

# Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Código (31105151)

Trabajo Fin de Máster en Ingeniería de Software y  
Sistemas Informáticos

## Gen Móvil: Conversor entre XAML y XML Android mediante DSL

Alumno: Mario Sánchez Pérez

Director: Ismael Abad Cardiel

Curso: 2018/2019

Convocatoria: septiembre 2019



# Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Código (31105151)

Trabajo Fin de Máster en Ingeniería de Software y  
Sistemas Informáticos

## Gen Móvil: Conversor entre XAML y XML Android mediante DSL

Alumno: Mario Sánchez Pérez

Director: Ismael Abad Cardiel



**DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO,  
PARA LA DEFENSA DEL TRABAJO FIN DE MÁSTER**

Fecha: 19/09/2019

Quién suscribe:

Autor(a): Mario Sánchez Pérez D.N.I/N.I.E/Pasaporte.: 52013166T
--

Hace constar que es el autor del trabajo:

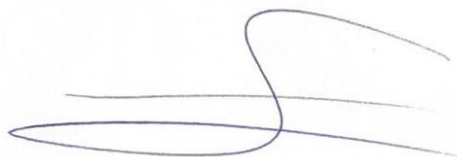
Gen Móvil: Conversor entre XAML y XML Android mediante DSL
--

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores se han referenciado debidamente en el texto de dicho trabajo.

**DECLARACIÓN:**

- Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.


Fdo.



## **Autorización**

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es:

A handwritten signature in blue ink, consisting of a large, stylized 'S' shape with a horizontal line crossing it, and a long horizontal stroke extending to the right.



## **Resumen**

El presente trabajo describe la utilidad, la viabilidad y la estrategia de implementación de un prototipo de generador automático de código XAML, para interfaces Windows, y XML Android, para interfaces de aplicaciones Android.

La estrategia de generación de código se apoya en el uso de un lenguaje específico de dominio (DSL). Este lenguaje ha sido creado para el prototipo y se ha definido basándose en lenguaje de marcado XML.

## **Palabras clave**

DSL, generación, código, XAML, XML, Android, GUI, interfaz





# Índice

1	Introducción.....	1
2	Objetivos.....	2
2.1	Requisitos funcionales .....	4
2.2	Requisitos no funcionales .....	4
3	Estado del arte .....	5
3.1	Generación de código de interfaz de usuario a partir de código DSL .....	6
3.2	Definición de interfaces .....	7
3.3	Convertidores de código.....	8
3.4	Frameworks de desarrollo de aplicaciones multiplataformas.....	9
3.5	Resultados de la investigación .....	11
4	Desarrollo de la solución .....	13
4.1	Diagrama de caso de usos .....	13
4.2	Tecnología y recursos utilizados.....	15
4.3	Lectura de ficheros y reconocimiento de elementos y atributos.....	15
4.3.1	Lectura de Ficheros XAML y XML Android .....	16
4.3.2	Lectura de Fichero DSL .....	19
4.4	Análisis de XAML y XML Android.....	21
4.4.1	Análisis para la generación de DSL .....	21
4.4.1.1	Traducción directa de nombre de elemento y atributos.....	22
4.4.1.2	Traducción de conjunto de atributo y valor.....	22
4.4.1.3	Casos no parametrizables .....	23
4.4.2	Análisis para la generación de código final .....	24
4.4.2.1	Traducción directa de nombre de elemento y atributos.....	24
4.4.2.2	Traducción de conjunto de atributo y valor.....	25
4.4.2.3	Casos no parametrizables .....	25
4.5	Configuración de la traducción .....	26
4.5.1	Procesos parametrizables .....	26
4.5.1.1	Especificación de generación de DSL a partir de código .....	28
4.5.1.1.1	Traducción directa de nombre de elemento y atributos .....	28
4.5.1.1.2	Traducción de conjunto de atributo y valor .....	29
4.5.1.2	Especificación de generación de código final a partir de DSL.....	30
4.5.1.2.1	Traducción directa de nombre de elemento y atributos .....	30
4.5.1.2.2	Traducción de conjunto de atributo y valor .....	31

4.5.2.	Especificación de DSL.....	32
4.5.2.1	Traducción de elementos .....	32
4.5.2.1.1	Text .....	32
4.5.2.1.2	Button .....	34
4.5.2.1.3	Cajas de texto.....	35
4.5.2.1.4	CheckBox.....	36
4.5.2.1.5	Select.....	37
4.5.2.1.6	RadioButton .....	39
4.5.2.2	Traducción de atributo.....	40
4.5.2.2.1	Traducción de atributos manteniendo valor.....	40
4.5.2.2.2	Traducción de atributos y valor.....	41
4.4.3	Procesos no parametrizables .....	43
4.6	Traducción de proyectos .....	44
4.6.1	Traducción de Windows a Android .....	44
4.6.2	Traducción de Android a Windows .....	44
5	Caso Práctico .....	45
5.1	Generación de páginas .....	45
5.1.1	Página de Login.....	45
5.1.1.1	Generación a partir de DSL.....	46
5.1.1.2	Generación como traducción .....	47
5.1.1.2.1	Traducción XML Android a XAML .....	48
5.1.1.2.1	Traducción XAML a XML Android .....	49
5.1.2	Página de Registro.....	50
5.1.2.1	Generación a partir de DSL.....	50
5.1.2.2	Generación como traducción .....	51
5.1.2.2.1	Traducción XML Android a XAML .....	51
5.1.1.2.1	Traducción XAML a XML Android .....	52
5.2	Traducción de proyectos .....	53
5.2.1	Traducción de Window a Android .....	53
5.2.2	Traducción de Android a Window .....	55
6	Conclusiones.....	58
7	Líneas futuras .....	60
	Referencias bibliográficas .....	61
	Siglas .....	63
	Anexos.....	64

Manual de usuario.....	64
Uso del prototipo para generación de páginas a partir de DSL.....	65
Mediante consola de comandos.....	65
Mediante uso como librería.....	65
Ejemplos.....	66
Generación de ficheros XAML.....	66
Generación de ficheros XAML y XML Android.....	67
Uso del prototipo para traducción de páginas.....	67
Mediante consola de comandos.....	68
Mediante uso como librería.....	68
Ejemplos.....	68
Traducir un fichero XAML a XML Android.....	69
Traducir un fichero XML Android a XAML.....	69
Uso del prototipo para traducción de proyectos.....	70
Mediante consola de comandos.....	70
Mediante uso como librería.....	71
Ejemplos.....	71
Traducir un proyecto de plataforma Window a Android.....	71
Traducir un proyecto de plataforma Android a Window.....	72
Templates.....	74
templateXAMLVisual.template.....	74
templateAndroidVisual.template.....	74
Ejemplos de generación de páginas.....	75
Generación de página Login.....	75
Entrada en lenguaje DSL.....	75
Generación a partir de DSL.....	76
Fichero XAML de página de Login desde DSL.....	76
Fichero XML Android de página de Login desde DSL.....	77
Generación a partir de traducción.....	78
Traducción XML Android a XAML.....	78
Traducción XAML a XML Android.....	79
Generación de página Registro.....	80
Entrada en lenguaje DSL.....	80
Generación a partir de DSL.....	81
Fichero XAML de página de Registro desde DSL.....	81

Fichero XML Android de página de Registro desde DSL.....	83
Generación a partir de traducción.....	85
Traducción XML Android a XAML .....	85
Traducción XAML a XML Android.....	87

## Lista de Figuras

Figura 1 Aplicación del prototipo en un proceso de desarrollo de una aplicación multiplataforma .....	3
Figura 2 Identificación de requisitos .....	11
Figura 3 Análisis de requisitos sobre herramientas.....	11
Figura 4 Diagrama de caso de uso del prototipo .....	13
Figura 5 Diagrama de principales procesos del prototipo .....	14
Figura 6 Diagrama de flujo de datos de la aplicación .....	15
Figura 7 Diagrama de Clases: lectura ficheros plataformas .....	16
Figura 8 Diagrama de flujo de lectura de ficheros XAML o XML Android .....	18
Figura 9 Diagrama de secuencia de lectura de ficheros XAML o XML Android .....	19
Figura 10 Flujo de lectura de árbol de Nodos DSL.....	20
Figura 11 Diagrama de flujo de procesos de generación de DSL .....	21
Figura 12 Ejemplo traducciones directas en la generación de DSL .....	22
Figura 13 Ejemplo traducción atributo y valor para atributo “checked” en DSL .....	22
Figura 14 Ejemplo traducción atributo y valor para atributo “android: layout_gravity”	23
Figura 15 Ejemplo traducción por extensión para el atributo “margin” .....	23
Figura 16 Ejemplo de traducción de atributo texto .....	24
Figura 17 Ejemplo traducción atributo y valor para atributo “android: layout_gravity” mediante DSL.....	27
Figura 18 Ejemplo de traducción de atributo texto mediante DSL .....	27
Figura 19 Formato para traducción directa de nombre de elemento y atributos en la generación de DSL .....	28
Figura 20 Ejemplo para traducción directa en la generación de DSL desde Android XML .....	28
Figura 21 Ejemplo para traducción directa en la generación de DSL desde XAML.....	28
Figura 22 Formato para traducción de conjunto de atributo y valor en la generación de DSL.....	29
Figura 23 Ejemplo para traducción conjunta para atributo isChecked desde XAML....	29
Figura 24 Ejemplo para traducción conjunta para atributo isChecked desde XML Android.....	29
Figura 25 Ejemplo para traducción conjunta para atributo y valor android: layout_gravity ="bottom" desde XML Android.....	29
Figura 26 Ejemplo para traducción conjunta para atributo y valor android: layout_gravity y valor por defecto desde XML Android .....	30
Figura 27 Formato para traducción directa de nombre de elemento y atributos en la generación de código final.....	30

Figura 28 Formato para traducción directa de atributo en la generación de código final cuando el atributo puede ser traducido distinto en función del elemento que lo contiene .....	30
Figura 29 Ejemplo de especificación de traducción de atributo en la generación de código final cuando el atributo puede ser traducido distinto en función del elemento que lo contiene .....	30
Figura 30 Formato para traducción de conjunto de atributo y valor en la generación de código final .....	31
Figura 31 Ejemplo para traducción conjunta para atributo isChecked desde DSL.....	31
Figura 32 Ejemplo para traducción conjunta para atributo android:checked desde DSL .....	31
Figura 33 Ejemplo para traducción conjunta para generar el atributo XML Android android:layout_gravity y valor “button” .....	31
Figura 34 Elemento <text> en los distintos lenguajes.....	32
Figura 35 Imagen <TextBlock> en XAML.....	33
Figura 36 Imagen <TextView> en XML Android .....	33
Figura 37 Traducción de elemento <text> y sus atributos .....	33
Figura 38 Elemento <button> en los distintos lenguajes.....	34
Figura 39 Imagen <Button> en XAML.....	34
Figura 40 Imagen <Button> en XML Android .....	34
Figura 41 Traducción de elemento <button> y sus atributos .....	35
Figura 42 Elemento <textBox> en los distintos lenguajes .....	35
Figura 43 Imagen <TextBox> en XAML.....	35
Figura 44 Imagen <EditText> en XML Android .....	35
Figura 45 Traducción de elemento <textBox> y sus atributos.....	36
Figura 46 Elemento <checkBox> en los distintos lenguajes.....	36
Figura 47 Imagen <CheckBox> en XAML.....	36
Figura 48 Imagen <CheckBox> en XML Android .....	36
Figura 49 Traducción de elemento <checkBox> y sus atributos .....	37
Figura 50 Elemento <select> en los distintos lenguajes.....	37
Figura 51 Imagen <ComboBox> en XAML .....	38
Figura 52 Imagen <Spinner> en XML Android.....	38
Figura 53 Traducción de elemento <select> y sus atributos .....	38
Figura 54 Elemento <radioButton> en los distintos.....	39
Figura 55 Imagen <RadioButton> en XAML .....	39
Figura 56 Imagen <RadioButton> en XML Android.....	39
Figura 57 Traducción de elemento <radioButton> y sus atributos .....	40

Figura 58 Traducción de atributo “width” .....	40
Figura 59 Relación de traducción de atributos .....	40
Figura 60 Relación de traducción de atributos y valores.....	42
Figura 61 Ejemplo de atraducción por defecto para “android:gravity” .....	43
Figura 62 Diagrama de ejemplo de extensión de funcionalidad .....	42
Figura 63 Boceto de página Login .....	46
Figura 64 Imagen con fichero XML Android y XAML generado a partir de DSL .....	46
Figura 65 Páginas de Login generadas a partir de DSL .....	47
Figura 66 Página de Login XAML generada como traducción de XML Android .....	48
Figura 67 Página de Login XML Android generada como traducción de XAML.....	49
Figura 68 Boceto de página Registro .....	50
Figura 69 Páginas de Registro generadas a partir de DSL .....	51
Figura 70 Página de Registro XAML generada como traducción de XML Android.....	52
Figura 71 Página de Registro XML Android generada como traducción de XAML.....	52
Figura 72 Traducción de proyecto Window a Android.....	54
Figura 73 Imagen de ficheros origen en la traducción de XML Android a XAML.....	55
Figura 74 Imagen de ficheros generados en la traducción de XML Android a XAML.	55
Figura 75 Traducción de proyecto Android a Window .....	56
Figura 76 Imagen de ficheros origen en la traducción de XAML a XML Android.....	57
Figura 77 Imagen de ficheros generados en la traducción de XAML a XML Android.	57
Figura 78 Imagen con el menú mostrado en consola de comandos .....	64
Figura 79 Ejemplo de generación XAML mediante consola de comandos .....	66
Figura 80 Ejemplo de generación XAML mediante uso del prototipo como librería externa .....	66
Figura 81 Ejemplo de generación XAML y XML Android mediante consola de comandos. Opciones numéricas .....	67
Figura 82 Ejemplo de generación XAML y XML Android mediante consola de comandos. Opciones textuales.....	67
Figura 83 Ejemplo de generación XAML y XML Android mediante uso del prototipo como librería externa .....	67
Figura 84 Ejemplo de traducción XAML a XML Android mediante consola de comandos. Opciones numéricas .....	69
Figura 85 Ejemplo de traducción XAML a XML Android mediante consola de comandos. Opciones textuales.....	69
Figura 86 Ejemplo de traducción XAML a XML Android mediante uso del prototipo como librería externa .....	69



Figura 87 Ejemplo de traducción XML Android a XAML mediante consola de comandos. Opciones numéricas .....	70
Figura 88 Ejemplo de traducción XML Android a XAML mediante consola de comandos. Opciones textuales.....	70
Figura 89 Ejemplo de traducción XML Android a XAML mediante uso del prototipo como librería externa .....	70
Figura 90 Ejemplo de traducción de proyecto Window a Android mediante consola de comandos. Opciones numéricas .....	72
Figura 91 Ejemplo de traducción de proyecto Window a Android mediante consola de comandos. Opciones textuales.....	72
Figura 92 Ejemplo de traducción de proyecto Window a Android mediante uso del prototipo como librería externa .....	72
Figura 93 Ejemplo de traducción de proyecto Android a Window mediante consola de comandos. Opciones numéricas .....	72
Figura 94 Ejemplo de traducción de proyecto Android a Window mediante consola de comandos. Opciones textuales.....	72
Figura 95 Ejemplo de traducción de proyecto Android a Window mediante uso del prototipo como librería externa .....	73

# 1 Introducción

El proyecto desarrollado y descrito en el presente trabajo intenta servir de apoyo para hacer frente a un problema actual de alcance en el desarrollo de cualquier aplicación móvil. Se trata de la diversidad de plataformas y sistemas operativos. En el mercado actual, la globalización y el enfrentamiento entre las distintas compañías tecnológicas han ocasionado que los usuarios de aplicaciones móviles se encuentren repartidos en varios sistemas operativos. Esto puede llegar a ocasionar que una empresa pierda parte del mercado al orientar el desarrollo de sus aplicaciones a un único sistema operativo.

