://angular.io/>)
- *Ionic*: Es un marco de trabajo de código abierto, además de contener un conjunto de herramientas para la interfaz de usuario móvil en el desarrollo de aplicaciones multiplataforma como iOS, Android y Web nativas. (Fuente: <https://ionicframework.com/>)

4. Construcción de la herramienta para generación automática de aplicaciones web móvil dirigido por modelos

4.1 Metodología de trabajo

El desarrollo dirigido por modelos es una metodología donde todas las partes vitales y aspectos del sistema en estudio se describen con modelos [10].

Lo principal del desarrollo de software dirigido por modelos, es que los programadores empleen lenguajes que manejen conceptos más cercanos al dominio de la aplicación de un alto nivel de abstracción, en comparación a los lenguajes de programación de propósito general (LPG).

Esta metodología indica un enfoque de arriba hacia abajo, donde en un primer momento, se analizan las mejores prácticas para el desarrollo de aplicaciones web Móviles. Luego en el desarrollo dirigido por modelos con Xtext y DSL se detectan los elementos comunes de la plataforma de destino, formalizados en un meta-modelo y finalmente se desarrolla y se utiliza para generar el código fuente con algunas reglas de transformación.

La figura 4-1 muestra la descripción de cada paso de la metodología, las actividades y artefactos necesarios.

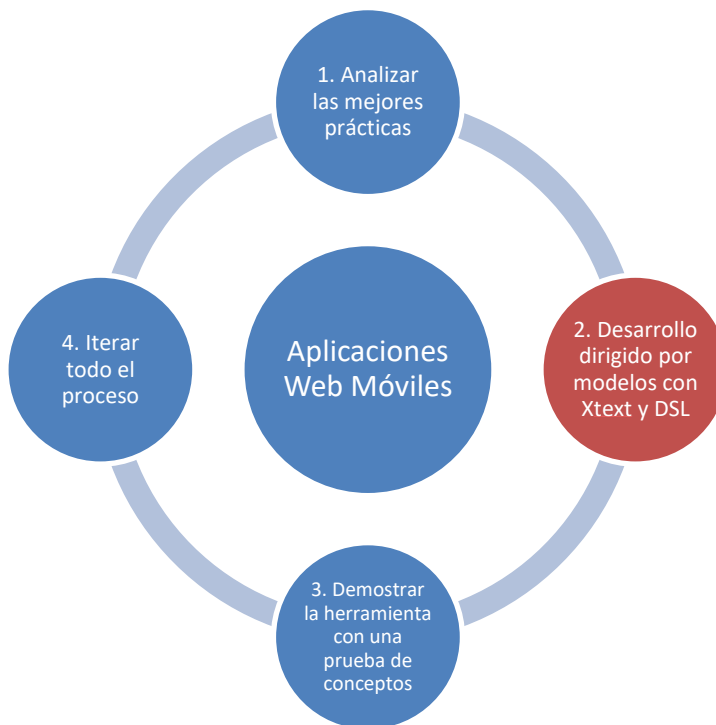


Figura 4-1. Metodología para el desarrollo de la herramienta, iniciando en la parte superior con el ítem 1, siguiendo a la derecha con el ítem 2, realizando luego el ítem 3 y finaliza en ítem 4, el cual indica que se debe iterar todo el proceso. (Elaboración propia)

Para el desarrollo de las aplicaciones Web móviles es necesario analizar las mejores prácticas propuestas por el consorcio W3C, para ser aplicadas en la implementación de referencia y la generación del código automático. El consorcio W3C es el encargado de estandarizar la Web.

La metodología responde a la pregunta, ¿cómo desarrollar una herramienta para la generación automática de aplicaciones web móvil dirigido por modelos?

Para el desarrollo de esta herramienta se sigue una metodología de desarrollo software iterativo incremental, esta metodología permite que pequeñas versiones del software se integren con versiones posteriores, en cada iteración se amplía la funcionalidad del sistema y se mejora el producto final [12]. La construcción de la herramienta, dentro del desarrollo dirigido por modelos, con Xtext y el DSL (figura 4-1) se detalla en la figura 4-2.



Figura 4-2. Metodología de trabajo, inicia en la parte superior con el ítem 1, continuando a la derecha, finalizando con el ítem 6. (Elaboración propia)

4.2 Implementación de referencia

Las tecnologías seleccionadas son los más populares y recomendadas en el mercado, para la implementación de referencia contienen elementos particulares del marco de trabajo Ionic, Angular y el lenguaje de programación *TypeScript*, en cada división lógica de la arquitectura de componentes. La aplicación se compone de tecnologías Ionic (elementos) con Angular (directivas, propiedades y eventos) y objetos (*TypeScript*) que contienen propiedades de redireccionamiento entre los componentes. Los datos se almacenan en memoria con la estructura de datos interfaz, el cual es un tipo de dato del lenguaje *TypeScript*. En la figura 4-3 se describe la arquitectura de la implementación de referencia.

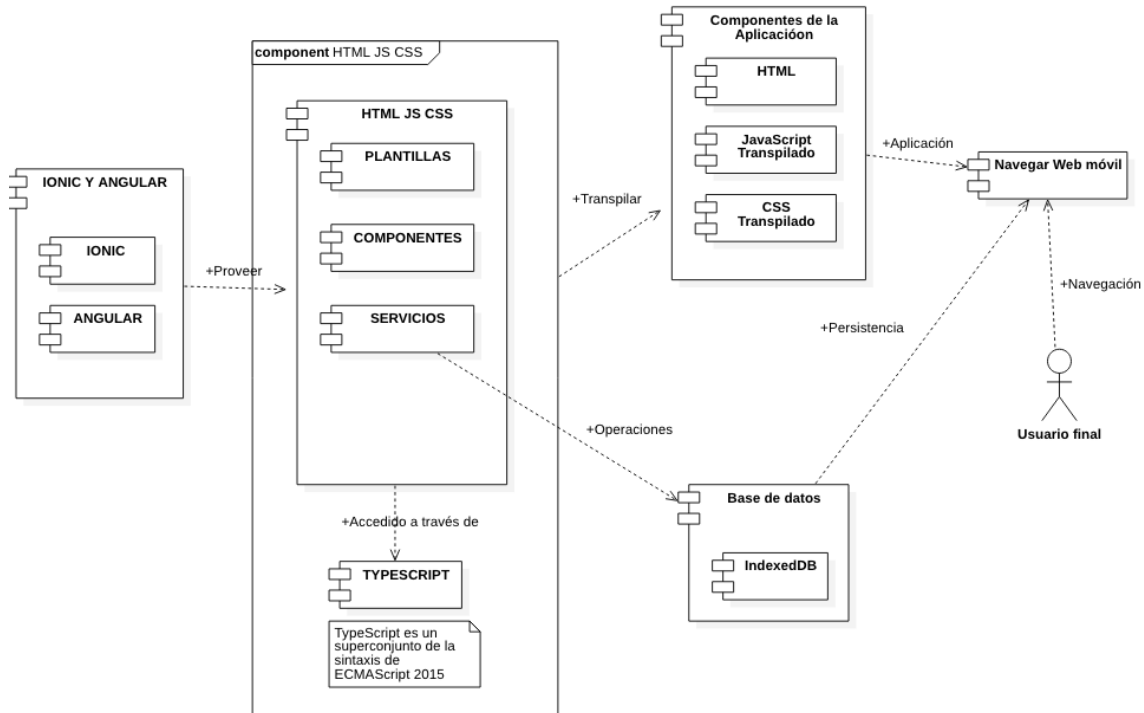


Figura 4-3. Arquitectura de la implementación de referencia. (Elaboración propia)

4.3 Análisis de componentes

En esta parte se hace una verificación de la implementación de referencia para obtener los componentes de la aplicación: Componentes genéricos, Componentes frecuentes, Componentes individuales de la aplicación de referencia.

4.3.1 Componentes genéricos

El componente genérico se identifica con la ejecución de los comandos Ionic para la creación de un proyecto de este marco de trabajo en blanco. El comando que se ejecuta desde la interfaz de línea de comandos en Ionic es:

```
> ionic start AppPrecio blank --type=angular
```

Al ejecutar el comando se creará una carpeta con el nombre "AppPrecio" y dentro de ella una estructura predefinida de carpetas y fichero. El cuadro 4-1 describe la función de cada archivo y carpeta. La figura 4-4 muestra la estructura del proyecto.

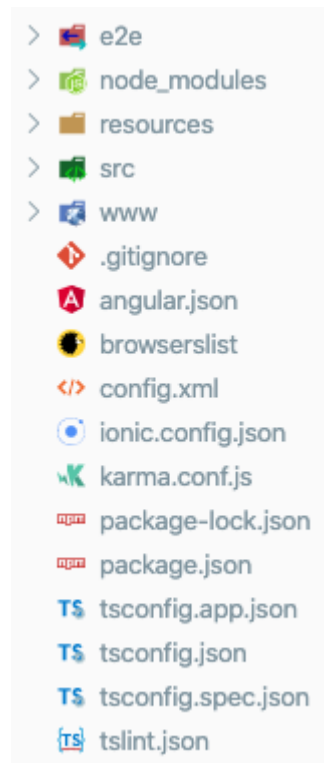


Figura 4-4. Proyecto AppPrecio en Ionic. (Elaboración propia)

Nombre	Archivo/Carpeta	Función
<i>e2e</i>	Carpeta	Contener las pruebas de aceptación de extremo a extremo (<i>test end to end</i>) de la aplicación.
<i>node_modules</i>	Carpeta	Contener los paquetes que son dependencias del proyecto, especificados en el archivo <i>package.json</i>
<i>src</i>	Carpeta	Contener archivos fuentes de la aplicación.
<i>.gitignore</i>	Archivo	Especificar los archivos o carpetas ignorados cuando se sebe en el repositorio Git
<i>angular.json</i>	Archivo	Configuración de la aplicación
<i>browserslist</i>	Archivo	Listar los navegadores para los cuales serán ajustadas las instrucciones CSS Y JavaScript.
<i>ionic.config.json</i>	Archivo	Contener información básica del proyecto.
<i>karma.config.js</i>	Archivo	Especificar la configuración del flujo de trabajo de pruebas para Karma, para las pruebas unitarios.

<i>package-lock.json</i>	Archivo	Presentar un árbol de dependencias incluido en el proyecto.
<i>package.json</i>	Archivo	Configurar las dependencias del gestor de paquetes node que están disponibles para el proyecto.
<i>tsconfig.app.json</i>	Archivo	Configurar el lenguaje TypeScript específica de la aplicación.
<i>tsconfig.json</i>	Archivo	Configurar el lenguaje TypeScript para proyectos.
<i>tsconfig.spec.json</i>	Archivo	Configurar el lenguaje TypeScript para pruebas de la aplicación
<i>tslint.json</i>	Archivo	Configurar la herramienta TSLint, para el análisis de código estático extensible que comprueba el código TypeScript en busca de errores de legibilidad, mantenibilidad y funcionalidad.

Cuadro 4-1. Componentes genéricos del proyecto. (Elaboración propia)

Dentro del proyecto se encuentra la carpeta SRC la cual contiene los archivos con el contenido de la aplicación, que es donde el desarrollador invierte la mayor parte de su trabajo. Las subcarpetas contienen el código principal y la configuración específica de la aplicación. La carpeta y los archivos se visualizan en la figura 4-5. El cuadro 4-2 describe la función de cada archivo y carpeta de la carpeta SRC.

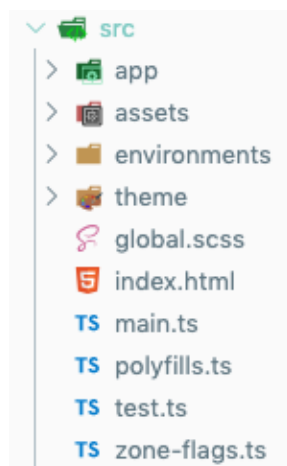


Figura 4-5. Carpeta principal donde se almacena. (Elaboración propia)

Nombre	Archivo/Carpeta	Función
<i>app</i>	Carpeta	Contener los archivos creadas por los desarrolladores para la aplicación y la lógica de programación.
<i>assets</i>	Carpeta	Almacenar recursos necesarios para la aplicación como imágenes, videos y ficheros de traducción que no son propiedad de ningún componente.
<i>environments</i>	Carpeta	Definir variables de entorno en archivo de producción o desarrollo
<i>theme</i>	Carpeta	Contener el fichero de estilos globales para la plantilla.
<i>global.scss</i>	Archivo	Reglas de estilos para toda la aplicación, importar archivos CSS/SASS que se incluirán en salida CSS.
<i>index.html</i>	Archivo	Página HTML principal y punto de inicio de la aplicación.
<i>main.ts</i>	Archivo	Modulo principal de la aplicación y primer código que se ejecuta.
<i>polyfills.ts</i>	Archivo	Instrucciones que hacen que la aplicación sea compatible con diferentes navegadores.
<i>test.ts</i>	Archivo	Cargar recursivamente los archivos con extensión. spec; archivo requerido por karma.conf.js
<i>zone-flags.ts</i>	Archivo	Evita que la detección de cambios angulares se ejecute con ciertas devoluciones de llamada de componentes web

Cuadro 4-2. Componentes genéricos de la carpeta src. (Elaboración propia)

Dentro de la carpeta *app* se encuentra toda la lógica y los datos del proyecto, los componentes, las plantillas, los estilos, módulos y restos de elementos que construyen la aplicación. La figura 4-6 muestra los archivos genéricos para la aplicación. El cuadro 4-3 describe la función de cada archivo de la carpeta *app*.