La adaptación al mercado multiplataforma depende de la integración y robustez del enfoque multiplataforma en los procesos de desarrollo. Una mala orientación o el uso de una tecnología poco acertada en este ámbito pueden ocasionar que se gaste un exceso de recursos de forma inapropiada. Frente a esto, hay varios enfoques posibles a aplicar en los procesos de desarrollo. Por un lado, desde la concepción de un proyecto se puede marcar como objetivo que la aplicación a desarrollar pueda ser ejecutada y usada en distintas plataformas. Por otro lado, una vez finalizada la aplicación, se puede realizar un proyecto de migración a otra plataforma distinta para la cual fue concebida. Es necesario aplicar el enfoque correcto teniendo en cuenta la disponibilidad de recursos y cómo encaja mejor con el objetivo deseado.

El prototipo de generador de código desarrollado ha sido ideado para poder ser incorporado en distintas etapas en el desarrollo de una aplicación multiplataforma móvil y particularmente en el ámbito de la definición y desarrollo de interfaces de usuario. Por un lado, permite generar código final a partir de DSL para distintas plataformas. De esta forma, puede ser integrado en el desarrollo de interfaces de usuario desde el inicio del desarrollo de la aplicación y permitiendo crear aplicaciones multiplataforma. Su otra funcionalidad principal es la capacidad de aplicar ingeniería inversa y traducir código entre distintas plataformas. Esta última característica permite realizar la migración de una aplicación ya finalizada a otra plataforma.

## 2 Objetivos

En el presente apartado, se va a detallar las funcionalidades y características deseadas con las que debería contar una herramienta para cubrir las distintas necesidades en el proceso de desarrollo de una aplicación móvil multiplataforma. La generación automática de código puede cubrir distintos campos dentro del conjunto de código necesario a desarrollar para el correcto funcionamiento de una aplicación. Por ejemplo, se puede aplicar a distintos dominios [1] como la generación de documentación, de tests unitarios, de código SQL, de código de acceso a bases de datos o de capas de negocio. Todos estos ejemplos son procesos que, al ser analizados, permiten encontrar patrones de comportamiento para los que generar herramientas de generación automática de código. Identificando estos patrones y desarrollando procesos que los repliquen se puede lograr que, a partir de una entrada de datos que cumple unas normas, se puedan generar código final reduciendo los tiempos y costes de desarrollo.

Debido al amplio abanico de dominios a los que aplicar la generación de código en el desarrollo de una aplicación, se va a proceder a delimitar la investigación a realizar al ámbito de las interfaces de usuario de aplicaciones móviles y los procesos de desarrollo de aplicaciones móviles multiplataforma. Esta investigación se va a centrar en las posibilidades actuales para la realización de dos procesos que pueden llegar a ser muy productivos en el desarrollo de una aplicación multiplataforma: la generación automática de código para distintas plataformas partiendo de una misma entrada y la traducción de ficheros entre distintas plataformas.

Un punto importante es identificar el tipo de aplicaciones multiplataforma a generar. Esta tipología se define en base al tipo de fichero generado, si se trata de un fichero nativo de la plataforma o no. Estos tipos de aplicaciones pueden ser generalizados cómo:

- Aplicación web: la aplicación desarrollada reside en un servidor y es accedida y utilizada por el usuario mediante un navegador web de su dispositivo móvil. El desarrollo tiene un alto grado de independencia de la plataforma.
- Aplicación nativa: se trata de una aplicación que debe ser instalada en el dispositivo móvil del usuario. Por lo tanto, el desarrollo tiene un alto grado de dependencia de la plataforma.
- Aplicación híbrida: se trata de una aplicación que mezcla los dos conceptos anteriores. Mediante una aplicación nativa, se accede a una aplicación web con el uso de componente o contenedor.

El tipo de aplicación a tratar en la investigación de este trabajo serán las nativas. Esta decisión se debe a que el desarrollo multiplataforma de este tipo de aplicaciones es más costoso y el objetivo de la investigación a realizar es facilitar este tipo de proceso de desarrollo. Esta dificultad se encuentra en que el desarrollo de una aplicación nativa se ha de realizar con la tecnología y tipos de ficheros propios de dicha plataforma y no con herramientas más genéricas como en las aplicaciones web. Además, en cuanto a las posibilidades funcionales a desarrollar, las aplicaciones nativas son totalmente

compatibles con las características del hardware y la plataforma donde será ejecutada, volviendo la ejecución de la aplicación más eficiente. Sin embargo, las aplicaciones web o híbridas están limitadas al acceso a las características del dispositivo y se deberá realizar mediante librerías u otras utilidades.

Una vez seleccionado el tipo de aplicación a desarrollar, las plataformas seleccionadas para las cuales realizar un desarrollo multiplataforma serán:

- Android, cuyas interfaces son definidas mediante código XML Android. Esta plataforma ha sido seleccionada debido a que es la plataforma más utilizada en los dispositivos smartphones en el último año [2].
- Windows, cuyas interfaces son definidas mediante código XAML. Esta plataforma ha sido seleccionada debido al conocimiento adquirido en la realización del trabajo de investigación “Lenguaje de dominio específico – XAML” en la asignatura de “Generación Automática de Código”, en el cual se lograba generar código XAML a partir de un DSL inventado;

En los siguientes apartados, se va a proceder a realizar un listado de los requisitos funcionales y no funcionales que la herramienta debería de tener para poder ser integrada en un proceso de desarrollo multiplataforma de aplicaciones móviles en el ámbito de las interfaces de usuario. La figura 1 muestra como la herramienta buscada debe encajar en el proceso de desarrollo.

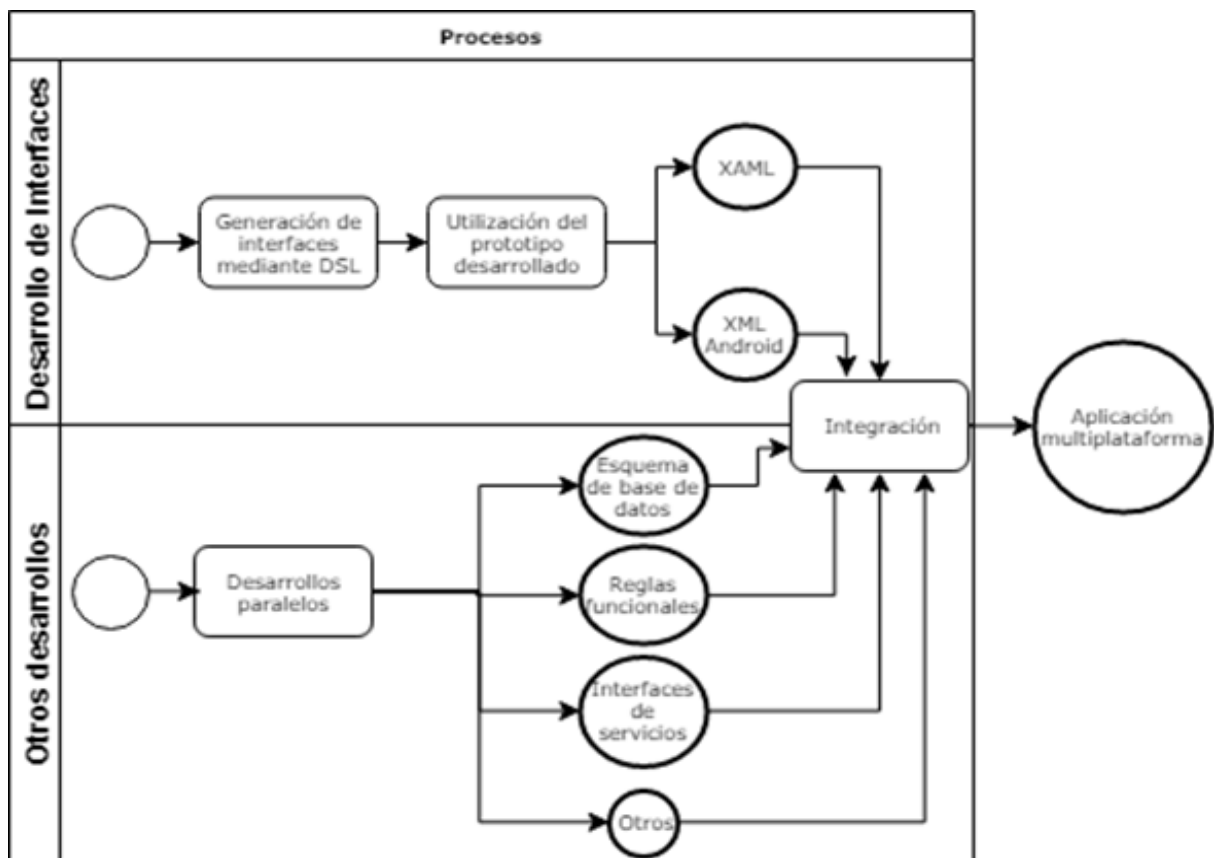


Figura 1 Aplicación del prototipo en un proceso de desarrollo de una aplicación multiplataforma

## 2.1 Requisitos funcionales

- Generación de ficheros XAML o XML Android a partir de una entrada DSL  
La herramienta debe de tener como entrada un fichero en lenguaje DSL y poder generar ficheros en lenguajes XAML y XML Android cumpliendo la definición contenida en fichero DSL.
- Traducción de ficheros XAML a XML Android y viceversa  
La herramienta debe de tener como entrada un fichero XAML o XML Android y realizar una traducción al lenguaje nativo de la otra plataforma manteniendo la misma definición de interfaz de usuario.
- Traducción de ficheros de definición de interfaz de usuario de un proyecto Window a Android y viceversa  
La herramienta debe de poder traducir todos los ficheros de definición de interfaz de usuario de un proyecto, Window o Android, a lenguaje nativo de definición de interfaz de otra plataforma.

## 2.2 Requisitos no funcionales

- No se debe de tener conocimiento técnico para el desarrollo mediante DSL  
El usuario que realice la definición de interfaz de usuario en el lenguaje DSL no debe de tener la necesidad de tener conocimiento de XAML, XML Android ni de ningún otro lenguaje de programación o definición de interfaz de usuario.
- Permitir la modificación de los ficheros generados  
Es necesario que permita la manipulación de los ficheros generados para que el usuario pueda realizar las modificaciones deseadas según la plataforma.
- La generación o traducción debe de ser parametrizable  
La herramienta debe de permitir parametrizar la generación o traducción de código. Debe de contar con un mecanismo para que el usuario pueda definir sus propias reglas de traducción para cumplir con las necesidades existentes según la plataforma deseada para la cual generar código.
- Facilidad de integración en proyectos de desarrollo  
La herramienta debe permitir ser integrada con sencillez en cualquier proceso de desarrollo. El proceso de desarrollo no debe estar limitado por las características del generador y debe permitir dividir el trabajo asociado al desarrollo de una aplicación (por ejemplo, independizar la definición de la interfaz de usuario al desarrollo de la lógica de negocio).

### 3 Estado del arte

En el presente apartado se va a proceder a realizar una búsqueda de frameworks o herramientas que ofrezcan utilidades y beneficios para el desarrollo de aplicaciones nativas multiplataforma en el ámbito de las interfaces de usuario. La investigación se realizará sobre los siguientes campos:

- Generación de código de interfaz de usuario a partir de código DSL: se realizará un análisis sobre herramientas que permitan generar código perteneciente al ámbito de definición de interfaces de usuarios a partir de código DSL. De esta forma, la definición de interfaces la puede realizar un desarrollador sin la necesidad de tener conocimiento de un lenguaje técnico que pueda ser difícil, si no utilizando un lenguaje de dominio más acorde a sus capacidades y conocimientos
- Definición de interfaces: se realizará un análisis de varias herramientas para comprobar si permiten la definición de interfaces de forma sencilla, interactiva y sin conocimientos técnicos. También se comprobará en que formatos se pueden exportar los resultados de la definición de interfaces para ser utilizados en procesos de desarrollo de aplicación.
- Conversores de código: se realizará un análisis sobre herramientas que permitan la transformación de código entre distintos lenguajes dentro del ámbito de definición de interfaces de usuario. Se comprobará si los lenguajes a los que son transformados el código de entrada sirven para definir interfaces para aplicaciones móviles nativas.
- Frameworks de desarrollo de aplicaciones multiplataformas: se estudiarán frameworks más complejos destinados al desarrollo de aplicaciones móviles multiplataforma. Se comprobará si es posible independizar la generación de interfaces de usuario, si se genera código de interfaz nativo para distintas plataformas y si esta generación puede ser parametrizable y modificada.

Sobre estas tecnologías, se realizará un análisis para comprobar en que grado satisfacen los requisitos establecidos en el apartado “2 Objetivos”. Para finalizar, y con los resultados de este estudio, se concluirá si existe alguna tecnología que sirva como generador de código a partir de DSL, permitiendo independizar el trabajo de generación de interfaces de usuario a otros procesos de desarrollo; y que, además, sirva de traductor de código entre distintas plataformas, permitiendo realizar migraciones de aplicaciones de una plataforma a otra.

## 3.1 Generación de código de interfaz de usuario a partir de código DSL

Un DSL es un lenguaje de programación creado para la resolución de un problema dentro de un dominio en particular. Lo opuesto a un lenguaje DSL es un lenguaje de programación de propósito general como C# o Java. La creación de un DSL es beneficioso en cuanto a términos de productividad y tiempo invertido en su desarrollo para facilitar la resolución de un problema en los casos que no exista otro lenguaje de programación o herramienta que permita resolver problemas en el dominio deseado.

Dentro del ámbito de generación de interfaces de usuario, se puede encontrar herramientas como DSL::HTML [3] o Lucid [4], las cuales permiten la generación de código HTML a partir de un lenguaje DSL. En primer lugar, DSL::HTML se centra en facilitar la construcción de árboles de elementos HTML a partir de plantillas y módulos reutilizables y parametrizables. No se trata de un DSL creado desde cero, se basa en Perl [5], un lenguaje de programación imperativo orientado al procesamiento de texto. Al basarse en un lenguaje ya existente y utilizar sus características, ya cuenta con un compilador y un corrector sintáctico que facilite el desarrollo. En resumen, este DSL permite a un programador crear interfaces de usuario sin necesidad de conocer las reglas de HTML.

Por otro lado, el DSL Lucid tiene un carácter menos algorítmico y se centra más en la generación de texto basándose en realizar sustitución de texto basándose en plantillas con reglas de traducción. El DSL es más bien un pseudo código HTML simplificado que la herramienta procesa y genera código HTML final.

Respecto al tema de investigación que nos ocupa, también encontramos un DSL capaz de generar código nativo para la plataforma de Android. Se trata de Anko [6], que mediante un lenguaje de programación declarativo permite generar XML Android con pocas instrucciones.

Todas estas herramientas cumplen su función de DSL dentro del ámbito de generación de interfaces de usuario. Sin embargo, no permiten la generación de código final en varios lenguajes. Son concedidas para cumplir el único objetivo de generar código en un único lenguaje. Además, sus lenguajes están orientados a tener cierto carácter de programación declarativa o imperativa. Lo ideal sería la utilización de un lenguaje más sencillo para que el diseñador de interfaces de usuario no tuviera que tener nociones de programación algorítmica.

## 3.2 Definición de interfaces

Existen infinidad de herramientas que permiten el diseño de interfaces de usuario para aplicaciones o páginas web de forma interactiva, es decir, permitiendo arrastrar elementos e interactuar con ellos. Algunas de ellas, como MockFlow [7] o FluidUi [8], no solo permiten definir páginas de interfaz de usuario, sino que también permite la definición del flujo de navegación entre las distintas páginas o pantallas de la aplicación. Incluso el IDE de Windows Visual Studio permite la definición de interfaces de forma interactiva [9], generando código XAML a integrar directamente en un proyecto.

Los aspectos que hacen seleccionar una herramienta u otra en un proceso de desarrollo pueden ser varios:

- La integración de estas herramientas en un entorno de desarrollo y el acoplamiento a otros procesos de desarrollo. Por un lado, se pueden tratar de herramientas web, como FluidUi o Balsamiq [10], que tienen una gran capacidad de integración debido a que pueden ser utilizadas desde cualquier equipo; o bien pueden encontrarse más integradas en un framework, como Glade [11] la cual es una herramienta que se encuentra integrada dentro del conjunto de librerías de GTK+ y del IDE GNOME; o simplemente aplicaciones de escritorios como MockFlow o Balsamiq en su distribución como herramienta de escritorio [12].
- El formato de exportación. Estas aplicaciones permiten realizar el diseño de páginas o pantallas para que posteriormente este resultado pueda ser utilizado en los proyectos de desarrollo de aplicación para ser compilado dentro de la solución. Para todas de ellas, la posibilidad más probable es que permitan realizar la exportación de los diseños en ficheros PNGs o PDF, debido a que son contenidos visuales. Los formatos de exportación más interesantes son HTML, permitido por MockFlow [13] o FluidUi [14]; o XML permitido por Glade, que mediante la librería GtkBuilder es posible importar esta definición en distintos lenguajes de programación [15] para poder ser utilizada en el desarrollo de cualquier aplicación.

La principal utilidad de este tipo de herramientas es que permiten que un diseñador pueda definir interfaces sin necesidad de tener ningún tipo de conocimiento técnico de la plataforma destino para la cual el diseño se está realizando. Además, formatos de exportación como XML, HTML o XAML vuelve el resultado totalmente integrable en el proceso de desarrollo de aplicaciones. De forma particular, al realizar la exportación a HTML, es posible que el resultado pueda ser ejecutado en un navegador web pudiendo tener el feedback del desarrollador o de un usuario.

Herramientas más actuales permiten un paso más en este aspecto. Se tratan de herramientas que permiten generar código nativo de diseño de interfaces de usuario a partir de una imagen. Para ello, el proceso es el siguiente: primero procesan la imagen y



a partir de este análisis se genera código DSL específico de la herramienta y, a partir de este código DSL, generan el código nativo de interfaces de usuario para la aplicación deseada. Por ejemplo, UI2Code [16] [17], permite generar código Dart o XML Android.

Otra herramienta similar que se encuentra en fase experimental es Pix2Code [18], la cual, mediante aprendizaje automático que permite mejorar un modelo de referencia de traducción, permite generar automáticamente código a partir de una imagen con más del 77% de precisión para iOS, Android y aplicaciones web. A partir de este trabajo de investigación, se desarrolla Code2pix [19] [20], herramienta que hace lo contrario. Utilizando el mismo DSL definido para Pix2Code, genera una imagen con la definición de un fichero de definición de interfaz de usuario.

Aparte de que se tratan de herramientas en estado de investigación y que se encuentran en un marco más teórico que práctico, uno de los problemas encontrados en las herramientas Pix2Code y Code2pix es que la generación a partir de DSL no puede ser parametrizable. Como podemos encontrar en sus ficheros de referencia, para realizar la generación de código [21], la generación se realiza mediante sustitución de texto. Es decir, si se quiere generar un texto de color rojo, haría falta definir una especificación de generación para un elemento label con un atributo de color rojo; si se quiere definir un texto de color azul, es necesario definir otra especificación de generación. Sería necesario realizar muchas definiciones de especificaciones o bien esperar a que el modelo de traducción tenga bastantes datos adquiridos a partir del aprendizaje automático. Por otro lado, Code2pix permite realizar ingeniería inversa y generar DSL a partir de código fuente nativo... Pero no existe la posibilidad de realizar una traducción entre distintos códigos fuentes, lo cual daría la posibilidad de realizar la migración de aplicaciones a otras plataformas.

En lo que respecta a la motivación de este estudio, lo más interesante se encuentra en la salida de este tipo de aplicaciones. El formato de salida XML lo convierte a un firme candidato a convertirse en la entrada de un prototipo que genere código de interfaz de usuario nativo para distintas plataformas. Es decir, generar XAML o XML Android a partir del trabajo que ha realizado un diseñador gráfico sin tener conocimiento de estas tecnologías. De esta forma, se reduce el coste de recursos en generar interfaces de usuario en aplicaciones multiplataformas.

### 3.3 Conversores de código

Las herramientas de conversión de código, que permiten la traducción o transformación de código de un lenguaje a otro distinto, pueden estar orientadas a desarrolladores que no tienen conocimiento suficiente para desarrollar en un lenguaje y por eso parten de otro lenguaje si conocido. Aparte, también pueden tener una aplicación dentro de procesos de desarrollo de aplicaciones multiplataforma en los que se quiere que, a partir de un fichero de entrada definido en un lenguaje dado, se generen varios ficheros en distintos lenguajes. Esta última aplicación es la más interesante en el desarrollo de aplicaciones multiplataformas.

CSHTML5 [22] se trata de una extensión (gratuita para fines académicos) de Visual Studio que permite crear aplicaciones web. La aplicación es desarrollada mediante lenguaje XAML y C# en Visual Studio y, tras ser finalizada y compilada, se obtiene código HTML y javascript. De esta forma, la aplicación se ejecutará de forma nativa dentro de cualquier navegador, sin librerías ni necesidad de terceros.

Por otro lado, podemos encontrar xaml2html.net [23], una solución WPF, es decir, una aplicación de escritorio con interfaz de usuario Windows. Al ejecutarse esta aplicación se puede realizar la traducción de un fichero en XAML a lenguaje HTML y CSS. De esta forma, no es necesario que el desarrollador tenga conocimientos de HTML y CSS. Si no que, a partir de los conocimientos de los frameworks de la plataforma Windows, podrá crear aplicaciones Web con contenido nativo para navegadores.

Ambas herramientas son muy útiles para el desarrollo web ya que facilita a los desarrolladores experimentados en el desarrollo de aplicaciones web mediante tecnologías relacionadas con Windows realizar la transición al enfoque multiplataforma móvil. Un punto positivo dentro de la investigación realizada es que tanto CSHTML5 como xaml2html.net permiten diseñar interfaces de forma sencilla, de la misma forma que se ha descrito en el apartado “3.1. Definición de interfaces”. Esto se debe a que el código de entrada para los conversores es XAML y el IDE Visual Studio permite la definición de forma sencilla y sin conocimiento técnicos, es decir, mediante el arrastre de elementos y pudiendo interactuar con ellos.

Sin embargo, y con respecto a los requisitos, el contenido generado, HTML, JS y CSS, no sirve como código para la definición de interfaz de usuario en el desarrollo de aplicaciones nativas para plataformas móviles, XAML o XML Android en nuestro caso, solamente para aplicaciones híbridas o aplicaciones que se ejecuten mediante navegador. Por otro lado, la traducción no es parametrizable pese a que si se puede tener acceso al código modificado. Por último, en cuanto a la integración en algún proceso de desarrollo software, la necesidad de realizar la traducción mediante una aplicación de escritorio no es apenas integrable en cualquier otro proceso de desarrollo.

### 3.4 Frameworks de desarrollo de aplicaciones multiplataformas

Existen frameworks muy útiles que simplifican el trabajo de generar una misma aplicación para distintas plataformas. Por un lado, encontramos Xamarin [24], un framework de desarrollo que permite desarrollar una aplicación en código C# y generar a partir de este desarrollo aplicaciones nativas para plataformas iOS, Android o Windows Phone. Al tratarse de un framework de Microsoft, su IDE se encuentra dentro de Visual Studio. Respecto al ámbito de las interfaces de usuario, Xamarin pone a disposición del desarrollador varias utilidades. Entre ellas, Xamarin.Forms, un conjunto de herramientas dedicadas a la creación de interfaces de usuario que permite diseñarlas mediante código XAML. Una vez desarrollada una aplicación mediante C# y definida

su interfaz de usuario con XAML, se puede generar una aplicación nativa para la plataforma deseada.

Respecto a los requisitos deseados, funcionalmente solo permite la traducción de proyectos Windows a proyectos Android. No permite la traducción inversa ni la generación de código de interfaz nativo para otras plataformas a partir de un DSL y la generación de código para interfaz de usuario se encuentra limitada al diseño mediante XAML. Respecto a los requisitos no funcionales, al tratarse de un framework de Microsoft, es necesario tener conocimiento de sus herramientas propias (Visual Studio y XAML) y no se puede integrar en ningún otro proceso de desarrollo. El desarrollo de interfaces de usuario y la lógica de negocio se encuentran en el mismo IDE. Además, ningún proceso es parametrizable ni se tiene capacidad para modificar los ficheros con las interfaces generadas.

Otra herramienta, y menos ligada a las tecnologías de Microsoft es Flutter [25], un framework de Google orientado para el desarrollo de aplicaciones multiplataformas nativas. Permite desarrollar la aplicación en el lenguaje de programación Dart y obtener aplicaciones para las plataformas iOS, Android y Google Fuchsia, el nuevo sistema operativo que se encuentra en desarrollo por Google. La distribución de la herramienta permite que el desarrollo se puede realizar en los principales IDEs de desarrollo: Visual Studio, Xcode y Android Studio.

Analizando este framework frente a los requisitos deseados, la principal carencia se encuentra en cómo se realiza la definición de interfaces de usuario. El proceso se encuentra totalmente acoplado al desarrollo de la aplicación y es necesario tener conocimiento técnico para realizarlo sobre Dart y sus entidades Widgets. Lo único que puede solventar la necesidad de conocer este lenguaje de programación es la herramienta UI2Code [16] [17], que a partir de una imagen de un diseño de interfaz de usuario permite generar código Dart o XAML. Además, el resultado que da Flutter al generar las aplicaciones multiplataformas es un compilado y no los ficheros de definición de interfaz, para poder ser modificados en caso de tener esa necesidad.

Tanto Flutter como Xamarin generan aplicaciones nativas. Desde otro punto de vista y como ejemplo de framework que genera aplicaciones híbridas se encuentra Apache Cordova [26]. Se trata de un framework que permite el uso de HTML, CSS y Javascript para el desarrollo de una aplicación multiplataforma. El desarrollo de las interfaces se realizará como si se tratará de una página web debido a los lenguajes de programación utilizado y permite probar la aplicación en cualquier navegador web. Al finalizar la aplicación, el resultado se puede ejecutar en plataformas como Android, iOS o Windows. El resultado, una aplicación web, se ejecutará dentro de un contenedor en el dispositivo y mediante librerías APIs tendrá acceso a las características de cada dispositivo.

En resumen, se tratan de frameworks con mucho potencial. La posibilidad de desarrollar una aplicación, finalizarla y después realizar un proceso de migración a Android o iOS cubriría el principal problema en el desarrollo de aplicaciones multiplataformas. Pero en lo que respecta a la investigación a realizar y al dominio de generación de interfaces de usuario con código nativo para la aplicación deseada, el proceso se encuentra demasiado acoplado al desarrollo del resto de módulos de la aplicación. Además, no cuenta con

ninguna posibilidad de parametrizar el proceso de generación de código o modificar los resultados en el caso de que sea necesario, ya que, aunque Xamarin o Flutter generen código nativo, no es posible su acceso. En resumen, solo cubre parte de los requisitos.