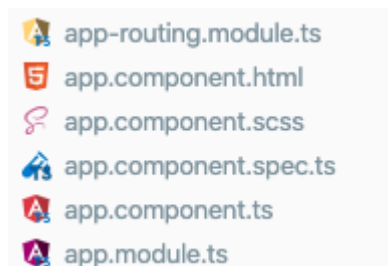


Figura 4-6. Archivos dentro de la carpeta *app*. (Elaboración propia)

Archivo	Función
<i>app-routing.module.ts</i>	Archivo principal de rutas para incorporar la carga perezosa (<i>Lazy Loading</i>).
<i>app-component.html</i>	Define la plantilla HTML asociada con el root AppComponent.
<i>app.component.scss</i>	Contener todas las variables y estilos de Sass (metalenguaje de hojas de estilo que se traduce a css) para ser usado globalmente dentro de la aplicación.
<i>app.component.spec.ts</i>	Realizar pruebas
<i>app.component.ts</i>	Definir partes que interesan tener al inicio de la aplicación como el idioma por defecto.
<i>app.modules.ts</i>	Cargar componentes y módulos útiles para las páginas además de controlar los recursos utilizados por la aplicación.

Cuadro 4-3. Componentes genéricos de la carpeta app. (Elaboración propia)

4.3.2 Componentes frecuentes

Con el proyecto AppPrecio creado en blanco los artefactos frecuentes son componente (*component*), página (*page*), servicio (*service*) y directiva (*directive*). El cuadro 4-4 describe la función de cada tipo.

Tipo	Descripción
Página	<p>Es la creación de una nueva página con los archivos: <i>xxxx-routing.module.ts</i> <i>xxxx.module.ts</i> <i>xxxx.page.scss</i> <i>xxxx.page.html</i> <i>xxxx.page.spec.ts</i> <i>xxxx.page.ts</i> Donde xxxx es el nombre de la nueva página.</p>
Componente	<p>Es una pieza de código que se puede utilizar en cualquier parte de la aplicación, mediante el cual se construyen los elementos y la lógica de la página, crea los archivos: <i>xxxx.component.scss</i> <i>xxxx.component.html</i> <i>xxxx.component.spec.ts</i> <i>xxxx.component.ts</i> Donde xxxx es el nombre de la nueva componente.</p>
Directiva	<p>Es un modificador de atributos para ser utilizado en cualquier elemento de la aplicación, añade comportamiento dinámico al HTML, crea los archivos <i>xxxx.directive.spec.ts</i> <i>xxxx.directive.ts</i> Donde xxxx es el nombre de la nueva directiva.</p>
Servicio	<p>Es un nuevo servicio, encargado de manipular los datos en localstore o SQLite, crea los archivos: <i>xxxx.service.spec.ts</i> <i>xxxx.service.ts</i> Donde xxxx es el nombre del nuevo servicio.</p>
Clase	<p>Agrega una clase a la aplicación <i>xxxx.spec.ts</i> <i>xxxx.ts</i> Donde xxxx es el nombre de la nueva clase.</p>
Modulo	<p>Permite agrupar componentes, servicios, directivas, etc. en la aplicación, crea los archivos: <i>xxxx.module.ts</i> Donde xxxx es el nombre del nueva modulo.</p>

Cuadro 4-4. Componentes frecuentes. (Elaboración propia)

4.3.3 Componentes individuales

Estos comprenden los artefactos específicos de la implementación de referencia. El cuadro 4-5 describe la función de cada artefacto.

Tipo	Identificador	Descripción
Página	<i>home.module.ts</i> <i>home.page.html</i> <i>home.page.scss</i> <i>home.page.spec.ts</i> <i>home.page.ts</i> <i>app-routing.module.ts</i>	Página de la aplicación.
Componente	<i>app.component.html</i> <i>app.component.scss</i> <i>app.component.spec.ts</i> <i>app.component.ts</i>	Componentes principales.
Servicio	<i>storage.service.spec.ts</i> <i>storage.service.spec.ts</i>	Gestiona los datos en el localstore o SQLite según la plataforma.
Interface	Item	Creación de un nuevo tipo con sus propiedades descritas y tipadas para hacer persistencia de datos.
Modulo	<i>home.module.ts</i> <i>app.module.ts</i>	Organizar las partes de la aplicación en bloques, además permite extender la aplicación con librerías externas que podrán ser reutilizados en la aplicación.

Cuadro 4-5. Componentes individuales. (Elaboración propia)

4.4 Metamodelo

El metamodelo para la implementación de referencia (AppPrecio) tiene tres artefactos principales: Dominio, Arquitectura y Tecnología (Figura 4-7). El artefacto raíz es la aplicación “AppPrecio” la cual puede tener cero (0) “Dominio” o un (1) “Dominio”, cero (0) “Arquitectura” o una (1) “Arquitectura”, cero (0) “Tecnología” una (1) “Tecnología”.

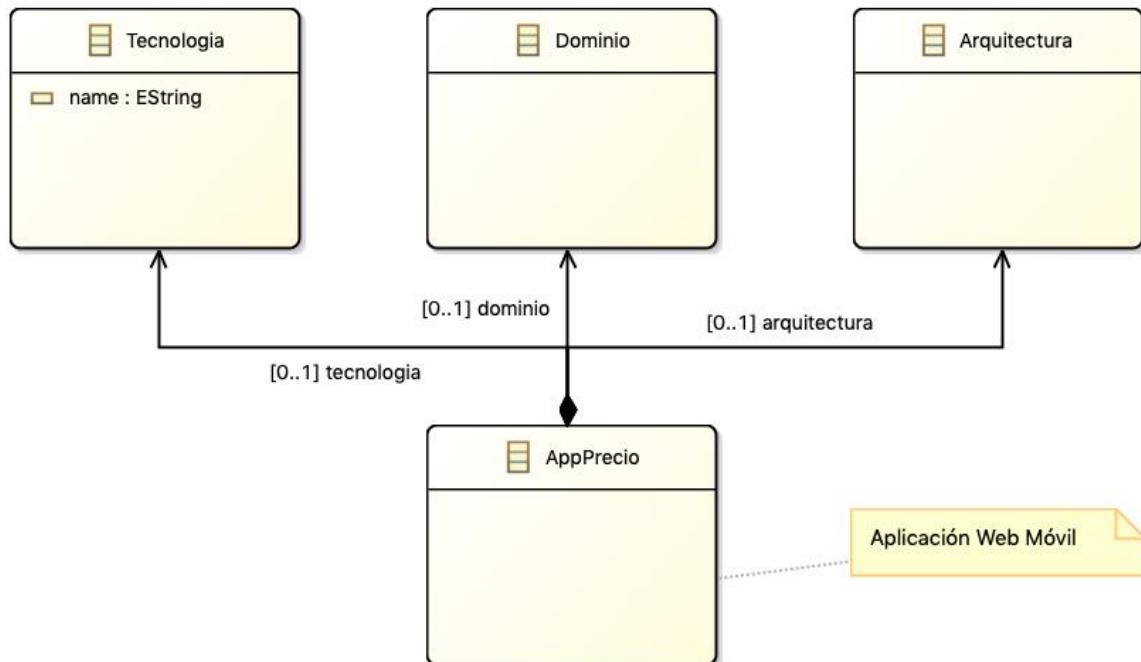


Figura 4-7. Elementos principales del metamodelo. (Elaboración propia)

4.4.1 Metamodelo del dominio

El metamodelo del “Dominio” (Figura 4-8) muestra el artefacto de la definición del dominio de negocio: el “Servicio” y la “Entidad”. El servicio encapsula la funcionalidad principal (Métodos) para los demás componentes, como registrar, consultar, eliminar y actualizar los datos de los productos de la canasta familiar, haciendo la persistencia de los datos en el mismo dispositivo móvil. La “Entidad” es la estructura de datos que apoya el almacenamiento de los datos en memoria.

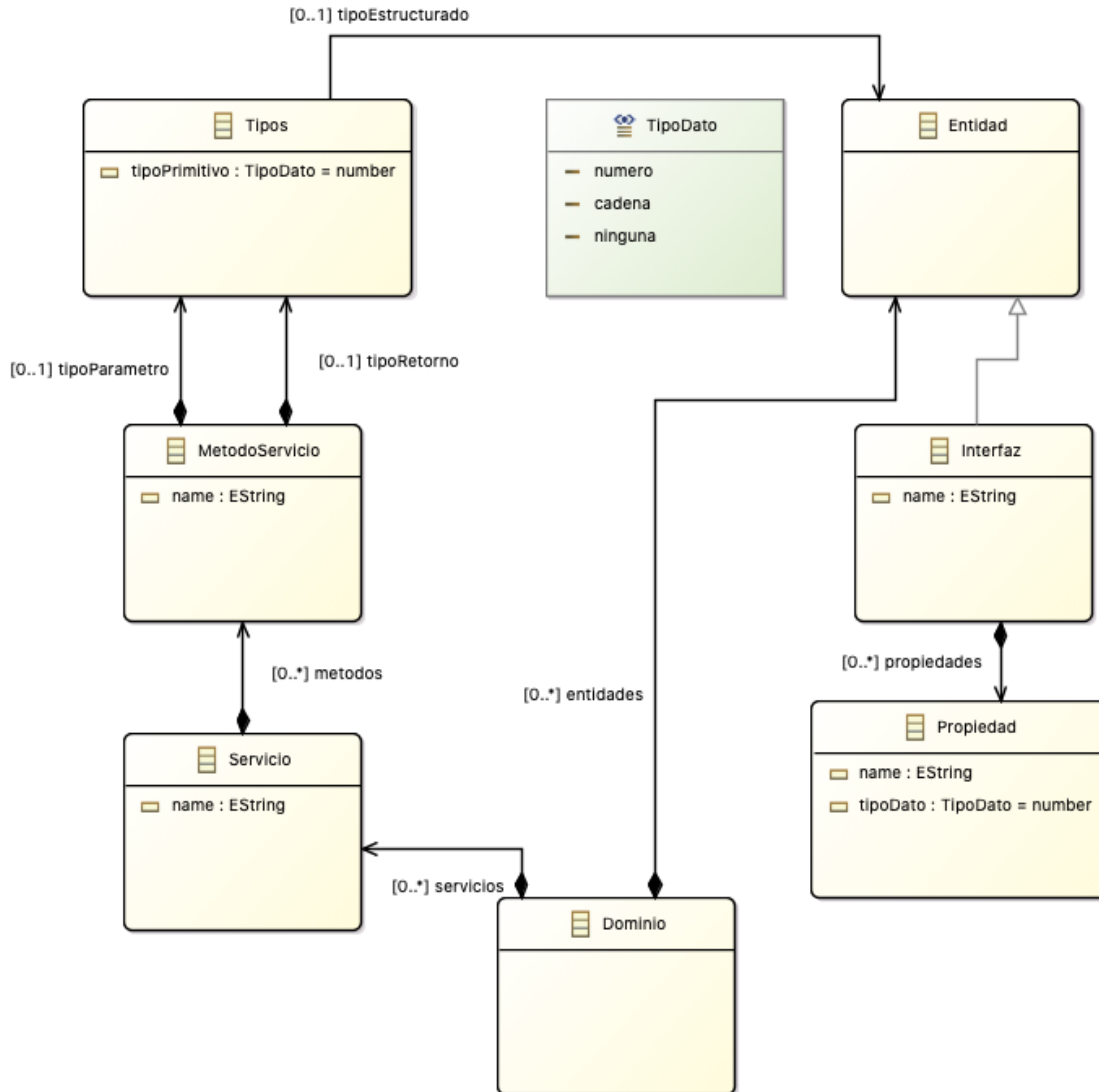


Figura 4-8. Metamodelo del dominio. (Elaboración propia)

4.4.2 Metamodelo de la arquitectura

El metamodelo de la “Arquitectura” (Figura 4-9) identifica la arquitectura de la aplicación en “Componentes”. Los metadatos de un componente le indican a Angular dónde obtener los bloques de construcción necesarios para crear y presentar el componente y su vista.

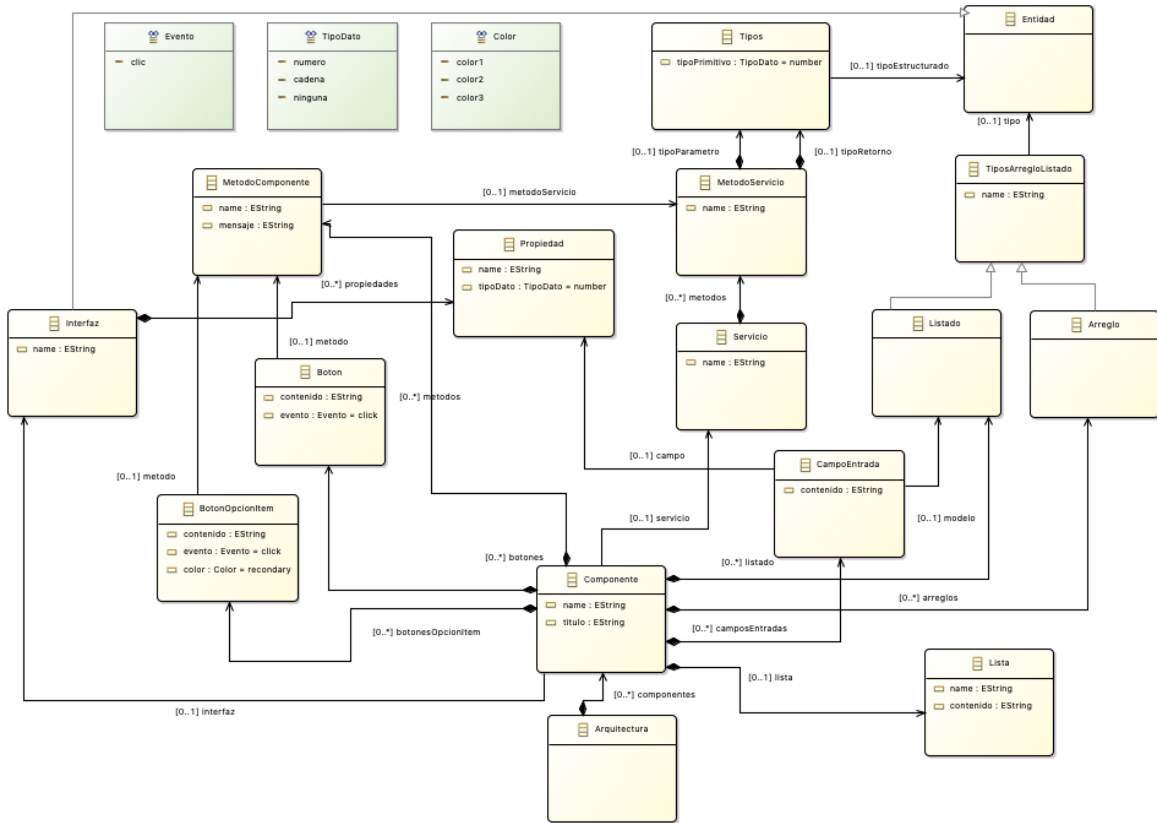


Figura 4-9. Metamodelo de la arquitectura. (Elaboración propia)

4.4.3 Metamodelo de tecnología

El metamodelo de “Tecnología” (Figura 4-10) contiene artefactos particulares del marco de trabajo Ionic para el almacenamiento de datos multiplataforma sin conexión a la red (Almacenamiento fuera de línea – Offline Storage en inglés).

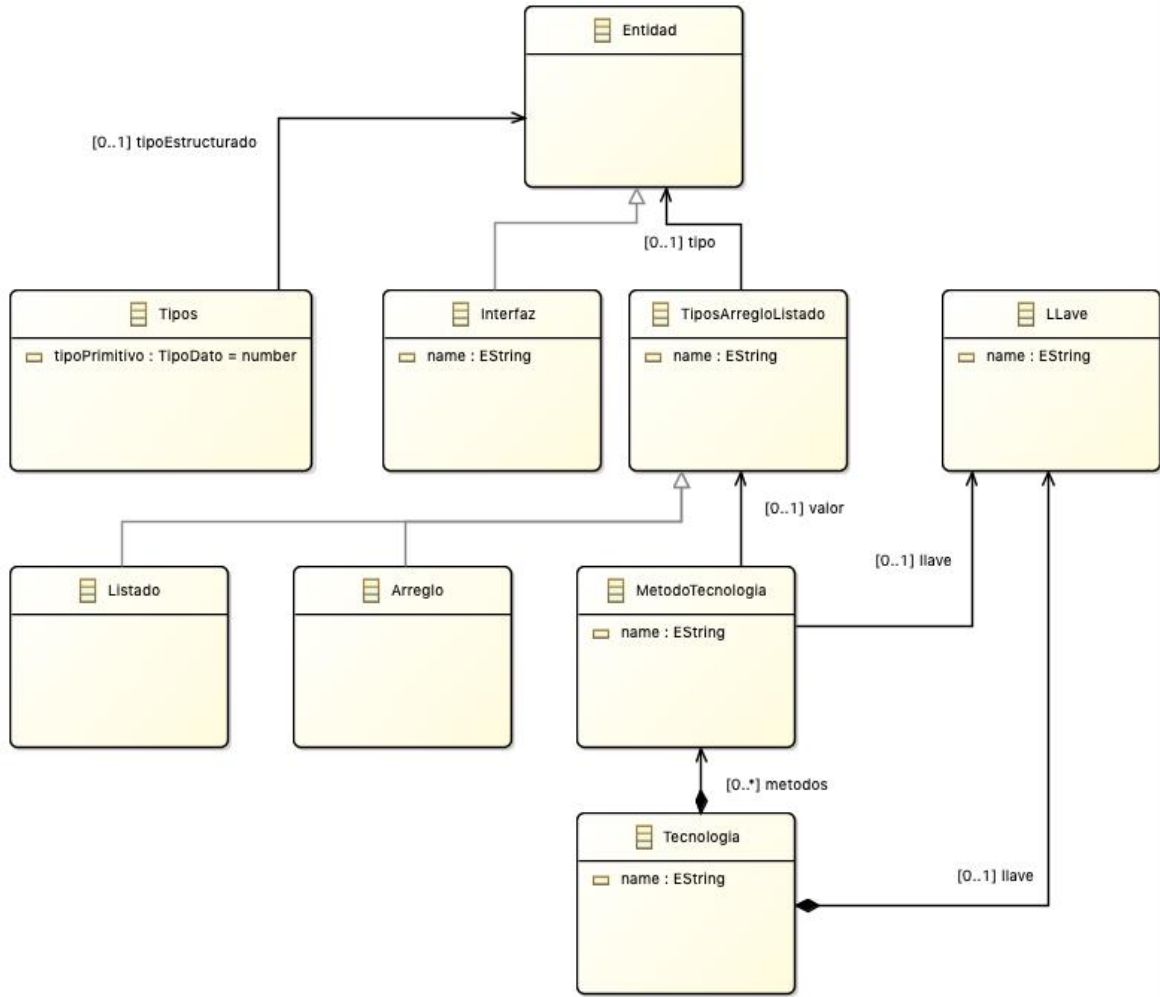


Figura 4-10. Metamodelo de la tecnología. (Elaboración propia)

4.5 Diseño del DSL.

Se determina la gramática del lenguaje de dominio específico (DSL) para modelar la aplicación de referencia (AppPrecio) presentada en árboles de sintaxis concreta. La gramática a continuación se define según el “Dominio”, “Arquitectura” y “Tecnología” de la especificación de referencia en la Figura 4-11.

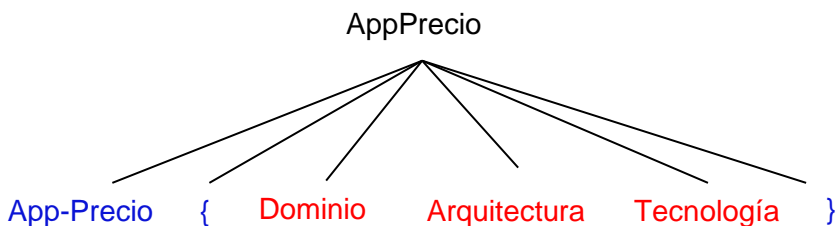


Figura 4-11. Árbol de sintaxis concreta general. (Elaboración propia)

4.5.1 Dominio

El dominio contiene los artefactos: “Componente”, “Servicio” (Servicio de almacenamiento), y Entidad (estructura de datos). Cada artefacto permite definir el dominio de negocio para crear una aplicación de negocio lo que involucra una CRUD (en inglés: *Create, Read, Update, Delete*. Acrónimo de Crear, Leer, Actualizar y Borrar), para el almacenamiento local en dispositivos móviles con su respectiva interfaz gráfica. En la siguiente figura podemos ver el árbol de la sintaxis concreta del dominio (figura 4-12).

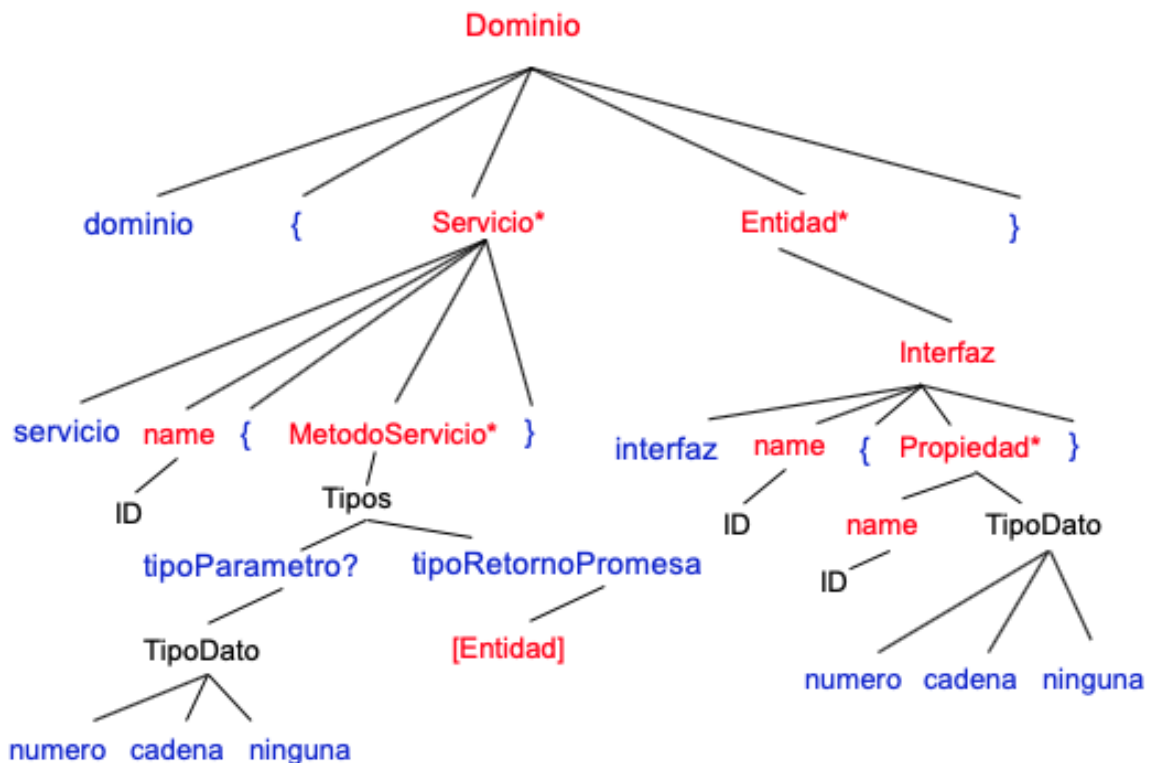


Figura 4-12. Árbol de sintaxis concreta del dominio. (Elaboración propia)

- Servicio

El servicio es una clase que encapsula las operaciones (CRUD) para la persistencia de la información (ver figura 4-12).

- Entidad

La entidad es una estructura de datos que contiene sus propiedades fuertemente tipados; puede contener de cero o más propiedades (ver figura 4-12).

4.5.2 Arquitectura

La gramática de la arquitectura consta de los artefactos arquitectónicos basada en componentes para obtener componente de software débilmente acoplados, apoyada en las características de los entornos de trabajo *Ionic* y *Angular*. Podemos ver el árbol de la sintaxis en la figura 4-13. De la misma manera se detallan los árboles de los componentes de arquitectura de las figuras 4-14 a 4-18.

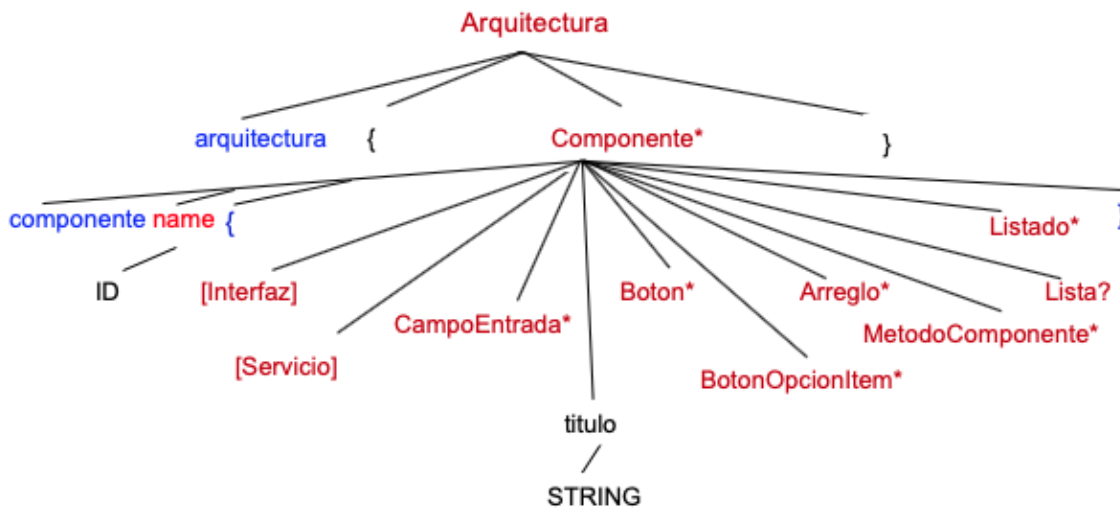


Figura 4-13. Árbol de sintaxis concreta de la arquitectura. (Elaboración propia)

- Interfaz y Servicio

El elemento “Componente*” implementa “Interfaz” y “Servicio” como referencia cruzada, la cual es implementada y descrita en el dominio (ver figura 4-12).

- CampoEntrada

Los campos de entrada para la aplicación, es donde se captura toda la información suministrada por el usuario (ver figura 4-14).