### 3.5 Resultados de la investigación

Tras haber realizado un estudio en busca de herramientas que cumplan con los requisitos establecidos en el apartado “2 Objetivos”, llegamos a la conclusión de que no existe ninguna herramienta que cumpla con todos los requisitos a la vez. En la siguiente tabla se muestra de forma resumida en que grado las herramientas analizadas previamente cumplen los requisitos deseados. Para ello, la valoración que se le dará a una herramienta respecto al cumplimiento de un requisito será: “No lo cumple” (0), “Lo cumple con dificultades” (1) y “Lo cumple de forma satisfactoria” (2).

	Requisito	Identificador
Funcional	Generación de ficheros XAML o XML Android a partir de una entrada DSL	RF1
	Traducción de ficheros XAML a XML Android y viceversa	RF2
	Traducción de ficheros de definición de interfaz de usuario de un proyecto Window a Android y viceversa	RF3
No funcional	No se debe de tener conocimiento técnico para el desarrollo mediante DSL	RNF1
	Permitir la modificación de los ficheros generados	RNF2
	La generación o traducción debe de ser parametrizable	RNF3
	Facilidad de integración en proyectos de desarrollo	RNF4

Figura 2 Identificación de requisitos

Herramientas	Requisitos						
	RF1	RF2	RF3	RNF1	RNF2	RNF3	RNF4
DSL::HTML	0	0	0	1	2	2	1
Lucid	0	0	0	2	2	2	1
Anko	1	0	0	0	2	1	1
MockFlow	0	0	0	2	2	0	2
FluidUi	0	0	0	2	2	0	2
Balsamiq	0	0	0	2	2	0	2
Glade	0	0	0	2	2	0	0
Visual Studio	1	0	0	2	2	0	0

UI2Code	1	0	0	2	2	0	2
Pix2Code	2	0	0	2	2	2	2
Code2pix	2	0	0	0	2	2	2
CSHTML5	0	0	0	1	0	0	2
xaml2html.net	0	0	0	1	0	0	2
Xamarin	2	1	1	1	0	0	0
Flutter	2	0	0	0	0	0	0

Figura 3 Análisis de requisitos sobre herramientas

En cuanto a la generación de código a partir de un lenguaje DSL, no encontramos una herramienta que permita como entrada un lenguaje sencillo para poder generar ficheros con definición de interfaz de usuario nativa para plataformas Windows o Android. Lo ideal en el proceso de definición de interfaces de usuario sería utilizar una herramienta que reúna las características vistas en el apartado “3.1 Definición de interfaces”, sencillez e interactividad sin necesidad de tener ningún conocimiento técnico, y utilizar el resultado exportado de estas herramientas. Para mejorar el proceso, el formato de exportación de estas herramientas sería conveniente que fuera XML o algún lenguaje de marcado basado en XML como HTML, puesto que se trata de un estándar para el intercambio de información estructurada entre distintas aplicaciones y plataformas además del procesado y análisis de su contenido es sencillo.

Respecto a la traducción de ficheros nativos de definición de interfaces de una plataforma a otra no se ha encontrado ninguna herramienta robusta que no se encuentre más avanzado que un estado de investigación. Como se ha comprobado, las herramientas más robustas, como Xamarin o Apache Cordova, se enfocan en el desarrollo de una aplicación completa desde cero para luego terminar siendo exportadas o sirviendo como base para generar aplicaciones multiplataformas (nativas o híbridas). Pero no permiten ser utilizadas para el proceso de desarrollo de la migración de una aplicación ya finalizada a otra plataforma. Además, el resultado de este tipo de aplicaciones es una aplicación ya compilada y que no se puede modificar sus ficheros generados para poder ser modificados.

Además, contra más robustez cuenta una herramienta menos integrable es en otros proyectos. Como se ha visto, las herramientas cuentan con su propio IDE de desarrollo o bien son plugins para otros IDEs de desarrollo. Lo ideal sería que se trataran de librerías que, mediante la importación en proyectos o mediante la configuración de algún compilador como Maven, pudieran permitir ser ejecutadas en procesos independientes al generar la solución para cada plataforma.

## 4 Desarrollo de la solución

En el presente apartado, se va a proceder a definir y desarrollar un prototipo de generador de código de definición de interfaces nativas multiplataformas. Se centrará en cumplir con los requisitos establecidos en el apartado “2 Objetivos” y se basará en ideas y carencias encontradas en la investigación realizada en el apartado “3 Estado del arte”.

### 4.1 Diagrama de caso de usos

Tras realizar un análisis de los requisitos, el comportamiento del prototipo y las funcionalidades que debe permitir se encuentran reflejadas en el siguiente diagrama de uso.



Figura 4 Diagrama de caso de uso del prototipo

Como se puede apreciar en la figura 4, el usuario debería tener la capacidad de configurar los procesos de traducción y generación de código. Por esta razón, el DSL a utilizar por la aplicación debe encontrarse basado en esta configuración. Por otro lado, se le debería poder mostrar en cualquier momento las opciones disponibles en el prototipo.

Los principales procesos por ejecutar por el prototipo y que sirven para desempeñar toda la funcionalidad esperada son principalmente dos. La traducción de ficheros entre distintos lenguajes y la generación de ficheros a partir de la entrada de un fichero en DSL. Como se puede apreciar en la figura 5, el proceso de traducción de ficheros utiliza el proceso de generación de código a partir de DSL. Los dos subprocesos importantes por analizar y desarrollar son los de procesado de un fichero y la traducción de elementos y atributos. En los siguientes apartados se describirá como se ha diseñado la solución para estos dos procesos.

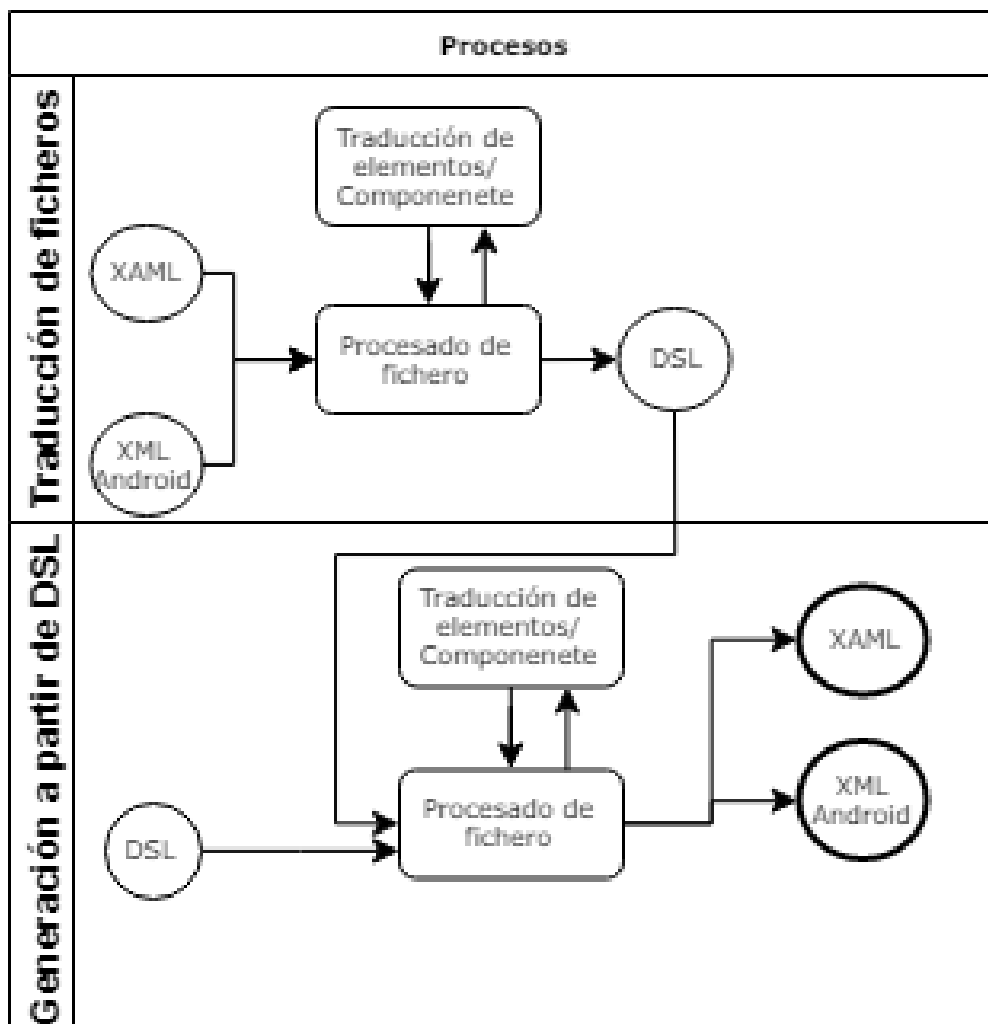


Figura 5 Diagrama de principales procesos del prototipo

Una vez desarrollada una solución para el procesado de fichero y traducción de elementos, se vuelve más sencillo la definición de los procesos del prototipo. En el diagrama de la figura 6, se puede apreciar cuales son las entradas y las salidas entre los distintos procesos de traducción y como se apoyan unos en otros para la reutilización de funcionalidad. Por ejemplo, la traducción de ficheros utiliza el proceso de generación de código final a partir de DSL.

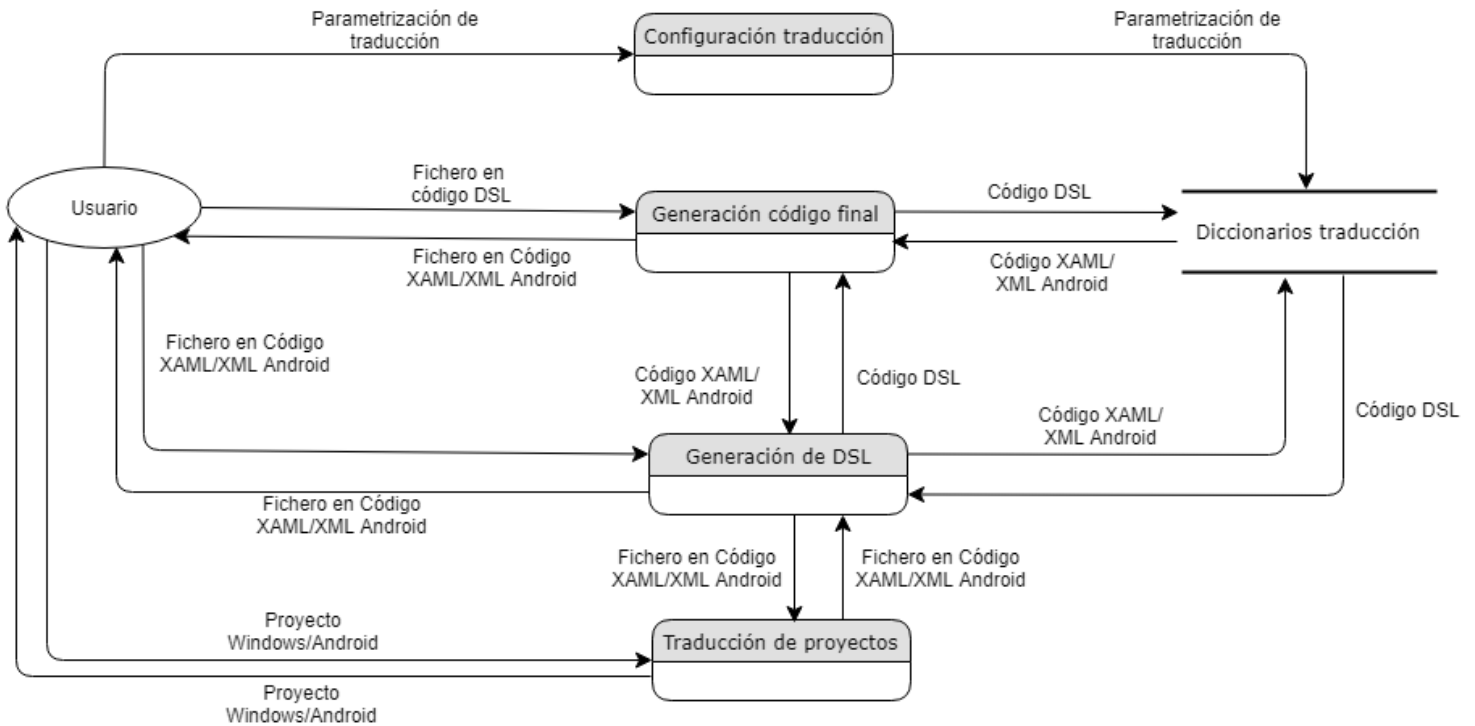


Figura 6 Diagrama de flujo de datos de la aplicación

## 4.2 Tecnología y recursos utilizados

El lenguaje de programación utilizado para realizar la solución es Java. La elección se ha basado en la experiencia y en el conocimiento propio tanto de las reglas del lenguaje como del entorno Eclipse. Por otro lado, se va a utilizar los IDEs Visual Studio y Android Studio para probar las salidas del prototipo desarrollado y el estudio de los lenguajes XAML y XML Android para el desarrollo del proceso de generación y traducción.

## 4.3 Lectura de ficheros y reconocimiento de elementos y atributos

La lectura de ficheros es un punto que solventar tanto para el proceso de generación de código a partir de DSL como para el proceso de traducción entre ficheros en lenguaje



XAML o XML Android. Que los tres lenguajes a tratar (XAML, XML Android y el DSL) cumplan con la normativa XML permite que el proceso de lectura e iteración de elementos sea en gran parte común y reutilizado.

Para la lectura y tratado de ficheros se ha utilizado el paquete de utilidades org.w3c.dom [27]. Esta librería permite iterar los elementos de un documento XML como si se tratase de un árbol de nodos. Mediante esta transformación, y de forma recursiva, es posible iterar todos los elementos de un fichero nativo de interfaz de usuario de forma sencilla y ordenada mientras que a la vez se genera otro árbol de nodos conteniendo la traducción en DSL. Lo mismo ocurre para el proceso contrario, la iteración de elementos en DSL para la generación de código final.