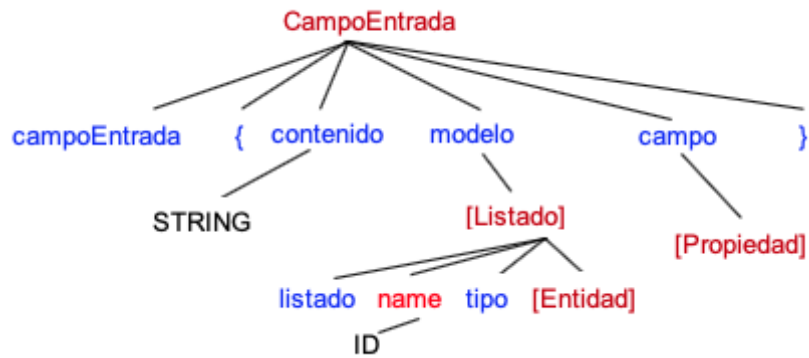


Figura 4-14. Árbol de sintaxis concreta de la Arquitectura – *CampoEntrada*. (Elaboración propia)

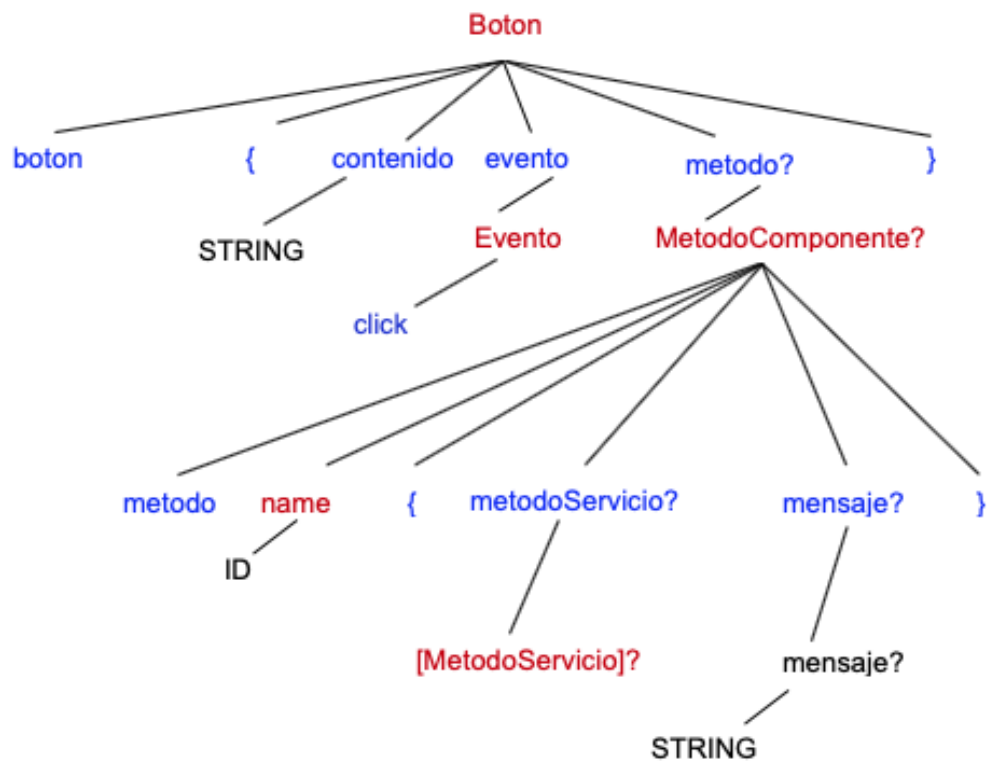


Figura 4-15. Árbol de sintaxis concreta de la Arquitectura – Botón. (Elaboración propia)

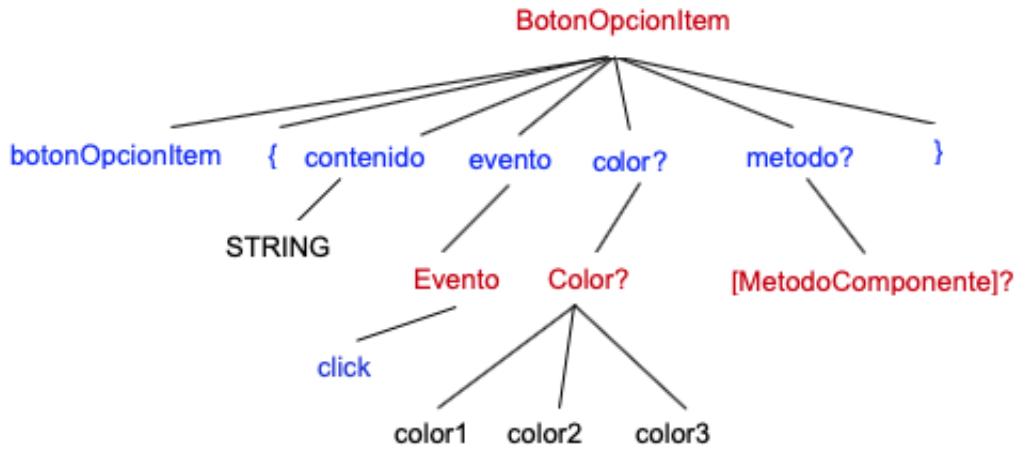


Figura 4-16. Árbol de sintaxis concreta de la Arquitectura – *BotonOpcionItem*. (Elaboración propia)

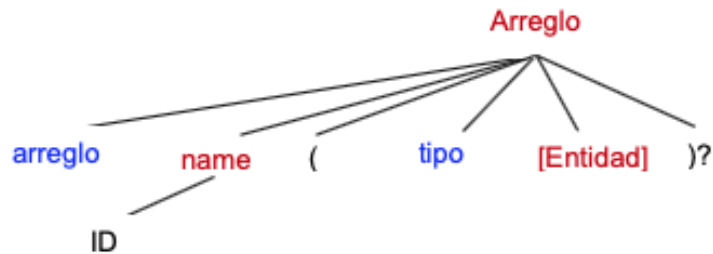


Figura 4-17. Árbol de sintaxis concreta de la Arquitectura – Arreglo. (Elaboración propia)



Figura 4-18. Árbol de sintaxis concreta de la Arquitectura – Listado. (Elaboración propia)

4.5.3 Tecnología

La gramática de la tecnología permite identificar parámetros de configuración de los módulos, se muestra a continuación en la Figura 4-19. En la figura 4-19 se presenta el elemento MetodoTecnologia del componente tecnología

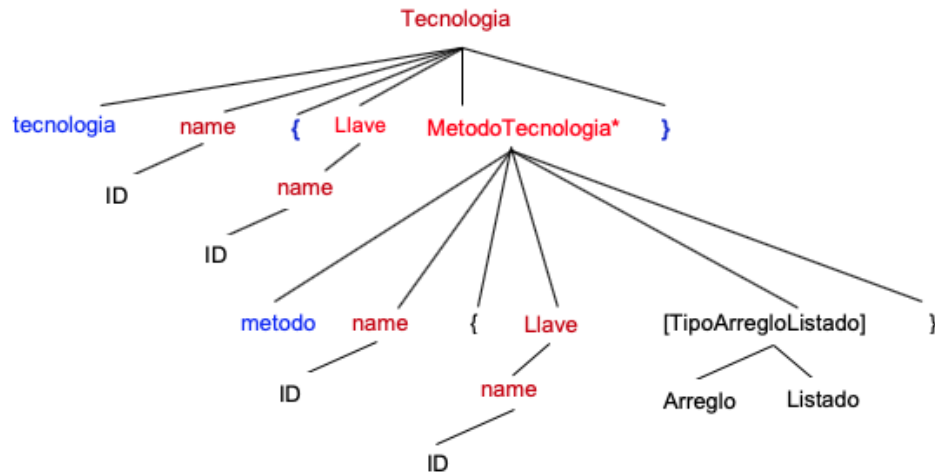


Figura 4-19. Árbol de sintaxis concreta de la tecnología. (Elaboración propia)

4.6 Gramática del lenguaje específico de dominio (DSL) en Xtext

Se determina la gramática del lenguaje de dominio específico (DSL) para modelar la aplicación de referencia (AppPrecio) presentada en la definición del lenguaje Xtext. La gramática a continuación se define según el dominio, arquitectura y tecnología de la especificación de referencia (Ver anexo A).

La gramática inicia con el nombre del lenguaje (co.edu.unal.AppPrecio) (Ver anexo A) extendida con la gramática básica de Xtext (org.eclipse.xtext.common.Terminals), la cual reúne las reglas elementales de la definición de gramáticas Xtext. Con “generate” los analizadores Xtext crean gráficos de objetos en memoria mientras consumen texto. Se definen los artefactos que componen la aplicación en el lenguaje Xtext.

La aplicación de referencia AppPrecio contiene las siguientes características: un Dominio, una Arquitectura y una Tecnología. Ver anexo A.

4.7 Desarrollo de transformaciones

Para la generación automática de código se deben realizar las transformaciones de la gramática de acuerdo a los artefactos específicos a la plataforma a generar, como lo describe el cuadro 4-6.

Paquete	Nombre	Descripción
Aplicación Web Móvil	AppPrecio	Aplicación subdividida, para las transformaciones de “Arquitectura”, “Dominio” y “Tecnología”.
	Dominio	Usa las transformaciones de “Servicio” y “Entidad”.
	Tecnologia	Usa las transformaciones de “Llave” y “MetodoTecnologia”
	Arquitectura	Usa las transformaciones de los componentes
Dominio	Servicio	Genera los metodos para escritura, lectura, actualización y eliminación que interactuaran con el almacenamiento fuerade conexión multiplafarma de Ionic, con sus respectivas promesas. Los metodo en sus parametros reciben y retornan el tipo de dato “Entidad” en su gran mayoría. Genera el archivo <i>storage.service.ts</i>
	Entidad	Genera la estructura de datos Interfaz en el lenguaje de programación TypeScript
Tecnologia	Llave	Genera la llave para gestionar la persistencia de los datos
	MetodoTecnologia	Genera los metodos que seran implementados en la transformación.
Arquitectura	Co mponentes	Genera todos los componentes de la aplicación y los archivos <i>home.module.ts</i> , <i>home.page.html</i> , <i>home.page.scss</i> , <i>home.page.ts</i>
	CamposEntrada	Genera el componente Angular para las etiquetas Ionic
	MetodoComponete	Genera la estructura y loc. Metodos para las operaciones CRUD implementando cada uno de los metodos de los servicios.

Cuadro 4-6. Descripción de las transformaciones. (Elaboración propia)

4.7.1 Definición de las transformaciones

Con la realización del modelo específico del dominio, se realizan las transformaciones de modelo a texto con el lenguaje Xtend, donde los artefactos generados en la generación de código son instrucciones detalladas de la implementación de referencia. En el anexo C se

detallan las definiciones de las transformaciones en las etapas DSL, de modelo a texto y Generación de código.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Modela el Componente del modulo	Se crea el componente "Home"	Compila el componente del módulo, con la primera letra en mayúscula: name	Todo el código fuente del modulo
Modelado en el lenguaje Xtext			
Componente: <pre>'componente' name=ID '{ }'</pre>			
Lenguaje específico de dominio (DSL)			
<pre>arquitectura { componente Home {} }</pre>			
Transformación de modelo a texto con Xtend			
<pre>def CharSequence compileModule(Componente c)""" import { «c.name.toFirstUpper»Page } from './«c.name.toLowerCase».page'; component: «c.name.toFirstUpper»Page declarations: [«c.name.toFirstUpper»Page] export class «c.name.toFirstUpper»PageModule {} """</pre>			
Código fuente generado			
<pre>import { HomePage } from './home.page'; component: HomePage declarations: [HomePage] export class HomePageModule {}</pre>			

Cuadro 4-7. Definición de las transformaciones – del "Modulo".

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Modela los elementos de la página Web: CampoEntrada, Boton, Lista, Interfaz, Arreglo, BotonOpcionItem	Se crea el componente "Home" con la propiedad de interfaz = "Item", el arregle "items" del	Compila los componentes con sus elementos, para algunas instrucciones se debe iterar entre la lista de elementos del mismo tipo: camposEntradas, titulo,	Todo el código fuente de la página HTML

	tipo "Item" y la lista con el nombre "myList"	botones, lista, interfaz, arreglos, botonesOpcionItem.	
Modelado en el lenguaje Xtext			
<pre> Dominio: {Dominio} 'dominio' '{ (entidades+=Entidad)* } ; Entidad: Interfaz ; Interfaz: 'interfaz' name=ID '{ (propiedades+=Propiedad)* } ; Lista: 'lista' name=ID '{ 'contenido' ':' contenido=STRING } ; Arreglo: 'arreglo' name=ID ('tipo' tipo=[Entidad])? ; Componente: 'componente' name=ID '{ 'interfaz' ':' interfaz=[Interfaz] 'titulo' ':' titulo=STRING (camposEntradas+=CampoEntrada)* (botones+=Boton)* (botonesOpcionItem+=BotonOpcionItem)* (arreglos+=Arreglo)* lista=Lista? } ; </pre>			
Lenguaje específico de dominio (DSL)			
<pre> dominio { interfaz Item { id : number almacen : string producto: string precio : number modifica: number } } arquitectura { componente Home { interfaz : Item titulo : "Productos de la canasta familiar" arreglo items tipo Item lista myList { contenido : "Productos registrados" } } } </pre>			
Transformación de modelo a texto con Xtend			

```
def CharSequence compileHtml(Componente c)"""
    <ion-title><c.titulo></ion-title>

    <ion-content>

    «FOR ce : c.camposEntradas»
        «ce.compileCamposEntrada»
    «ENDFOR»

    «FOR cb : c.botones»
        «cb.compileBotones»
    «ENDFOR»

    <ion-list #«c.lista.name»>

    <ion-list-header>
        <ion-label><c.lista.contenido></ion-label>
    </ion-list-header>

    <ion-item-sliding *ngFor="let item of «c.arreglos.get(0).name»">
        <ion-item>
            <ion-label class="ion-text-wrap">
                «FOR p : c.interfaz.propiedades»
                    «IF p.name.equals("almacen")»
                        <h2>{{ item.«p.name»}}</h2>
                    «ENDIF»
                    «ENDFOR»
                    «FOR p : c.interfaz.propiedades»
                        «IF p.name.equals("producto")»
                            <h1>{{ item.«p.name» | uppercase }}</h1>
                        «ENDIF»
                    «ENDFOR»
                    «FOR p : c.interfaz.propiedades»
                        «IF p.name.equals("precio")»
                            <ion-text color="primary">
                                <p>{{ item.«p.name» | currency:'USD': '$': '1.0'}}</p>
                            </ion-text>
                        «ENDIF»
                    «ENDFOR»
                    «FOR p : c.interfaz.propiedades»
                        «IF p.name.equals("modifica")»
                            <p>{{ item.«p.name» | date:'short' }}</p>
                        «ENDIF»
                    «ENDFOR»
                </ion-label>
            </ion-item>

            <ion-item-options>
                «FOR cboi : c.botonesOpcionItem»
                    «cboi.compileBotonesOpcionItem»
                «ENDFOR»
            </ion-item-options>

        </ion-item-sliding>
    </ion-list>

</ion-content>
```

Código fuente generado
<pre> ''' <ion-tittle> Productos de la canasta familiar </ion-tittle> <ion-content> <ion-list #myList> <ion-list-header> <ion-label>Productos registrados</ion-label> </ion-list-header> <ion-item-sliding *ngFor="let item of items"> </ion-item-sliding> </ion-list> </ion-content> ''' </pre>

Cuadro 4-8. Definición de las transformaciones – de los artefactos “CampoEntrada”, “titulo”, “CamposEntradas”, “compileBotones”, “Lista”.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
CampoEntrada, Boton, BotonOpcionItem.	Se crean los campos de entrada: “Almacen”, “Producto”, “Precio”; se crea el botón “Agregar”; se crea el servicio “storage” con la interfaz “Item”	Compila los elementos específicos del componente principal: contenido, modelo, campo, evento, color	Elementos de la página HTML
Modelado en el lenguaje Xtext			
<pre> Listado: 'listado' name=ID 'tipo' tipo=[Entidad] ; enum TipoDato: numero='number' cadena='string' ninguna='any' ; Entidad: Interfaz ; Interfaz: 'interfaz' name=ID '{ (propiedades+=Propiedad)* }' ; Propiedad: name=ID ':' tipoDato=TipoDato ; CampoEntrada: 'campoEntrada' '{ 'contenido' ':' contenido=STRING 'modelo' ':' modelo=[Listado] 'campo' ':' campo=[Propiedad] }' ; Tipos: tipoPrimitivo=TipoDato tipoEstructurado=[Entidad] ; MetodoServicio: </pre>			


```

    }
    campoEntrada {
        contenido : "Precio"
        modelo : newItem
        campo : precio
    }
    boton {
        contenido : "Agregar"
        evento : click
        metodo : addItem
    }
    metodo addItem {
        MetodoServicio : add
        mensaje : "Producto agregado"
    }
    metodo loadItems {
        MetodoServicio : getAll
    }
    metodo updateItem {
        MetodoServicio : update
        mensaje : "Producto Actualizado!"
    }
    metodo deleteItem {
        MetodoServicio : delete
        mensaje : "Producto eliminado"
    }
}

```

Transformación de modelo a texto con Xtend

```

def CharSequence compileCamposEntrada(CampoEntrada ce)""
    <ion-item>
        <ion-label position="fixed"><ce.contenido></ion-label>
        <ion-input [(ngModel)]=<ce.modelo.name>,<ce.campo.name>></ion-input>
    </ion-item>
    ""
def CharSequence compileBotones(Boton b)""
    <ion-button expand="full" (<b.evento>)=<b.metodo.name>()>
    <b.contenido>
    </ion-button>
    ""
def CharSequence compileBotonesOpcionItem(BotonOpcionItem b)""
    <ion-item-option color=<b.color>""
    (<b.evento>)=<b.metodo.name>(item)">
    <b.contenido>
    </ion-item-option>
    ""

```

Código fuente generado

```

<ion-header>
<ion-toolbar color="primary">
<ion-title>
    Productos de la canasta familiar
</ion-title>
</ion-toolbar>
</ion-header>
<ion-content>
<ion-item>
    <ion-label position="fixed">Almacen</ion-label>
    <ion-input [(ngModel)]=<newItem.almacen></ion-input>
</ion-item>
</ion-item>

```

```

<ion-label position="fixed">Producto</ion-label>
<ion-input [(ngModel)]=newItem.producto</ion-input>
</ion-item>
<ion-item>
<ion-label position="fixed">Precio</ion-label>
<ion-input [(ngModel)]=newItem.precio</ion-input>
</ion-item>
<ion-button expand="full" (click)="addItem()">Agregar</ion-button>
<ion-list #myList>
<ion-list-header>
<ion-label>Productos registrados</ion-label>
</ion-list-header>
<ion-item-sliding *ngFor="let item of items">
<ion-item>
<ion-label class="ion-text-wrap">
<h2>{{ item.almacen}}</h2>
<h1>{{ item.producto | uppercase }}</h1>
<ion-text color="primary">
<p>{{ item.precio | currency:'USD': '$':'1.0'}}</p>
</ion-text>
<p>{{ item.modifica | date:'short' }}</p>
</ion-label>
</ion-item>
<ion-item-options>
<ion-item-option color="secondary" (click)="updateItem(item)">
Actualizar
</ion-item-option>
<ion-item-option color="danger" (click)="deleteItem(item)">
Eliminar
</ion-item-option>
</ion-item-options>
</ion-item-sliding>
</ion-list>
</ion-content>

```

Cuadro 4-9. Definición de las transformaciones – de los artefactos “CampoEntrada”, “Boton”, “BotonOpcionItem”.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Modelo del Componente: Componente, Arreglos, Interfaz, Listado, Lista, Servicio, MetodoComponente.	Se hace referencia a la interfaz “Item”, el servicio “storage”. Se crean los arreglos “items” y “newItems” del tipo “Item”	Compila los elementos del componente: name, arreglos, interfaz, listado, lista, servicio, metodos	Código fuente del componente (@Component), clase del componente
Modelado en el lenguaje Xtext			
Componente:			
<pre> 'componente' name=ID '{ 'interfaz' ':' interfaz=[Interfaz] 'servicio' ':' servicio=[Servicio] (listado+=Listado)* lista=Lista? }' </pre>			
Lenguaje específico de dominio (DSL)			

```

arquitectura {
    componente Home {
        interfaz : Item
        servicio : storage
        arreglo items tipo Item
        arreglo newItems tipo Item
        listado newItem tipo Item
    }
}

```

Transformación de modelo a texto con Xtend

```

def CharSequence compileComponent(Componente c)'''

@Component({
    selector: 'app-«c.name.toLowerCase»',
    templateUrl: «c.name.toLowerCase».page.html',
    styleUrls: [«c.name.toLowerCase».page.scss],
})

export class «c.name.toLowerCase»Page {
    «FOR a : c.arreglos»
        «IF a.name.equals("items")»
            «a.name»: «c.interfaz.name» [] = [];
        «ENDIF»
    «ENDFOR»

    «FOR l : c.listado»
        «IF l.name.equals("newItem")»
            «l.name»: «c.interfaz.name» = <«c.interfaz.name»>{};
        «ENDIF»
    «ENDFOR»

    @ViewChild(«c.lista.name.toLowerCase»',
    {static: true}) mylist: IonList;

    constructor(private «c.servicio.name.toLowerCase»Service:
    StorageService,
    private plt: Platform, private toastController: ToastController) {
        this.plt.ready().then(=>{
            «FOR m : c.metodos»
                «IF m.name.equals("loadItems")»
                    this.«m.name»();
                «ENDIF»
            «ENDFOR»
        });
    }
}
'''

```

Código fuente generado

```

@Component({
    selector: 'app-home',

```

```

templateUrl: 'home.page.html',
styleUrls: ['home.page.scss'],
})
export class HomePage {
items: Item[] = [];
newItem: Item = <Item>{};
@ViewChild('mylist', {static: true}) mylist: IonList;
constructor(private storageService: StorageService, private plt: Platform, private toastController: ToastController) {
this.plt.ready().then(()=>{
this.loadItems();
});
}
}

```

Cuadro 4-10. Definición de las transformaciones – de los artefactos “Componente”, “arreglos”, “interfaz”, “listado”, “lista”, “servicio”, “métodos”.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Se modelan métodos del componente. Tipos de datos: Listado, Interfaz	Se crea el método “addItem” el cual hace referencia al método del servicio con nombre “add”, además del mensaje que el usuario visualizara	Transforma el método “addItem” del componente	Método para guardar dato
Modelado en el lenguaje Xtext			
MetodoComponente: <pre> 'metodo' name=ID '{ ('MetodoServicio' ':' metodoServicio=[MetodoServicio])? ('mensaje' ':' mensaje=STRING)? }' ; Interfaz: 'interfaz' name=ID '{ (propiedades+=Propiedad)* }' ; </pre>			
Lenguaje específico de dominio (DSL)			
<pre> metodo add { tipoParametro : Item tipoRetornoPromesa : any } metodo addItem { MetodoServicio : add mensaje : "Producto agregado" } interfaz Item { </pre>			

```

id : number
almacen : string
producto: string
precio : number
modifica: number

```

```

}

```

Transformación de modelo a texto con Xtend

```

def CharSequence compileComponent(Componente c)''

```

```

/** Create */

```

```

«FOR m : c.metodos»

```

```

  «IF m.name.equals("addItem")»

```

```

    «m.name»() {

```

```

      «FOR l : c.listado»

```

```

        «IF l.name.equals("newItem")»

```

```

          «FOR p : c.interfaz.propiedades»

```

```

            «IF p.name.equals("modifica")»

```

```

              this.«l.name».«p.name» = Date.now();

```

```

            «ENDIF»

```

```

          «ENDFOR»

```

```

        «ENDIF»

```

```

      «ENDFOR»

```

```

    «FOR l : c.listado»

```

```

      «IF l.name.equals("newItem")»

```

```

        «FOR p : c.interfaz.propiedades»

```

```

          «IF p.name.equals("id")»

```

```

            this.«l.name».«p.name» = Date.now();

```

```

          «ENDIF»

```

```

        «ENDFOR»

```

```

      «ENDIF»

```

```

    «ENDFOR»

```

```

    «FOR l : c.listado»

```

```

      «IF l.name.equals("newItem")»

```

```

        «FOR sm : c.servicio.metodos»

```

```

          «IF sm.name.equals("add")»

```

```

            this.e.toLowerCase»Service.

```

```

            «l.name»).then(item =>{

```

```

              this.«l.name» = <<c.interfaz.name>>{};

```

```

            «ENDIF»

```

```

          «ENDFOR»

```

```

        «ENDIF»

```

```

      «ENDFOR»

```

```

        this.showToast('«m.mensaje»');

```

```

    «FOR me : c.metodos»

```

```

      «IF me.name.equals("loadItems")»

```

```

        this.«me.name»();

```

```

      );

```

```

    }

```

```

  «ENDIF»

```

```

«ENDFOR»

```

```

«ENDIF»

```

```

«ENDFOR»

```

```

''

```

Código fuente generado

```

/** Create */

```

```

addItem() {

```

```

    this.newItem.modifica = Date.now();

```

```

    this.newItem.id = Date.now();

```

```

    this.storageService.add(this.newItem).then(item =>{

```

```

this.newItem = <Item>{};
                this.showToast('Producto agregado');
this.loadItems();
                });
            }
    
```

Cuadro 4-11. Definición de las transformaciones – del artefacto “addItem”.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Se modelan métodos del componente. Tipos de datos: Listado, Interfaz	Se crea el método “loadItems” el cual hace referencia al método del servicio con nombre “getAll”, además del mensaje que el usuario visualizará	Transforma el método “loadItems” del componente	Método para consultar todos los datos
Modelado en el lenguaje Xtext			
MetodoComponente: <pre> 'metodo' name=ID '{ ('MetodoServicio' ':' metodoServicio=[MetodoServicio])? ('mensaje' ':' mensaje=STRING)? }' </pre>			
Lenguaje específico de dominio (DSL)			
<pre> ppPrecio { dominio { servicio storage { metodo getAll { tipoRetornoPromesa : Item } } } arquitectura { componente Home { metodo loadItems { MetodoServicio : getAll } } } } </pre>			
Transformación de modelo a texto con Xtend			
<pre> def CharSequence compileComponent(Componente c) /** Read */ «FOR m : c.metodos» «IF m.name.equals("loadItems")» «m.name»() { «FOR sm : c.servicio.metodos» «IF sm.name.equals("getAll")» this. «c.servicio.name.toLowerCase» } } } </pre>			