### 4.3.1 Lectura de Ficheros XAML y XML Android

Independiente del lenguaje del fichero de entrada, XAML o XML Android, el proceso de lectura es idéntico. El principal objetivo es recorrer los elementos XML e ir generando el resultado de la traducción en DSL. La entrada al proceso es la ruta en el sistema del fichero a tratar. Por otro lado, el resultado de ambos procesos es un objeto de tipo “org.w3c.dom.Document”, el cual representa un árbol de nodos y contiene los elementos XML en lenguaje DSL. Este objeto resultante será el utilizado internamente por el prototipo para generar el fichero traducido al lenguaje salida en los procesos de traducción.

Las diferencias entre los procesos de tratado de ficheros XAML y XML Android se encuentran en la identificación de tipo de elementos y atributos. Por esta razón, se ha decidido utilizar la herencia de java y la programación orientada a objetos. Los algoritmos comunes en los dos procesos se ubican en una clase padre de la cual heredan otras clases que definen los procesos propios de las casuísticas para los lenguajes XAML y XML Android (figura 4).

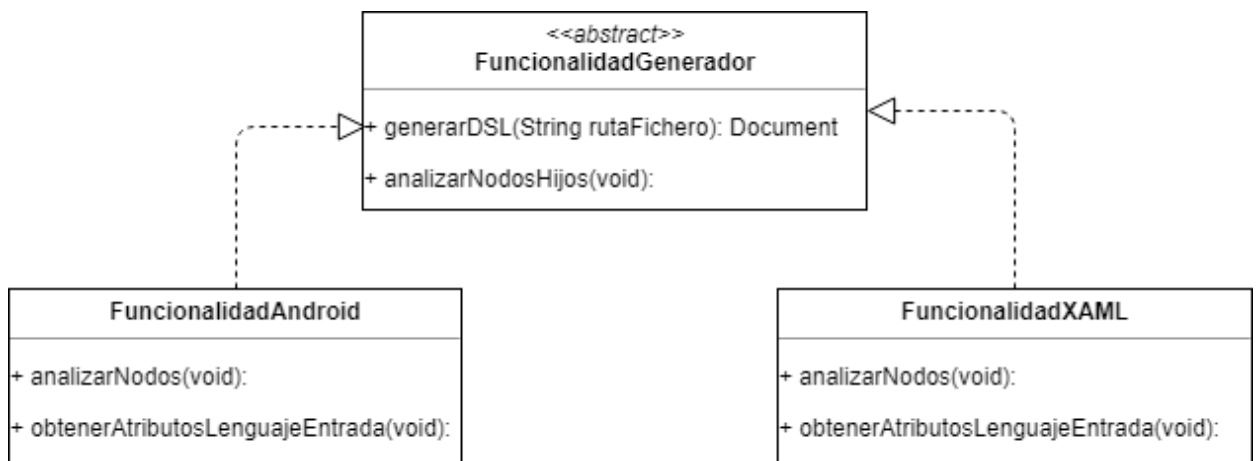


Figura 7 Diagrama de Clases: lectura ficheros plataformas

En el siguiente diagrama (figura 5) se puede observar el flujo desarrollado para el análisis de los nodos del fichero XML de entrada. Es importante identificar los elementos “Layouts” o elementos capas debido a que los elementos hijos de dicho nodo en el árbol DSL generado han de estar también asociados al nodo padre. De esta forma, se logra que la jerarquía de la salida se encuentre acorde a la jerarquía de elementos de la entrada y no se pierda información. Una vez estudiado el elemento de tipo layout y traducido sus atributos a su valor correspondiente en el código DSL se procede a analizar los elementos hijos asociados a este elemento Layout.

En el estudio de los elementos hijos de un nodo se comprobará si el tipo de elemento es reconocible. Esto significa que el elemento será tratado solamente si se encuentra en el listado de elementos a traducir en la configuración de la traducción de lenguaje XAML o XML Android a DSL. En caso de que así sea, se procede a su traducción a DSL y para ello se obtiene la traducción de sus atributos y sus valores en caso de que sea necesario traducir su valor.

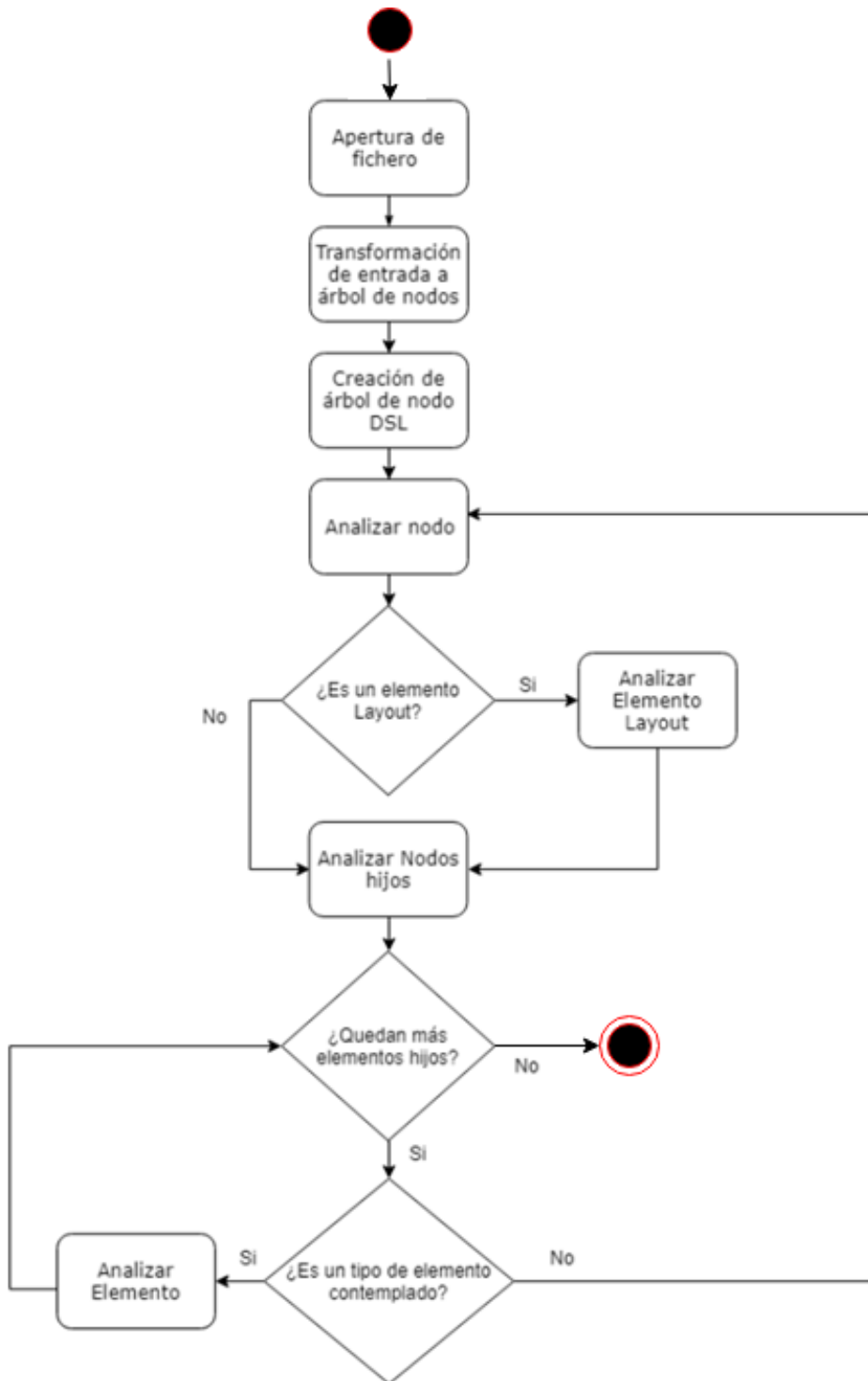


Figura 8 Diagrama de flujo de lectura de ficheros XAML o XML Android

En el siguiente diagrama (figura 6) se muestra mediante un diagrama de secuencia un ejemplo de generación de DSL a partir de código XAML o XML Android. En él, se muestra como el flujo de procesos se mueve entre la clase “FuncionalidadGenerador” y las clases hijas correspondientes a la funcionalidad XAML y XML Android según el tipo de lenguaje nativo del fichero de entrada.

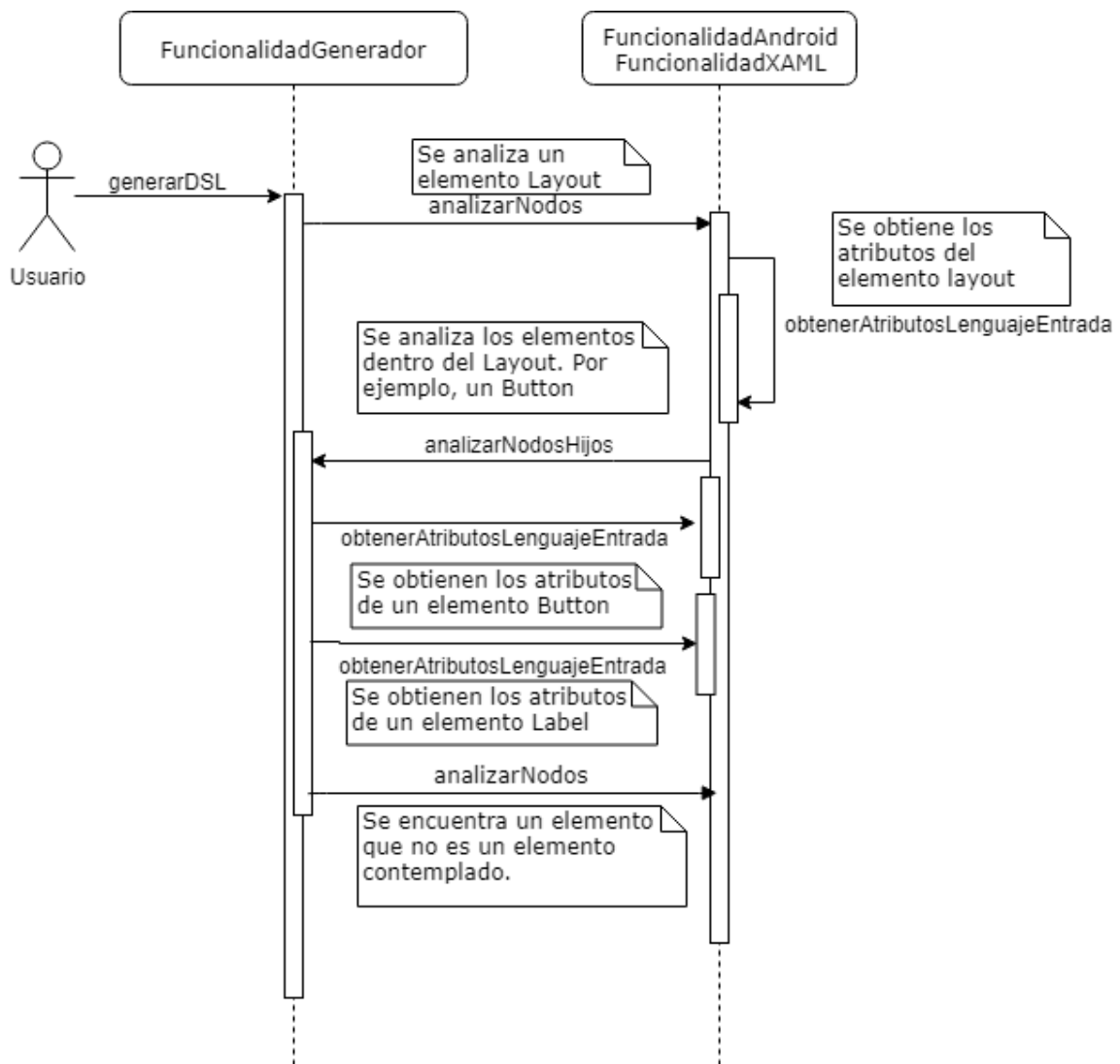


Figura 9 Diagrama de secuencia de lectura de ficheros XAML o XML Android

### 4.3.2 Lectura de Fichero DSL

El proceso de generación de código final a partir de DSL tiene varias peculiaridades en comparación al proceso descrito en el apartado anterior. Como se puede comprobar en la figura 7, el flujo de tratado del árbol casi no varía. Esto se debe a que ambas entradas son ficheros XML y la jerarquía de elementos es igual. El cambio se encuentra principalmente en el resultado, que en este caso es un fichero de texto y no contenido DSL. Por lo que, a la vez que se procesan elementos y atributos DSL, se irá generando texto traducido a escribir en ficheros.