<pre> metodo updateItem { MetodoServicio : update } } } } </pre>
<p>Transformación de modelo a texto con Xtend</p>
<pre> def CharSequence compileComponent(Componente c) /** update */ «FOR m : c.metodos» «IF m.name.equals("updateItem")» «m.name»(item: «c.interfaz.name») { «FOR p : c.interfaz.propiedades» «IF p.name.equals("almacen")» item.«p.name» = `UPDATED: \${item.«p.name}`; «ENDIF» «IF p.name.equals("modifica")» item.«p.name» = Date.now(); «ENDIF» «ENDFOR» «FOR sm : c.servicio.metodos» «IF sm.name.equals("update")» this. «c.servicio.name.toLowerCase» Service.«sm.name» (item).then(item =>{ «ENDIF» }) «ENDIF» «IF m.name.equals("updateItem")» this.showToast(«m.mensaje»); this.mylist.closeSlidingItems(); «ENDIF» «FOR me : c.metodos» «IF me.name.equals("loadItems")» this.«me.name»(); «ENDIF» «ENDFOR» } «ENDIF» «ENDFOR» /** </pre>
<p>Código fuente generado</p>
<pre> /** update */ updateItem(item: Item) { item.almacen = `UPDATED: \${item.almacen}`; item.modifica = Date.now(); this.storageService.update(item).then(item =>{ this.showToast('Producto Actualizado!'); this.mylist.closeSlidingItems(); this.loadItems(); }); } </pre>

Cuadro 4-13. Definición de las transformaciones – del artefacto “updateItem”.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Se modelan métodos del componente. Tipos de datos: Listado, Interfaz	Se crea el método “deleteItem” el cual hace referencia al método del servicio con nombre “delete”, además del mensaje que el usuario visualizará	Transforma el método “deleteItem” del componente	Método para consultar todos los datos
Modelado en el lenguaje Xtext			
MetodoComponente: <pre> 'metodo' name=ID '{ ('MetodoServicio' ':' metodoServicio=[MetodoServicio])? ('mensaje' ':' mensaje=STRING)? }' </pre>			
Lenguaje específico de dominio (DSL)			
<pre> ppPrecio { dominio { servicio storage { metodo delete { tipoRetornoPromesa : Item } } } arquitectura { componente Home { metodo deleteItem { MetodoServicio : update } } } } </pre>			
Transformación de modelo a texto con Xtend			
<pre> def CharSequence compileComponent(Componente c)''' /** Delete */ «FOR m : c.metodos» «IF m.name.equals("deleteItem")» «m.name»(item: «c.interfaz.name»){ «FOR sm : c.servicio.metodos» «IF sm.name.equals("delete")» this. «c.servicio.name.toLowerCase» Service.«sm.name» (item.id).then(item =>{ «ENDIF» }) «ENDFOR» } «ENDIF» » </pre>			

```

        this.showToast(«m.mensaje»);
        this.mylist.closeSlidingItems();
        «ENDIF»
        «FOR me : c.metodos»
            «IF me.name.equals("loadItems")»
                this.«me.name»();
            «ENDIF»
        «ENDFOR»
    «ENDIF»
«ENDIF»
...

```

Código fuente generado

```

/** Delete */
deleteltem(item: Item){
    this.storageService.delete(item.id).then(item =>{
        this.showToast("Producto eliminado");
        this.mylist.closeSlidingItems();
        this.loadItems();
    });
}

```

Cuadro 4-14. Definición de las transformaciones – del artefacto “deleteltem” del componente.

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Modelo del Servicio: Componente, Arreglos, Interfaz, Listado, Lista, MetodoServicio.	Crea la estructura de dador “Item” del tipo Interfaz interfaz y el servicio con el nombre “storage”; además de la tecnología con el nombre “store” y la palabra clave “mis_items” para identificar la información en la base de datos.	Transforma los elementos del servicio: name, arreglos, interfaz, listado, lista, metodos	El Servicio (@Injectable) con el nombre “storage” y su respectiva clase.
Modelado en el lenguaje Xtext			
Servicio: <pre> 'servicio' name=ID '{ (metodos+=MetodoServicio)* }' </pre> ; enum TipoDato: <pre> numero='number' cadena='string' ninguna='any' </pre> ;			

```

Entidad:
    Interfaz
;
Interfaz:
    'interfaz' name=ID '{
        (propiedades+=Propiedad)*
    }'
;
Propiedad:
    name=ID ':' tipoDato=TipoDato
;
Tipos:
    tipoPrimitivo=TipoDato | tipoEstructurado=[Entidad]
;
LLave:
    name=ID
;
Arreglo:
    'arreglo' name=ID ('tipo' tipo=[Entidad])?
;
Listado:
    'listado' name=ID 'tipo' tipo=[Entidad]
;
TiposArregloListado:
    Arreglo | Listado
;
MetodoServicio:
    'metodo' name=ID '{
        ('tipoParametro' ':' tipoParametro=Tipos)?
        ('tipoRetornoPromesa' ':' tipoRetorno=Tipos)
    }'
;
Tecnologia:
    'tecnologia' name=ID '{
        'llave' ':' llave=LLave
    }'
;

```

Lenguaje específico de dominio (DSL)

```

appPrecio {
    dominio {
        servicio storage {

            metodo add {
                tipoParametro : Item
                tipoRetornoPromesa : any
            }

            metodo getAll {
                tipoRetornoPromesa : Item
            }

            metodo update {
                tipoParametro : Item
                tipoRetornoPromesa : any
            }

            metodo delete {
                tipoParametro : number
                tipoRetornoPromesa : Item
            }
        }
    }
}

```

```
    }
  }
  interfaz Item {
    id : number
    almacen : string
    producto: string
    precio : number
    modifica: number
  }
}
tecnologia store {
  llave : mis_items
  metodo set {
  }
  metodo get {
  }
}
}
```

Transformación de modelo a texto con Xtend

```
def CharSequence compileService(Componente c)""
  export interface «c.interfaz.name» {
    «FOR p : c.interfaz.propiedades»
    «IF p.name.equals("id")»
    «p.name»: number,
    «ENDIF»
    «IF p.name.equals("almacen")»
    «p.name»: string,
    «ENDIF»
    «IF p.name.equals("producto")»
    «p.name»: string,
    «ENDIF»
    «IF p.name.equals("precio")»
    «p.name»: number,
    «ENDIF»
    «IF p.name.equals("modifica")»
    «p.name»: number
    «ENDIF»
    «ENDFOR»
  }
  const ITEMS_KEY = '«c.tecnologia.llave.name»';
  @Injectable({
    providedIn: 'root'
  })
  export class «c.servicio.name.toFirstUpper»Service {
    constructor(private «c.servicio.name»: Storage) { }
  }
```

Código fuente generado

<pre> } } } </pre>
<p>Transformación de modelo a texto con Xtend</p> <pre> /** Create */ «FOR m : c.servicio.metodos» «IF m.name.equals("add")» «m.name»(item: «c.interfaz.name»): Promise<any> { return this.«c.servicio.name».get(ITEMS_KEY).then((items: «c.interfaz.name»[]) => { if (items) { items.push(item); return this.«c.servicio.name».set(ITEMS_KEY, items); }else { return this.«c.servicio.name».set(ITEMS_KEY, [item]); } }); } «ENDIF» «ENDFOR» </pre>
<p>Código fuente generado</p> <pre> /** Create */ add(item: Item): Promise<any> { return this.storage.get(ITEMS_KEY).then((items: Item[]) => { if (items) { items.push(item); return this.storage.set(ITEMS_KEY, items); }else { return this.storage.set(ITEMS_KEY, [item]); } }); } </pre>

Cuadro 4-16. . Definición de las transformaciones – del artefacto “add” del servicio

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Se modelan metodos del servicio. Tipos de datos: Listado, Interfaz	Se crea el metodo “getAll” del servicio “storage”	Transforma el metodo “getAll” del servicio	Metodo para consultar dato en la base de datos
Modelado en el lenguaje Xtext			
Tipos:			
<pre> tipoPrimitivo=TipoDato tipoEstructurado=[Entidad] ; </pre>			
MetodoServicio:			

<pre> 'metodo' name=ID '{ ('tipoParametro' ':' tipoParametro=Tipos)? ('tipoRetornoPromesa' ':' tipoRetorno=Tipos) }' ; Servicio: 'servicio' name=ID '{ (metodos+=MetodoServicio)* }' ; </pre>
<p>Lenguaje específico de dominio (DSL)</p> <pre> appPrecio { dominio { servicio storage { metodo getAll { tipoRetornoPromesa : Item } } } } </pre>
<p>Transformación de modelo a texto con Xtend</p> <pre> /** Read */ «FOR m : c.servicio.metodos» «IF m.name.equals("getAll")» «m.name»: Promise<«c.interfaz.name»[]> { return this.«c.servicio.name».get(ITEMS_KEY); } «ENDIF» «ENDFOR» </pre>
<p>Código fuente generado</p> <pre> /** Read */ getAll(): Promise<Item[]> { return this.storage.get(ITEMS_KEY); } </pre>

Cuadro 4-17. Definición de las transformaciones – del artefacto “getAll” del servicio

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Se modelan métodos del servicio. Tipos de datos: Listado, Interfaz	Se crea el método “update” del servicio “storage”	Transforma el método “update” del servicio	Método para actualizar dato en la base de datos
Modelado en el lenguaje Xtext			
Tipos:			
<pre> tipoPrimitivo=TipoDato tipoEstructurado=[Entidad] ; </pre>			
MetodoServicio:			
<pre> 'metodo' name=ID '{ </pre>			

<pre> ('tipoParametro' ':' tipoParametro=Tipos)? ('tipoRetornoPromesa' ':' tipoRetorno=Tipos) '}' ; Servicio: 'servicio' name=ID '{ (metodos+=MetodoServicio)* }' ; </pre>
<p>Lenguaje específico de dominio (DSL)</p> <pre> appPrecio { dominio { servicio storage { metodo update { tipoParametro : Item tipoRetornoPromesa : any } } } tecnologia store { llave : mis_items metodo set { } metodo get { } } } </pre>
<p>Transformación de modelo a texto con Xtend</p> <pre> /** Update */ «FOR m : c.servicio.metodos» «IF m.name.equals("update")» «m.name»(item: «c.interfaz.name»): Promise<any> { return this.«c.servicio.name».get(ITEMS_KEY).then((items: «c.interfaz.name»[]) => { if (!items items.length == 0) { return null; } «FOR a : c.arreglos» «IF a.name.equals("newItems")»let «a.name»: «c.interfaz.name»[] = []; for (let i of items) { if (i.id == item.id) { «a.name».push(item); } else { «a.name».push(i); } } } } </pre>

```

    }
    return this.«c.interfaz.name»
    .set(ITEMS_KEY, «a.name»);
    });
    }
    «ENDIF»
    «ENDFOR»
    «ENDIF»
    «ENDFOR»

```

Código fuente generado

```

/** Update */
update(item: Item): Promise<any> {
  return this.storage.get(ITEMS_KEY).then((items: Item[]) => {

    if(!items || items.length == 0) {
      return null;
    }
    let newItems: Item[] = [];

    for (let i of items) {
      if(i.id == item.id) {
        newItems.push(item);
      } else {
        newItems.push(i);
      }
    }

    return this.Item.set(ITEMS_KEY, newItems);

  });
}

```

Cuadro 4-18. Definición de las transformaciones – del artefacto “update” del servicio

Modelado en el lenguaje Xtext	Lenguaje específico de dominio (DSL)	Transformación de modelo a texto con Xtend	Código fuente generado
Se modelan metodos del servicio. Tipos de datos: Listado, Interfaz	Se crea el metodo “update” del servicio “storage	Transforma el metodo “delete” del servicio	Método para eliminar dato en la base de datos
Modelado en el lenguaje Xtext			
Tipos: <pre> tipoPrimitivo=TipoDato tipoEstructurado=[Entidad] ; MetodoServicio: 'metodo' name=ID '{ ('tipoParametro' ':' tipoParametro=Tipos)? ('tipoRetornoPromesa' ':' tipoRetorno=Tipos) }' ; Servicio: </pre>			

<pre>'servicio' name=ID '{ (metodos+=MetodoServicio)* }' ; </pre>
<h3>Lenguaje específico de dominio (DSL)</h3>
<pre>appPrecio { dominio { servicio storage { metodo delete { tipoParametro : number tipoRetornoPromesa : Item } } } interfaz Item { id : number almacen : string producto: string precio : number modifica: number } } arquitectura { componente Home { arreglo items tipo Item arreglo newItems tipo Item listado newItem tipo Item } } tecnologia store { llave : mis_items metodo set { } metodo get { } } }</pre>
<h3>Transformación de modelo a texto con Xtend</h3>
<pre>/** Delete */ «FOR m : c.servicio.metodos» «IF m.name.equals("delete")» «m.name»(id: number): Promise<«c.interfaz.name»> { return this.«c.servicio.name»</pre>

<pre> .get(ITEMS_KEY).then((items: «c.interfaz.name»[]) =>{ if (!items items.length == 0) { return null; } let toKepp: «c.interfaz.name»[] = []; for (let i of items) { «FOR p : c.interfaz.propiedades» «IF p.name.equals("id")»if (i.«p.name» != id) {«ENDIF» «ENDFOR» toKepp.push(i); } return this.«c.servicio.name».set(ITEMS_KEY, toKepp); }); } «ENDIF» «ENDFOR» </pre>
<p>Código fuente generado</p> <pre> /** Delete */ delete(id: number): Promise<Item> { return this.storage.get(ITEMS_KEY).then((items: Item[]) =>{ if (!items items.length == 0) { return null; } let toKepp: Item[] = []; for (let i of items) { if (i.id != id) { toKepp.push(i); } } return this.storage.set(ITEMS_KEY, toKepp); }); } </pre>

Cuadro 4-19. Definición de las transformaciones – del artefacto “delete” del servicio

4.8 Generación del Editor

Una vez especificada la gramática del lenguaje de modelado, esta misma contiene la sintaxis concreta del lenguaje, Xtext genera *plugins* para el entorno integrado de desarrollo Eclipse para generar un editor de texto junto a características como el color de texto, detención de errores, autocompletar palabras claves del lenguaje, etc. Ver Figura 4-20.

Con la creación del metamodelo a partir del lenguaje textual se puede verificar el modelamiento gráfico de la gramática. se creará el archivo que permitirá visualizar el metamodelo construido de forma gráfica. Ver Figura 4-21 y Figura 4-22. Con la prueba solo

se verifica que la gramática este bien construida, para que se genere el código específico para el sistema de información se crearan plantillas con Xtend.

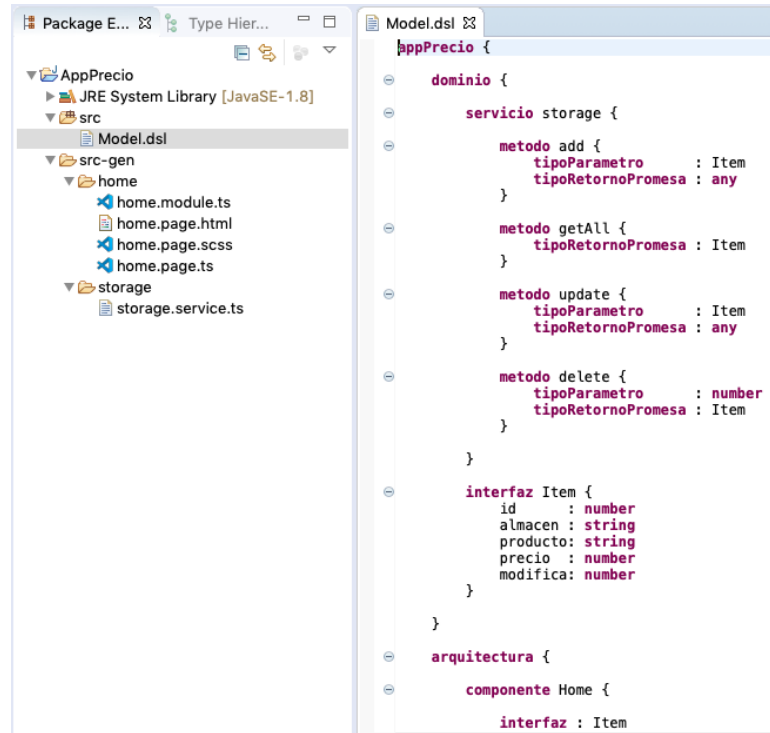


Figura 4-20. Pruebas del nuevo DSL

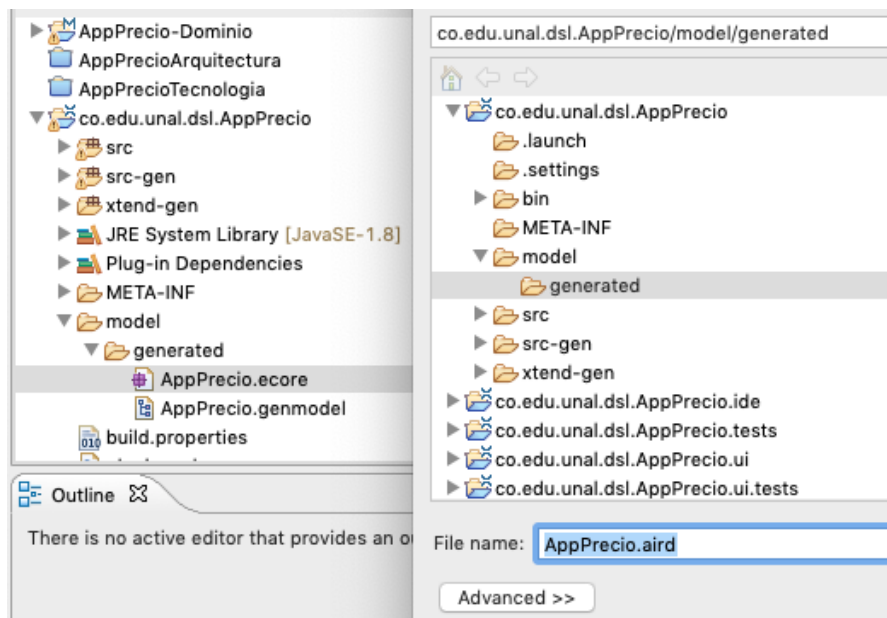


Figura 4-21. Asistente para ver el metamodelo gráfico

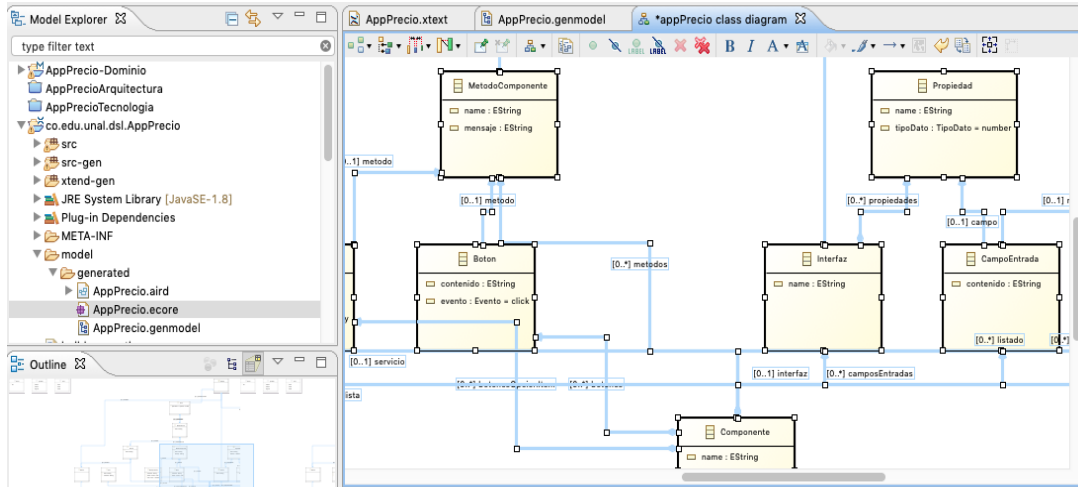


Figura 4-22. Prueba del metamodelo grafico a partir de la creación con texto

Para hacer las pruebas de la gramática desarrollada para la herramienta, primero se debe generar el flujo de trabajo.

El nuevo editor de texto permite crear proyecto Java, para la implementación del lenguaje del dominio específico; la extensión para los archivos que soportan este lenguaje es DSL. Ver Figura 4-20.

4.9 Plantillas de generación de código automática

Con la herramienta Xtend se puede implementar el DSL para la generación de código automático. Haciendo un análisis del código genérico y código esquemático repetitivo en la plantilla el código genérico se introduce como se encuentra en la implementación de referencia. El código esquemático repetitivo se agrega de manera dinámica con instrucciones entre en símbolo «» (angulares). Ver Anexo A y Anexo B.

Al construir el DSL y tomando la opción de guardar de Eclipse como resultado de la acción se generan los artefactos planteados de manera automática. Hay que tener en cuenta que cuando una de las sintaxis no corresponde al metamodelo planteado, se produce un error, lo cual no permite que se generen los artefactos.

5. Evaluación

5.1 Análisis conceptual

Un aspecto fundamental es que la herramienta cumpla con los objetivos y ventajas del desarrollo dirigido por modelo: Modelar a nivel de dominio (Arquitectura), mejorar la calidad del software, facilitar reusabilidad, portabilidad e incremento en la productividad.

La importancia de implementar la arquitectura dirigida por modelos - MDA (en inglés: Model-Driven Architecture) en el presente trabajo es presentar modelos para una evidente y sistemática separación de responsabilidades lo que permite que los modelos "Dominio", "Arquitectura" y "Tecnología" puedan ser mejorados por separado, además de permitir el acompañamiento en el ciclo de vida de la aplicación desde las especificaciones expresadas como modelos hasta el código fuente de la aplicación Web móvil.

La herramienta permite controlar los cambios tecnológicos gracias a la estructura del modelo planteado. La adaptación a los cambios en los requerimientos es muy sencilla para la herramienta, solo es desarrollar el modelo específico para ese nuevo requerimiento.

5.2 Prueba de concepto

Con los artefactos generados por la herramienta se realizan los criterios de evaluación seleccionando la construcción de una nueva aplicación dentro del dominio común.

Con el modelamiento se eleva el nivel de abstracción pensando en arquitectura, dominio y tecnología de software, lo que logró que la implementación de referencia permitiera desarrollar otra aplicación del mismo dominio de negocio, cumpliendo así con el incremento de la productividad.

Con la implementación del lenguaje específico de dominio se genera código en menos tiempo, además sus transformaciones permiten generar código específico sin la intervención del desarrollador de software.

5.2.1 Lenguaje específico de dominio

Con la discriminación de los componentes de la implementación de referencia, se permitió el diseño del metamodelo para crear el modelo específico del dominio con su lenguaje, logrando definir las especificaciones de la aplicación de referencia y las pruebas de concepto.

5.2.2 Transformaciones

Las transformaciones de modelo a modelo se realizaron de manera automática sin hacer ninguna intervención. En cuanto a la generación de código se logró implementar el lenguaje Xtend, reflejando las buenas prácticas propuestas por W3C, incluyendo artefactos de los marcos de trabajo Ionic y Angular en el lenguaje TypeScript.

5.2.3 Generación de código

Se alcanzó un nivel de código generado en una proporción suficiente, teniendo en cuenta el nivel de detalle en los artefactos generados por cada marco de trabajo, siguiendo la identificación del código genérico, código esquemático repetitivo y código individual.

5.3 Análisis práctico

En la generación de la aplicación se hace un análisis del número de líneas de código digitadas por el desarrollador, faltantes en la implementación de referencia frente al número de línea de código generada por la herramienta. El código digitado es mínimo en comparación al código generado. Ver el cuadro 5-1.

Artefacto	Líneas de código			Porcentaje (%) generado
	Manuales	Generadas	Diferencia	
Página, Ionic, Angular	55	50	5	90,9
Servicio TypeScript	81	73	18	90,1
Componente TypeScript	74	67	7	90,5
Modulo	25	23	2	92,0

Cuadro 5-1. Identificación de líneas de código fuente frente a las generadas de manera automática.

Como se puede ver en el cuadro 5-1, el porcentaje de código generado por la herramienta es evidentemente alto, en comparación con la generación de código manual, lo que reduce el esfuerzo en el desarrollo del software. La herramienta genera en promedio un 90,8% del código fuente para la construcción de la aplicación Web móvil.

El porcentaje de líneas de código fuente generado con la herramienta en el cuadro 5-1 la columna "Artefacto", específicamente "Página, Ionic, Angular" el 55 es la cantidad de líneas de código manuales en la implementación de referencia y es el 100% necesario para la solución al dominio del problema; la herramienta genera 50 líneas de código lo que equivale al 90,9% del código en la implementación de referencia.

La diferencia de líneas de código entre las manuales y la generadas de manera automática es aproximadamente el 10% en cada uno de los artefactos. Es el código necesario para ser adaptado al proyecto generado por el Framework Ionic y no generar errores en la ejecución de la aplicación.

6. Conclusiones y Trabajo futuro

Se presentó una herramienta basada en el desarrollo dirigido por modelos para la generación automática de aplicaciones web con base en una metodología general del desarrollo dirigido por modelos. La metodología propuesta se esquematizó en fases o actividades que decantaron en la herramienta, basada fundamentalmente en una implementación de referencia que contenía los elementos básicos y comunes de las aplicaciones web.

De este trabajo también se puede distinguir el estado del arte del desarrollo de aplicaciones web móviles, los lineamientos establecidos por la W3C, y las pautas generales y buenas prácticas existentes de dicho desarrollo.

Se resalta de manera fundamental, el resultado de la herramienta y la eficiencia en la generación de código fuente utilizable, de aproximadamente un 90,8 % sobre un desarrollo tradicional y manual, dejando claro las ventajas en el ahorro de tiempo para la construcción de aplicaciones web móviles.