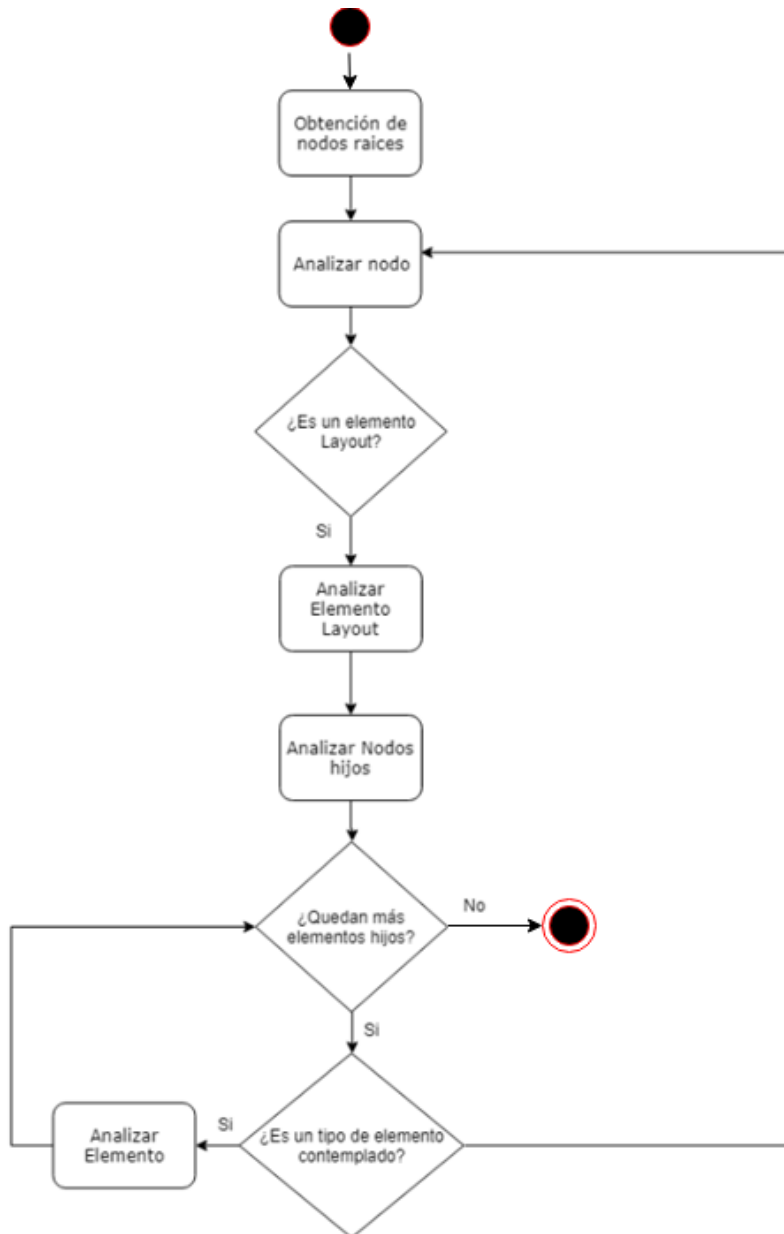


Figura 10 Flujo de lectura de árbol de Nodos DSL

La generación de ficheros se apoya en el uso de templates. El prototipo desarrollado cuenta con un template para XAML y otro para XML Android los cuales sirven de base para el contenido generado a partir del DSL. Aparte del contenido traducido, los templates necesitan el nombre de la clase o del fichero y el nombre del proyecto en el caso de los ficheros XAML. Estos datos son parámetros de entrada en este proceso. Estos ficheros templates se pueden llegar a modificar en el caso de que sea necesario por las necesidades del usuario.

## 4.4 Análisis de XAML y XML Android

Para poder definir el DSL y los procesos del prototipo, se ha realizado un análisis de los lenguajes nativos de las plataformas Android y Window. El análisis se ha orientado a varios aspectos expuestos en los siguientes apartados.

- Procesos de generación de DSL a partir de XAML o XML Android.
- Proceso de generación de XAML o XML Android a partir de DSL.

### 4.4.1 Análisis para la generación de DSL

El problema de generación de DSL a partir de ficheros de definición de interfaces de usuario en lenguaje XAML y XML Android se ha reducido a tres posibles tipos de procesos de traducción. En ellos, se ha intentado buscar prioritariamente la parametrización del proceso y configuración de la traducción a realizar. En los siguientes apartados se va a tratar en detalle cada uno de estos casos y cuáles son los datos necesarios para realizar la configuración para que se aplique dicha traducción a DSL a la hora de utilizar el prototipo. En la figura 8 se muestra la regla a aplicar cuando el prototipo se encuentra un elemento XAML o XML Android y tendrá que decidir qué tipo de traducción va a realizar para generar el DSL indicado.

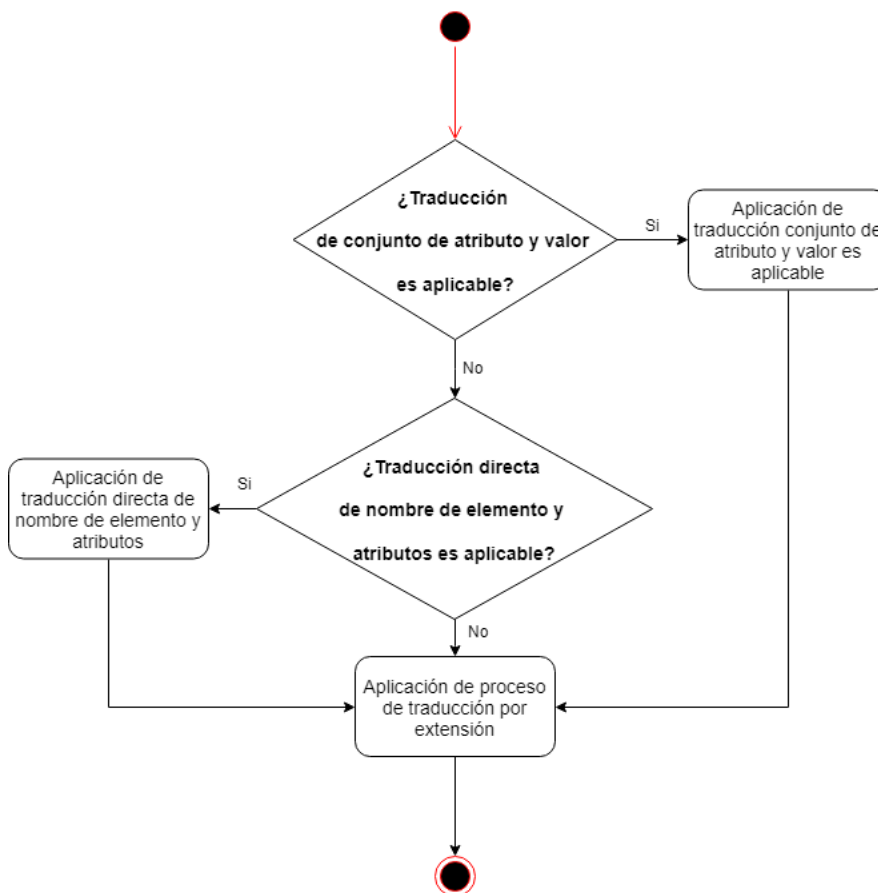


Figura 11 Diagrama de flujo de procesos de generación de DSL

#### 4.4.1.1 Traducción directa de nombre de elemento y atributos

En este tipo de traducción, el nombre de un elemento o un atributo equivalen a un único elemento o atributo en el DSL a generar. Como se puede observar en la tabla de la figura 9, la traducción se realiza uno a uno. El elemento “TextView” (XML Android) es equivalente a “TextBlock” (XAML). Por otro lado, el atributo “android:text” (XML Android) y “Text” (XAML) son equivalente independientemente del valor de los atributos. Se trata por tanto de una traducción totalmente parametrizable donde sería necesario indicar la palabra que se encontrará el prototipo y la palabra a la cual tendrá que traducirlo.

XML Android
<TextView android:text="Hello World" />
XAML
<TextBlock Text="Hello World" />

Figura 12 Ejemplo traducciones directas en la generación de DSL

#### 4.4.1.2 Traducción de conjunto de atributo y valor

Este proceso nace de la necesidad de cubrir casuísticas donde es necesario traducir el conjunto formado por un atributo y su valor. Esta necesidad se puede encontrar en varios casos:

1. Que los valores del atributo no son de texto libre sino de opciones acotadas. Por ejemplo, existen atributos cuyos valores puedan ser “False” o “True”. El problema radica que en XAML estos valores han de tener la primera letra mayúscula (IsChecked=”True”) y en XML Android minúscula (android:checked=”true”). En este caso, el traducir únicamente el nombre del atributo y mantenerse el mismo valor dará pie a un error.

Para solventar este problema, el prototipo al leer el código de entrada deberá generar un conjunto formado por el nombre del atributo y su valor que unifique estas diferencias entre lenguajes.

XML Android
<CheckBox android:checked="false" />
XAML
<CheckBox IsChecked="False"/>

Figura 13 Ejemplo traducción atributo y valor para atributo “checked” en DSL

- En XML Android, el valor “bottom” para el atributo “android:layout\_gravity” equivale a los atributos “HorizontalAlignment:Left” y “HorizontalAlignment:Bottom” en XAML. Es decir, un atributo en un lenguaje tiene el mismo valor de definición que dos atributos en otro lenguaje. Es por esta necesidad por la que hay que implementar un algoritmo que permita dividir un atributo en varios en el lenguaje DSL para que al realizar la traducción al otro lenguaje genere los dos atributos que lo equivalen.

XML Android
<code>&lt;CheckBox android:layout_gravity="bottom" /&gt;</code>
XAML
<code>&lt;checkbox HorizontalAlignment="Left" HorizontalAlignment:"Bottom" /&gt;</code>

Figura 14 Ejemplo traducción atributo y valor para atributo “android:layout\_gravity”

Para la definición de este tipo de traducción, será necesario especificar como elemento a reconocer el conjunto formado por el nombre del atributo en el lenguaje de entrada y el valor correspondiente, y definir la traducción a generar, que consistirá en pares de atributos con sus valores correspondientes en el DSL.

#### 4.4.1.3 Casos no parametrizables

A este último proceso de traducción se llega por descarte de los dos anteriores. Puede haber atributos y valores que no puedan ser traducidos por los dos procesos definidos anteriormente. Un ejemplo se encuentra en el atributo para definir el atributo “margin” para los elementos XAML y XML Android. En XAML su valor se define mediante un único atributo cuyo valor contiene el valor para los 4 márgenes posibles (arriba, abajo, derecha e izquierda) mientras que en XML Android estos valores se pueden definir en cuatro atributos distintos, uno por cada tipo de margen. Para estos casos, el proceso de traducción no es parametrizable.

XAML
<code>&lt;CheckBox Margin="50,30,46,10"/&gt;</code>
XML Android
<code>&lt;CheckBox android:layout_marginLeft="50" android:layout_marginUp="30" android:layout_marginRight="46" android:layout_marginDown="10"/&gt;</code>

Figura 15 Ejemplo traducción por extensión para el atributo “margin”



## 4.4.2 Análisis para la generación de código final

En este apartado el análisis se ha orientado a resolver el problema de que a partir de un código DSL generar ficheros de definición de interfaces de usuario en lenguaje XAML y XML Android. Para la resolución de este problema, se han identificado tres posibles tipos de procesos de traducción. En ellos, se ha intentado buscar prioritariamente la parametrización del proceso y configuración de la traducción a realizar. En los siguientes apartados se va a tratar en detalle cada uno de estos casos y como los datos necesarios para realizar la configuración para generar el código final. La aplicación de estos procesos y el flujo es el mismo que el diagrama de la figura 8.

### 4.4.2.1 Traducción directa de nombre de elemento y atributos

En este proceso, el nombre de un elemento o un atributo en el DSL de entrada equivalen a un único elemento o atributo en el fichero de salida. El proceso de traducción y configuración es igual al de la generación de DSL visto en el apartado “4.3.1.1 Traducción directa de nombre de elemento y atributos”, es decir, el nombre de un elemento XML se puede traducir por otro nombre y lo mismo ocurre con los atributos. El nombre del atributo se traduce por otro nombre y se mantiene el mismo valor.

Sin embargo, para este flujo de traducción, se ha encontrado que el valor funcional de un atributo en un lenguaje, entendiéndose como valor funcional la función que realiza dicho atributo sobre el elemento, pueda estar representado por distintos atributos en el otro lenguaje en función de elemento que contiene ese atributo.

Por ejemplo, el atributo en XML Android “android:text” puede ser utilizado para hacer referencia al texto que acompaña a un elemento “textBox”, el texto que hay dentro de un botón o bien el texto de un label. En XAML el nombre del atributo para realizar estas funciones difiere y en ocasiones es “Content” y en otras “Text” en función del elemento. Como se puede comprobar en el ejemplo de la figura 13, el atributo asociado a un elemento de interfaz de usuario para definir el texto de dicho elemento es distinto en XAML. Lo ideal es que en el DSL no importe el nombre de elemento, sino que se defina mediante un único atributo esta funcionalidad. En definitiva, para algunos casos no existe una traducción directa, sino que depende del elemento que lo contiene. Para ello, se ha de parametrizar también la posible traducción en función del elemento que contiene el atributo.

XML Android
<TextView android:text="Hello World"/> <Button android:text="Button" />
XAML
<TextBlock Text="Hello World" /> <Button Content="Button"/>

Figura 16 Ejemplo de traducción de atributo texto

#### 4.4.2.2 Traducción de conjunto de atributo y valor

Al igual que en la generación de DSL, a la hora de traducir de DSL a código final nos encontramos con problemas que la traducción directa no puede solucionar. Aunque, la especificación de estas traducciones sí que puede ser parametrizable.

- 1 Al igual que en la generación de DSL, hay ocasiones que los valores de los atributos no son de texto libre sino de opciones acotadas (figura 10). Para solventar este problema, el prototipo al leer el código de entrada generará un conjunto de atributo y valor que unifique estas diferencias entre lenguajes.
- 2 En otras ocasiones, varios atributos en DSL equivaldrán a un único atributo en el lenguaje generado. Por ejemplo, en XAML podemos encontrar los atributos `HorizontalAlignment:Left` y `HorizontalAlignment:Bottom` cuya traducción en XML Android es `android:layout_gravity:bottom` (figura 11). Si en el caso de generación de DSL, un atributo y su valor podía equivaler a uno o más pares de atributo y valor en DSL, en la generación de código final a partir de DSL es al revés. Varios pares de atributo y sus correspondientes valores equivalen a un único atributo y su valor en el código generado.

#### 4.4.2.3 Casos no parametrizables

Al igual que en el proceso de generación de DSL, debe existir un proceso que permita la traducción de código DSL a código nativo de definición de interfaz para casos en los que esta traducción no pueda ser parametrizada y no se pueda aplicar ningún proceso de los anteriores. Para comprobar su necesidad y siguiendo el problema expuesta en el apartado “4.3.1.3 Casos no parametrizables”, en el DSL los posibles atributos para definir los márgenes de un elemento (izquierda, derecha, abajo y arriba) se han de unificar en un único atributo en XAML (figura 12).

## 4.5 Configuración de la traducción

El prototipo debe permitir poder parametrizar el proceso de la generación de código y, por lo tanto, el proceso de realizar ingeniería inversa para poder realizar la traducción entre lenguajes de distintas plataformas. Para ello, es necesario que el DSL pueda ser configurado y modificado en función de las necesidades. El prototipo contará con un DSL ya definido que deberá permitir la traducción de un conjunto de elementos y atributos para poder demostrar su funcionalidad. Aun así, este DSL no se tratará de un lenguaje cerrado al usuario, sino que deberá poder ser modificado.

El lenguaje DSL desarrollado se encuentra basado en XML. Los principales motivos para esta decisión son:

- Tanto XAML y XML Android se encuentra basados en XML. Por lo tanto, la generación del código DSL y la generación de código final se puede realizar de forma secuencial y a la par a del proceso de iteración del árbol de elementos XML.
- Se trata de un estándar para el intercambio de información estructurada entre distintas aplicaciones y plataformas. Esto podría permitir generar herramientas con IU para definir pantallas de forma sencilla y cuya definición en código de dichos ficheros se realice mediante este DSL. A partir de esta salida y mediante el prototipo de generación de código desarrollado, se generaría código XAML o XML Android.

### 4.5.1 Procesos parametrizables

Principalmente, el DSL servirá como lenguaje intermediario entre XAML y XML Android y se definirá a partir de la configuración que se realice de los procesos de traducción y generación DSL. Como conclusión del apartado “4.3 Análisis de XAML y XML Android”, se toma la decisión de realizar una definición de traducción en función del sentido del proceso:

- Definición de traducción entre XAML o XML Android a DSL: para cubrir casos como el visto en el apartado “4.3.1.2 Traducción de conjunto de atributo y valor”, donde un conjunto de atributo y su valor pueda equivaler a varios en el otro lenguaje. Para estos casos, el DSL servirá como lenguaje intermediario y contendrá la división de dicho atributo en el número de atributos necesario en el otro lenguaje nativo.

XML Android
<CheckBox android:layout_gravity="bottom" />
DSL
<checkBox HorizontalAlignment="Left" HorizontalAlignment:"Bottom" />
XAML
<checkBox HorizontalAlignment="Left" HorizontalAlignment:"Bottom" />

Figura 17 Ejemplo traducción atributo y valor para atributo "android: layout\_gravity" mediante DSL

- Definición de traducción entre DSL a XAML o XML Android: para cubrir varios problemas. El primero de ellos el visto en el apartado "4.3.2.1 Traducción directa de nombre de elemento y atributos", donde la funcionalidad realizada por un único atributo para un lenguaje la realicen varios atributos en función del tipo de elemento que contenga a dicho atributo en otro lenguaje. Por ejemplo, en la figura 13, la funcionalidad de definir el texto asociado a un elemento la realiza el atributo "TextContent" en el DSL. Es necesario definir la traducción de dicho atributo en XAML. El otro problema es el ya visto en la figura 14, donde varios pares de conjuntos de atributo y sus valores (HorizontalAlignment="Left" y HorizontalAlignment:"Bottom") pueden equivaler a un único atributo y su valor en el XML Android (android: layout\_gravity ="bottom")

XML Android
<TextView android:text="Hello World"/> <Button android:text="Button" />
DSL
<text textContent="Hello World"/> <button textContent="Button"/>
XAML
<TextBlock Text="Hello World" /> <Button Content="Button"/>

Figura 18 Ejemplo de traducción de atributo texto mediante DSL

La especificación y la parametrización de estas traducciones se realizarán en ficheros con extensión ".propertie". Estos ficheros se encontrarán dentro del prototipo y podrán ser ejecutados antes de ser realizada cualquier traducción.

- diccionarioDSLXAML.propertie: configuración de traducción de DSL A XAML.
- diccionarioDSLAndroid.propertie: configuración de traducción de DSL A XML Android.
- diccionarioXAMLDSL.propertie: configuración de traducción de XAML a DSL.
- diccionarioAndroidDSL.propertie: configuración de traducción de XML Android a DSL.

#### 4.5.1.1 Especificación de generación de DSL a partir de código

En este apartado se va a detallar las decisiones tomadas para la configuración de la generación de elementos y atributos en el lenguaje DSL a partir de un fichero de entrada en lenguaje XAML o XML Android. Se detallará las normas de como se ha de realizar la especificación en los ficheros “diccionarioAndroidDSL.propertie”, para la traducción de XML Android a DSL, y “diccionarioXAMLDSL.propertie”, para la traducción de XAML a DSL.

##### 4.5.1.1.1 Traducción directa de nombre de elemento y atributos

Para realizar la configuración correspondiente de este tipo de traducción en los ficheros “.propertie“, el formato es el siguiente:

(Nombre de elemento/atributo en código origen)//(Nombre de elemento /atributo en código DSL)

*Figura 19 Formato para traducción directa de nombre de elemento y atributos en la generación de DSL*

Tomando como referencia el ejemplo de la figura 9, en el fichero “diccionarioAndroidDSL.propertie” encontraríamos la siguiente configuración:

TextView//text  
android:text//textContent

*Figura 20 Ejemplo para traducción directa en la generación de DSL desde Android XML*

- “TextView” es el elemento XML Android para poder mostrar textos. “text” será el nombre del elemento que permita realizar la misma función en el lenguaje DSL.
- “android:text” es el atributo que permite definir el texto que mostrará el elemento “TextView” en XML Android. En el DSL generado, el atributo “textContent” realiza la misma función.

Para realizar la misma generación de DSL a partir de XAML la configuración de estas dos traducciones las podemos encontrar en el fichero “diccionarioXAMLDSL.propertie”:

TextBlock//text  
Text//textContent

*Figura 21 Ejemplo para traducción directa en la generación de DSL desde XAML*

- “TextBlock” es el elemento XAML para poder mostrar textos. “text” será el nombre del elemento que permita realizar la misma función en el lenguaje DSL.
- “Text” es el atributo que permite definir el texto que mostrará el elemento “TextBlock” en XML Android. En el DSL generado, el atributo “textContent” realiza la misma función.

#### 4.5.1.1.2 Traducción de conjunto de atributo y valor

La especificación de este tipo de traducción más compleja se realiza en los mismos ficheros que la especificación de traducciones del apartado anterior. El prototipo detecta la diferencia de tratado debido al formato en que se define la especificación. El formato para poder definir este tipo de traducciones es el siguiente:

```
Atributo lenguaje fuente//valor del atributo//(Atributo lenguaje DSL//valor del atributo en DSL) repetidas tantas veces como atributos equivalga
```

Figura 22 Formato para traducción de conjunto de atributo y valor en la generación de DSL

En los ficheros de configuración podemos encontrar las siguientes especificaciones necesarias para cubrir los siguientes casos de traducción ya previamente analizados:

- 1 Si el lenguaje del fichero de entrada es XAML, traducir el atributo “IsChecked” y sus posibles valores (“False” y “true”), se deberá especificar en el fichero “diccionarioXAMLDSL.propertie” lo siguiente:

```
IsChecked//False//checked//false  
IsChecked//True//checked//true
```

Figura 23 Ejemplo para traducción conjunta para atributo isChecked desde XAML

Para realizar la misma traducción para ficheros XML Android, sería necesario especificar en el fichero “diccionarioAndroidDSL.propertie” lo siguiente:

```
android:checked//false//checked//false  
android:checked//true//checked//true
```

Figura 24 Ejemplo para traducción conjunta para atributo isChecked desde XML Android

- 2 Para solventar el problema visto en la Figura 14, donde un atributo y su valor (android: layout\_gravity = "bottom") equivalen a varios en XAML, y por lo tanto también en el DSL, la especificación necesaria en el fichero “diccionarioAndroidDSL.propertie” sería la siguiente:

```
android:layout_gravity//bottom//VerticalAlignment//Bottom//HorizontalAlignment//Left  
android:layout_gravity///VerticalAlignment//Top//HorizontalAlignent//Left
```

Figura 25 Ejemplo para traducción conjunta para atributo y valor android: layout\_gravity = "bottom" desde XML Android

Como se puede comprobar en la figura 22, el primer par de valores indica el conjunto de atributo y valor a detectar en el lenguaje de entrada (android:layout\_gravity y bottom). Los siguientes pares son los atributos con su valor correspondiente que se generarán en el DSL. Además, se puede dejar vacío el valor del atributo de entrada (ejemplo de la segunda línea). Esto permite que el prototipo convierta esta especificación como el caso por defecto cuando se encuentre el atributo, pero no el valor. Por ejemplo, con la especificación de la figura 22 y para cualquier valor para el atributo “android: layout\_gravity” que no sea “bottom”, la traducción sería la siguiente:

XML Android
<CheckBox android:layout_gravity="valor" />
DSL
<checkBox VerticalAlignment="Top" HorizontalAlignment:"Left" />

Figura 26 Ejemplo para traducción conjunta para atributo y valor android: layout\_gravity y valor por defecto desde XML Android

#### 4.5.1.2 Especificación de generación de código final a partir de DSL

En este apartado se va a detallar las decisiones tomadas para permitir la parametrización de la traducción entre DSL y código nativo de definición de interfaz de usuario, es decir, XAML y XML Android. Estas especificaciones se encuentran en los ficheros “diccionarioDSLAndroid.propertie” y “diccionarioDSLXAML.propertie”.

##### 4.5.1.2.1 Traducción directa de nombre de elemento y atributos

Para realizar esta traducción donde el nombre de un elemento o un atributo en el DSL de entrada equivalen a un único elemento o atributo en el fichero de salida, la especificación a realizar es similar a la vista en el apartado “4.3.1.1.1 Traducción directa de nombre de elemento y atributos”, puesto que la traducción es a la inversa.

(Nombre de elemento/atributo en DSL)//(Nombre de elemento /atributo en código final)

Figura 27 Formato para traducción directa de nombre de elemento y atributos en la generación de código final

Sin embargo y tal y como se ha visto en el apartado “4.3.2.1 Traducción directa de nombre de elemento y atributos”, existe otra regla para la especificación que cubre el caso en el que un mismo atributo de DSL puede ser traducido de forma distinta en función del elemento que lo contenga (figura 15). Para ello, se ha parametrizado también la posible traducción en función del elemento:

(Nombre de atributo en DSL)//(Nombre de elemento en código final-traducción del atributo para dicho elemento) Repetido tantas veces como posibles traducciones existan y separadas por el carácter “/” teniendo en cuenta que la última traducción serviría como caso por defecto

Figura 28 Formato para traducción directa de atributo en la generación de código final cuando el atributo puede ser traducido distinto en función del elemento que lo contiene

Para generar XAML a partir de DSL, en el fichero “diccionarioDSLXAML.propertie” se encontraría la especificación:

textContent//button-Content/checkBox-Content/option-Content/radioButton-Content/text-Text

Figura 29 Ejemplo de especificación de traducción de atributo en la generación de código final cuando el atributo puede ser traducido distinto en función del elemento que lo contiene

Para dicha especificación, al encontrarse el prototipo con el atributo “textContent” en código DSL, comprobaría el elemento que contiene el atributo en lenguaje DSL. En el

caso de que el elemento fuera de tipo “bottom”, “checkbox”, “option” o “radioButton”, la traducción generada del atributo sería “content”. En los demás casos, la traducción sería “Text” debido a que es la última traducción especificada.

#### 4.5.1.2.2 Traducción de conjunto de atributo y valor

En este apartado se tiene en cuenta los casos en los que se necesita parametrizar la generación de código a partir de DSL cuando, en el fichero DSL, varios pares de atributos y sus valores correspondientes son traducidos como un único par de atributo y su valor (figura 14). Las especificaciones de este tipo de traducción se realizan en los mismos ficheros citados en el apartado anterior: “diccionarioDSLAndroid.propertie” y “diccionarioDSLXAML.propertie”. El formato para poder definir este tipo de traducciones es el siguiente:

```
(Atributo lenguaje DSL //valor del atributo en DSL) repetidas tantas veces como sea necesario//Atributo lenguaje final//valor del atributo en lenguaje final
```

Figura 30 Formato para traducción de conjunto de atributo y valor en la generación de código final

En los ficheros de configuración podemos encontrar las siguientes especificaciones para cubrir el problema de traducción siguientes:

- 1 Para conseguir la traducción de la figura 10, en el caso de generar el atributo “isChecked” y sus posibles valores “False” o “True” en código XAML a partir de DSL, en el fichero “diccionarioDSLXAML.propertie” encontraríamos:

```
checked//false//isChecked//False  
checked//true//isChecked//True
```

Figura 31 Ejemplo para traducción conjunta para atributo isChecked desde DSL

En el caso de querer realizar lo mismo para generar en código XML Android el atributo “android:checked” y sus posibles valores “false” y “true”, sería necesario especificar en el fichero “diccionarioDSLAndroid.propertie” lo siguiente:

```
checked//false//android:checked//false  
checked//true//android:checked//true
```

Figura 32 Ejemplo para traducción conjunta para atributo android:checked desde DSL

- 2 Para el caso en el que varios atributos DSL pueden generar un único atributo en el lenguaje de salida (figura 14), la especificación necesaria en el fichero “diccionarioAndroidDSL.propertie” sería la siguiente:

```
HorizontalAlignment//Left//VerticalAlignment//Bottom//android:layout_gravity//bottom
```

Figura 33 Ejemplo para traducción conjunta para generar el atributo XML Android android:layout\_gravity y valor “bottom”



Según la especificación, cuando el prototipo se encuentre el atributo y valor `HorizontalAlignment="Left"` o `VerticalAlignment="Bottom"` en DSL, comprobará si el mismo elemento contiene todos los pares de la especificación. Si es así, y si el elemento en lenguaje final no contiene ya el atributo en el contenido generado hasta el momento, se añadirá.

## 4.5.2. Especificación de DSL

En este apartado se va a describir cómo se ha realizado la configuración del DSL del prototipo. Este DSL se crea a partir de la especificación de los procesos de traducción. Cabe mencionar que, aunque el prototipo cuente con un DSL ya configurado, este se puede modificar realizando las especificaciones con el formato descrito en el apartado “4.4.1 Procesos parametrizables”.

### 4.5.2.1 Traducción de elementos

En este apartado, se va a describir cómo se ha realizado la configuración del prototipo para que permita realizar la generación y traducción de los principales elementos de UI. Este conjunto de elementos son los principales para poder construir formularios en aplicaciones y poder recolectar información del usuario. Se describirá como se representan estos elementos y sus principales atributos en el DSL para poder realizar sus traducciones en lenguaje XAML y XML Android.

#### 4.5.2.1.1 Text

En el DSL especificado, los elementos `<text>` representarán elementos de tipo texto cuya principal función es mostrar información al usuario en la pantalla.