Como actividades de trabajo futuro del presente documento, podemos señalar las siguientes:

- Extender el modelo para incluir la integración de Ionic con otros entornos de trabajo como React, Javascript, Vue, entre otros.
- Desarrollar el backend de la aplicación de referencia, para que la información pueda sincronizarse en la nube.
- Robustecer el modelo para incluir componentes de evaluación y prueba de los artefactos a generados

A. Anexo: Gramática del lenguaje Xtext

grammar co.edu.unal.dsl.AppPrecio **with** org.eclipse.xtext.common.Terminals

generate appPrecio "http://www.edu.co/unal/dsl/AppPrecio"

AppPrecio:

```
'appPrecio' '{  
    dominio=Dominio  
    arquitectura=Arquitectura  
    tecnologia=Tecnologia  
}';
```

Dominio:

```
{Dominio} 'dominio' '{  
    (servicios+=Servicio)*  
    (entidades+=Entidad)*  
}';
```

Arquitectura:

```
{Arquitectura} 'arquitectura' '{
```

```
(componentes+=Componente)*
```

```
};
```

Tecnologia:

```
'tecnologia' name=ID '{
```

```
  'llave' ':' llave=LLave
```

```
  (metodos+=MetodoTecnologia)*
```

```
};
```

Servicio:

```
'servicio' name=ID '{
```

```
  (metodos+=MetodoServicio)*
```

```
};
```

MetodoComponente:

```
'metodo' name=ID '{
```

```
  ('MetodoServicio' ':' metodoServicio=[MetodoServicio])?
```

```
  ('mensaje' ':' mensaje=STRING)?
```

```
};
```

MetodoServicio:

```
'metodo' name=ID '{
```

```
  ('tipoParametro' ':' tipoParametro=Tipos)?
```

```
  ('tipoRetornoPromesa' ':' tipoRetorno=Tipos)
```

```
};
```

MetodoTecnologia:

```
'metodo' name=ID '{  
    ('llave' ':' llave=[LLave])?  
    ('valor' ':' valor=[TiposArregloListado])?  
};
```

Entidad:

```
Interfaz;
```

Interfaz:

```
'interfaz' name=ID '{  
    (propiedades+=Propiedad)*  
};
```

Propiedad:

```
name=ID ':' tipoDato=TipoDato;
```

Componente:

```
'componente' name=ID '{  
    'interfaz' ':' interfaz=[Interfaz]  
    'servicio' ':' servicio=[Servicio]  
    'tecnologia' ':' tecnologia=[Tecnologia]  
    'titulo' ':' titulo=STRING
```

```

    (camposEntradas+=CampoEntrada)*
    (botones+=Boton)*
    (botonesOpcionItem+=BotonOpcionItem)*
    (metodos+=MetodoComponente)*
    (arreglos+=Arreglo)*
    (listado+=Listado)*
    lista=Lista?
};

```

CampoEntrada:

```

'campoEntrada' '{
    'contenido' ':' contenido=STRING
    'modelo' ':' modelo=[Listado]
    'campo' ':' campo=[Propiedad]
};

```

Boton:

```

'boton' '{
    'contenido' ':' contenido=STRING
    'evento' ':' evento=Evento
    ('metodo' ':' metodo=[MetodoComponente])?
};

```

BotonOpcionItem :

```
'botonBotonOpcionItem' '{  
    'contenido' ':' contenido=STRING  
    'evento' ':' evento=Evento  
    ('color' ':' color=Color)?  
    ('metodo' ':' metodo=[MetodoComponente])?  
};
```

Lista:

```
'lista' name=ID '{  
    'contenido' ':' contenido=STRING  
};
```

Tipos:

```
tipoPrimitivo=TipoDato | tipoEstructurado=[Entidad];
```

LLave:

```
name=ID;
```

Arreglo:

```
'arreglo' name=ID ('tipo' tipo=[Entidad]);
```

Listado:

```
'listado' name=ID 'tipo' tipo=[Entidad];
```

TiposArregloListado:

Arreglo | Listado;

enum Evento:

click='click';

enum TipoDato:

numero='number' | cadena='string' | ninguna='any';

enum Color:

color1='recondary' | color2='danger' | color3='primary';

B. Anexo: Código fuente del lenguaje específico de dominio generado.

```
appPrecio {
  dominio {
    servicio storage {
      metodo add {
        tipoParametro : Item
        tipoRetornoPromesa : any
      }
      metodo getAll {
        tipoRetornoPromesa : Item
      }
      metodo update {
        tipoParametro : Item
        tipoRetornoPromesa : any
      }
      metodo delete {
        tipoParametro : number
        tipoRetornoPromesa : Item
      }
    }
  }
}
```

```
interfaz Item {
    id : number
    almacen : string
    producto: string
    precio : number
    modifica: number
}
}

arquitectura {
    componente Home {
        interfaz : Item
        servicio : storage
        tecnologia : store
        titulo : "Productos de la canasta familiar"
        campoEntrada {
            contenido : "Almacen"
            modelo : newItem
            campo : almacen
        }
        campoEntrada {
            contenido : "Producto"
            modelo : newItem
            campo : producto
        }
        campoEntrada {
```

```
        contenido : "Precio"

        modelo   : newItem

        campo    : precio
    }

    boton {

        contenido : "Agregar"

        evento   : click

        metodo   : addItem

    }

    botonBotonOpcionItem {

        contenido : "Actualizar"

        evento   : click

        color    : recondary

        metodo   : updateItem

    }

    botonBotonOpcionItem {

        contenido : "Eliminar"

        evento   : click

        color    : danger

        metodo   : deleteItem

    }

    metodo addItem {

        MetodoServicio : add

        mensaje : "Producto agregado"
```

```
    }
    metodo loadItems {
        MetodoServicio : getAll
    }
    metodo updateItem {
        MetodoServicio : update
        mensaje : "Producto Actualizado!"
    }

    metodo deleteItem {
        MetodoServicio : delete
        mensaje : "Producto eliminado"
    }

    arreglo items tipo Item
    arreglo newItems tipo Item
    listado newItem tipo Item
    lista myList {
        contenido : "Productos registrados"
    }
}

tecnologia store {
    llave : mis_items
    metodo set {}
    metodo get {}
}
}
```


Bibliografía

- [1] Gansemer, S.; Groner, U.; Maus, M.; "Database Classification of Mobile Devices," Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007. 4th IEEE Workshop on, vol., no., pp.699-703, 6-8 Sept. 2007. doi: 10.1109/IDAACS.2007.4488513
- [2] Lettner, M.; Tschernuth, M.; "Applied MDA for Embedded Devices: Software Design and Code Generation for a Low-Cost Mobile Phone," Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual, vol., no., pp.63-68, 19-23 July 2010. doi: 10.1109/COMPSACW.2010.21
- [3] Charaf, H.; "Developing Mobile Applications for Multiple Platforms," Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference on the, vol., no., pp.2, 5-6 Sept. 2011. doi: 10.1109/ECBS-EERC.2011.43
- [4] Kafaie, S.; Kashefi, O.; Sharifi, M.; "Augmented Mobile Devices through Cyber Foraging," Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on, vol., no., pp.145-152, 6-8 July 2011. doi: 10.1109/ISPDC.2011.30
- [5] L, E. D. L., G, M. G., S, M. L., & R, E. L. I. (n.d.). "Proceso de Desarrollo de Software Mediante Herramientas MDA".
- [6] Enrique, L. & Colso, C. D. (), 'Arquitectura dirigida por modelos para J2ME 1 Abstract 2 Keywords', 1--33.
- [7] OMG Background information, Disponible en-línea: <http://www.omg.org/news/about>.
- [8] Object Management Group. Model Driven Architecture Guide, 2003.
- [9] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
- [10] Mu, J. (n.d.). MDA a Debate.
- [11] Selic, B.; "Model-driven development: its essence and opportunities," Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on, vol., no., pp.7 pp., 24-26 April 2006. doi: 10.1109/ISORC.2006.54

- [12] Quintero, R.; Pelechano, V.; Fons, J. & Pastor, O. (), "Aplicación de MDA al Desarrollo de Aplicaciones Web en OOWS", 1--12.
- [13] Bernardo, J. & Anaya, R. (), "Marco de Referencia para la Evaluación de Herramientas Basadas en MDA", (c), 1--14.
- [14] Sun Developer Network: Java Metadata Interface (JMI) [documento en línea]. SUN (2002) [citado 10-may-2006].
- [15] Ruiz, F. & Piattini, M. (2007), "Model Driven Engineering Aplicado a Business Process Management".
- [16] Bézivin, J., MDA: From Hype to Hope, and Reality, Conferenciante invitado en UML'03 (2003).
- [17] Kennedy, A., Carter, K., Frank, W., & Architects, D. (2003). MDA Guide Version 1 .0, (May).
- [18] Bernardo, J. & Anaya, R. (), 'Marco de Referencia para la Evaluación de Herramientas Basadas en MDA', (c), 1--14.
- [19] Object Management Group: Revised submission for MOF 2.0 QVT rfp [documento en línea]. OMG (2002) [citado 22-ago-2006].
- [20] [Sun Developer Network: Java Metadata Interface (JMI) [documento en línea]. SUN (2002) [citado 10-may-2006].
- [21] Object Management Group. UML Specification (Action Semantics) [documento en línea]. OMG (2002) [citado 22-ago-2006].
- [22] Willink, E.: UMLX - A graphical transformation language for MDA. En: OOPSLA 2003 Conference. Anaheim, California (2003).
- [23] Generative Model Transformer. VIATRA2 Subproject [documento en línea]. GMT (2005) [citado 22-ago-2006] URL:
<http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/index.html>
- [24] Agrawal, A., Kalmar, Z., Karsai, G., Shi, F., Vizhanyo, A.: GReAT User Manual. Nashville: Institute for Software-Integrated Systems, Vanderbilt University (2003).
- [25] International Business Machines Corp.: Rational Rose XDE Modeler [documento en línea]. IBM (2006) [citado 23-ago-2006] es_ES/products/W107428N46756Z97.html
- [26] [20] Program-Transformation.Org. Stratego: Strategies for Program Transformation [documento en línea]. Program-Transformation (2004) [citado 23-ago-2006]
- [27] Marschall, F., Braun, P.: BOTL - The Bidirectional Object Oriented Transformation Language. Instituto de Informática, Universidad Técnica de Munich. Munich (2003).

- [28] Interactive Objects: ArcStyler 5.5. Documentation Roadmap [documento en línea]. IO (2006) [citado 23-ago-2006]
- [29] Codagen Technologies Corp.: Codagen Architect 3.0: Reviewer's Guide [documento en línea]. Codagen (2002) [citado 23-ago-2006]
- [30] The Modelling, Simulation And Design Lab. ATOM3: A Tool for Multi-formalism Meta-Modelling [documento en línea]. MSDL (2006) [citado 24-ago-2006]
- [31] Akehurst, D.H., Howells, W.G., McDonald-Maier K.D.: Kent Model Transformation Language. En: MoDELS 2005 Conference. Montego Bay, Jamaica (2005)
- [32] Charaf, H.; "Developing Mobile Applications for Multiple Platforms," Engineering of Computer Based Systems (ECBS-EERC), 2011 2nd Eastern European Regional Conference on the, vol., no., pp.2, 5-6 Sept. 2011
doi: 10.1109/ECBS-EERC.2011.43
- [33] Forstner, B.; Lengyel, L.; Levendovszky, T.; Mezei, G.; Kelenyi, I.; Charaf, H.; "Model-based system development for embedded mobile platforms," Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on, vol., no., pp.10 pp.-52, 30-30 March 2006
doi: 10.1109/MBD-MOMPES.2006.20
- [34] Choi, Y.; Yang, J.-S.; Jeong, J.; "Application framework for multi platform mobile application software development," Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on, vol.01, no., pp.208-213, 15-18 Feb. 2009
- [35] Jones, V.; Rensink, A.; Brinksma, E.; "Modelling mobile health systems: an application of augmented MDA for the extended healthcare enterprise," EDOC Enterprise Computing Conference, 2005 Ninth IEEE International, vol., no., pp. 58- 69, 19-23 Sept. 2005
doi: 10.1109/EDOC.2005.22
- [36] Khalifa, M.; Verner, J.M.; "Drivers for software development method usage," Engineering Management, IEEE Transactions on, vol.47, no.3, pp.360-369, Aug 2000
doi: 10.1109/17.865904
- [37] Jong-Won Ko; Sung-Ho Sim; Young-Jae Song; "Test Based Model Transformation Framework for Mobile Application," Information Science and Applications (ICISA), 2011 International Conference on, vol., no., pp.1-7, 26-29 April 2011
doi: 10.1109/ICISA.2011.5772373
- [38] Lettner, M.; Tschernuth, M.; "Applied MDA for Embedded Devices: Software Design and Code Generation for a Low-Cost Mobile Phone," Computer Software and Applications

Conference Workshops (COMPSACW), 2010 IEEE 34th Annual, vol., no., pp.63-68, 19-23 July 2010

doi: 10.1109/COMPSACW.2010.21

[39] Minovic, M.; Milovanovic, M.; Jovanovic, M.; Starcevic, D.; "Model driven development of user interfaces for educational games," Human System Interactions, 2009. HSI '09. 2nd Conference on, vol., no., pp.611-617, 21-23 May 2009

doi: 10.1109/HSI.2009.5091048

[40] Stoyanov, S.; Ganchev, I.; Popchev, I.; O'Droma, M.; "Service-oriented and agent-based approach for the development of InfoStation eLearning intelligent system architectures," Intelligent Systems, 2008. IS '08. 4th International IEEE Conference, vol.1, no., pp.6-20-6-25, 6-8 Sept. 2008. doi: 10.1109/IS.2008.4670434

[41] Kun Yang; Henning, I.; Shumao Ou; Azmoodeh, M.; "Model-based service discovery for next-generation mobile systems," Communications Magazine, IEEE, vol.44, no.9, pp.122-129, Sept. 2006. doi: 10.1109/MCOM.2006.1705988

[42] WebMovil @ www.w3c.es. (n.d.).

[43] Pastor O. et al An Object-Oriented Approach to Automate Web Applications Development. K. Bauknecht, S.K. Madria, G. Pernul (Eds.) EC-Web 2001, LNCS 2115, pp.16-28, 2001. Springer Verlag Berlin Heidelberg 2001.

[44] Visser, E. WebDSL: A Case Study in Domain-Specific Language Engineering 2008.

[45] WebDSL.org. (n.d.). Retrieved January 11, 2013, from <http://webdsl.org>

[46] EMFText. (n.d.). Retrieved January 11, 2013, from <http://www.emftext.org>

[47] Frank Budinsky. Eclipse Modelling Framework: Developer's Guide. Addison Wesley, 2003.

[48] World Wide Web Consortium (W3C). (n.d.). Retrieved January 11, 2013, from <http://www.w3.org/>

[49] Escobar, K. R. (2011). Un acercamiento al Desarrollo Dirigido por Modelos An approach to Model Driven Development, 4(9).