<b>DSL</b>	<code>&lt;text textContent="Hello World"/&gt;</code>
<b>XAML</b>	<code>&lt;TextBlock Text="Hello World" /&gt;</code> <code>&lt;TextBlock&gt;Hello World&lt;/TextBlock&gt;</code>
<b>XML Android</b>	<code>&lt;TextView android:text="Hello World"/&gt;</code>

Figura 34 Elemento `<text>` en los distintos lenguajes

- XAML: En código XAML, estos elementos son la traducción de elementos <TextBlock>. El texto que se mostrará se puede definir mediante el atributo “Text” o entre las etiquetas del elemento.

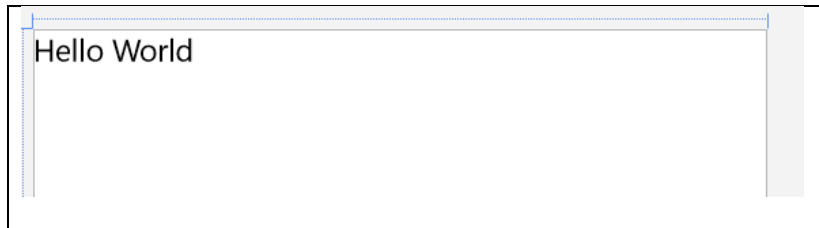


Figura 35 Imagen <TextBlock> en XAML

- XML Android: En código XML Android, estos elementos son la traducción de elementos <TextView>. El texto que se muestra en el elemento se encuentra en el atributo android:text.

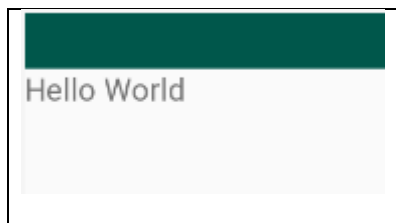


Figura 36 Imagen <TextView> en XML Android

Esta traducción se define mediante la especificación:

	<b>DSL</b>	<b>XML Android</b>	<b>XAML</b>
<b>Elementos</b>	TextView	TextView	TextBlock
<b>Atributos</b>	textContent	android:text	Text

Figura 37 Traducción de elemento <text> y sus atributos

#### 4.5.2.1.2 Button

En el DSL generado, los elementos `<button>` representan botones con los que el usuario podrá interactuar para que se realice alguna acción específica en la página. Por lo general, estas acciones suelen ser guardar los datos de un formulario o redirigir al usuario a otra página.

<b>DSL</b>	<code>&lt;button textContent="Button"/&gt;</code>
<b>XAML</b>	<code>&lt;Button Content="Button"/&gt;</code>
	<code>&lt;Button&gt;Button&lt;/Button&gt;</code>
<b>XML Android</b>	<code>&lt;Button android:text="Button" /&gt;</code>

Figura 38 Elemento `<button>` en los distintos lenguajes

- **XAML:** En código XAML, estos elementos son la traducción de elementos `<Button>`. El texto que se mostrará sobre el botón se puede definir mediante el atributo "Content" o entre las etiquetas del elemento.

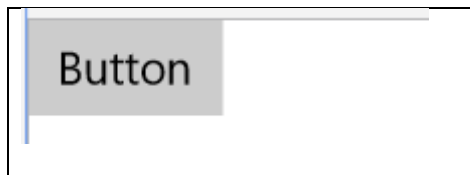


Figura 39 Imagen `<Button>` en XAML

- **XML Android:** En código XML Android, estos elementos son la traducción de elementos `<Button>`. El texto que se muestra en el botón se encuentra en el atributo `android:text`.

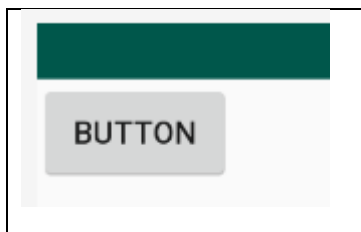


Figura 40 Imagen `<Button>` en XML Android

Esta traducción se define mediante la siguiente configuración de traducción de elementos y atributos:

	DSL	XML Android	XAML
<b>Elementos</b>	button	Button	Button
<b>Atributos</b>	textContent	android:text	Content

Figura 41 Traducción de elemento <button> y sus atributos

#### 4.5.2.1.3 Cajas de texto

En el DSL generado, los elementos <textBox> permiten al usuario introducir texto de forma manual. Es un elemento muy común en los formularios y es posible inicializarlo con un texto mediante el atributo “textContent” en el DSL configurado:

<b>DSL</b>	<textBox textContent="Introduce aquí tu texto"/>
<b>XAML</b>	<TextBox Text="Introduce aquí tu texto" />
<b>XML Android</b>	<EditText android:text="Introduce aquí tu texto"/>

Figura 42 Elemento <textBox> en los distintos lenguajes

- XAML: En código XAML, estos elementos son la traducción de elementos <TextBox>. Si se quiere mostrar con un texto por defecto, se realiza mediante el atributo “text”.



Figura 43 Imagen <TextBox> en XAML

- XML Android: En código XML Android, estos elementos son la traducción de elementos <EditText>. Si se quiere mostrar un texto por defecto, se puede definir mediante el atributo “android:text”.

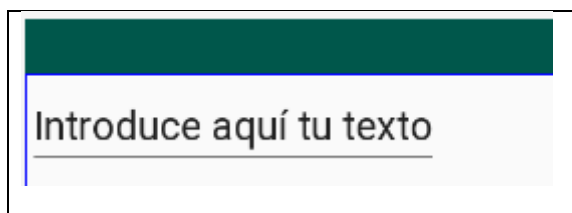


Figura 44 Imagen <EditText> en XML Android

Esta traducción se logra mediante la siguiente configuración de traducciones:

	<b>DSL</b>	<b>XML Android</b>	<b>XAML</b>
<b>Elementos</b>	textBox	EditText	TextBox
<b>Atributos</b>	textContent	android:text	Text

Figura 45 Traducción de elemento <textBox> y sus atributos

#### 4.5.2.1.4 CheckBox

En el DSL generado, los elementos <checkBox> representa un control de dos estados posibles para que el usuario pueda marcarlo como seleccionado o no seleccionado. Este elemento generalmente es útil para recolectar información que responde a preguntas simples cuyas posibles respuestas suelen ser verdadero o falso. Mediante el atributo “textContent” se puede definir el texto que acompañará al checkBox y mediante el atributo “checked” definir el estado en el que se encontrará el checkBox.

<b>DSL</b>	<code>&lt;checkBox checked="false" textContent="CheckBox"/&gt;</code>
<b>XAML</b>	<code>&lt;CheckBox Content="CheckBox" IsChecked="True"/&gt;</code>
<b>XML Android</b>	<code>&lt;CheckBox android:checked="false" android:text="CheckBox" /&gt;</code>

Figura 46 Elemento <checkBox> en los distintos lenguajes

- **XAML:** En código XAML, estos elementos son la traducción de elementos <CheckBox>. Mediante el atributo “Content” se puede definir el texto que acompañará al checkBox y mediante el atributo “isChecked” definir el estado inicial.

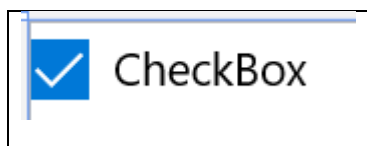


Figura 47 Imagen <CheckBox> en XAML

- **XML Android:** En código XML Android, estos elementos son la traducción de elementos <CheckBox>. Mediante el atributo “android:text” se puede definir el texto que acompañará al checkBox y mediante el atributo “android:checked” definir el estado inicial.

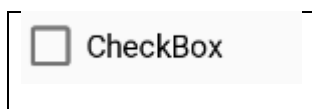


Figura 48 Imagen <CheckBox> en XML Android

Para configurar la traducción, debido a que los valores para los atributos son sensitivos a mayúsculas y minúsculas, no se puede realizar una traducción directa del atributo “checked”. Esto es debido a que a los valores “False” y “True” para el estado inicial del elemento “checkBox” ya que para XAML su primera letra ha de ser mayúscula y en XML Android minúscula. Por lo que la traducción de estos atributos ha sido configurada como una traducción compleja. Esto significa que se traduce tanto el nombre del atributo y su valor, y no únicamente el nombre del atributo manteniendo el valor del lenguaje entrada.

	<b>DSL</b>	<b>XML Android</b>	<b>XAML</b>
<b>Elementos</b>	checkBox	CheckBox	CheckBox
<b>Atributos</b>	textContent	android:text	Content
<b>Atributos y valores</b>	Checked ="false"	android:checked ="false"	IsChecked ="False"
	checked ="true"	android:checked ="true"	IsChecked ="True"

Figura 49 Traducción de elemento <checkBox> y sus atributos

#### 4.5.2.1.5 Select

En el DSL generado, los elementos <select> representa un control que permite al usuario poder seleccionar una opción entre un listado de opciones. Este listado de opciones son elementos del tipo <option>. El texto de estas opciones es posible definirlo mediante el atributo “textContent”. Además, es posible seleccionar una opción por defecto mediante el atributo “selected”.

<b>DSL</b>	<pre>&lt;select&gt;   &lt;option textContent="Item 1"/&gt;   &lt;option selected="true" textContent="Item 2"/&gt;   &lt;option textContent="Item 3"/&gt; &lt;/select&gt;</pre>
<b>XAML</b>	<pre>&lt;ComboBox&gt;   &lt;ComboBoxItem Content="Item 1"/&gt;   &lt;ComboBoxItem IsSelected="True" Content="Item 2" /&gt;   &lt;ComboBoxItem Content="Item 3"/&gt; &lt;/ComboBox&gt;</pre>
<b>XML Android</b>	<pre>&lt;Spinner /&gt;</pre>

Figura 50 Elemento <select> en los distintos lenguajes

- XAML: En código XAML, estos elementos son la traducción de elementos <ComboBox>. Los elementos disponibles en el listado de opciones son elementos de tipo <ComboBoxItem>. Respecto a atributos, es posible definir el texto de la opción mediante el atributo “Content” y definir el elemento seleccionado por defecto mediante el atributo “IsSelected”.

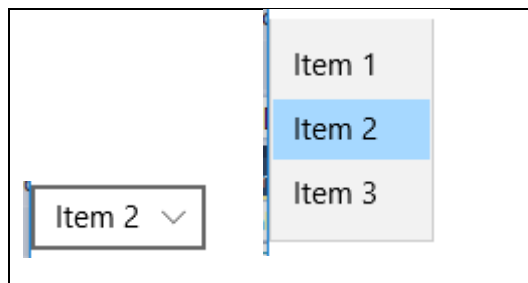


Figura 51 Imagen <ComboBox> en XAML

- XML Android: En código XML Android, estos elementos son la traducción de elementos <Spinner>. A diferencia de XAML, en XML Android no es posible definir el listado de opciones mediante código XML, sino que es necesario realizarlo de forma dinámica mediante código.



Figura 52 Imagen <Spinner> en XML Android

La generación y traducción es más productiva entre XAML y el DSL, puesto que en XML Android existe un alto nivel de acoplamiento debido a que el listado de opciones no se puede definir en ficheros XML Android. La traducción de estos elementos mediante el prototipo se logra mediante la siguiente configuración:

	DSL	XML Android	XAML
<b>Elementos</b>	select option	Spinner	ComboBox ComboBoxItem
<b>Atributos</b>	textContent		Content
<b>Atributos y valores</b>	Selected = "false" selected = "true"		IsSelected = "False" IsSelected = "True"

Figura 53 Traducción de elemento <select> y sus atributos

#### 4.5.2.1.6 RadioButton

En el DSL generado, los elementos `<radioButton>` representan un control de interfaz de usuario que permite al usuario poder seleccionar una opción entre un conjunto de opciones siempre visibles en pantalla. Cada una de estas opciones es un elemento `<radioButton>` independiente con dos estados posibles (seleccionado y no seleccionado) que suelen tener un atributo común para identificarlos como parte de un conjunto de opciones. Además, es posible seleccionar el estado inicial mediante el atributo “checked” y el texto a mostrar como opción mediante “textContent”.

<b>DSL</b>	<code>&lt;radioButton checked="true" textContent="Si"/&gt;</code> <code>&lt;radioButton checked="false" textContent="No"/&gt;</code>
<b>XAML</b>	<code>&lt;RadioButton Content="Si" IsChecked="True" /&gt;</code> <code>&lt;RadioButton Content="No" IsChecked="False" /&gt;</code>
<b>XML Android</b>	<code>&lt;RadioButton android:checked="true" android:text="Si" /&gt;</code> <code>&lt;RadioButton android:checked="false" android:text="No" /&gt;</code>

Figura 54 Elemento `<radioButton>` en los distintos lenguajes

- **XAML:** En código XAML, estos elementos son la traducción de elementos `<RadioButton>`. Para un uso adecuado de estos elementos, es posible definir el texto de la opción mediante el atributo “Content” y definir si el elemento se encuentra seleccionado por defecto mediante el atributo “IsChecked”.



Figura 55 Imagen `<RadioButton>` en XAML

- **XML Android:** En código XML Android, estos elementos son la traducción de elementos `<RadioButton>`. La opción se puede mostrar mediante el atributo “android:text” y el estado por defecto se puede seleccionar mediante el atributo “android:checked”.



Figura 56 Imagen `<RadioButton>` en XML Android



La traducción de este tipo de elementos se logra mediante la siguiente configuración:

	<b>DSL</b>	<b>XML Android</b>	<b>XAML</b>
<b>Elementos</b>	radioButton	RadioButton	RadioButton
<b>Atributos</b>	textContent	android:text	Content
<b>Atributos y valores</b>	Checked ="false"	android:checked ="false"	IsChecked ="False"
	checked ="true"	android:checked ="true"	IsChecked ="True"

Figura 57 Traducción de elemento <radioButton> y sus atributos

#### 4.5.2.2 Traducción de atributo

En este apartado se va a desarrollar como se ha logrado configurar el prototipo para conseguir la traducción de atributos. La traducción de atributos se logra mediante la siguiente clasificación de procesos.

##### 4.5.2.2.1 Traducción de atributos manteniendo valor

En este apartado se consideran aquellos atributos para los cuales solo se tiene la necesidad de traducir el nombre del atributo de un lenguaje a otro y se mantiene el mismo valor que en el código origen. Un ejemplo de este tipo de traducción sería la configuración del atributo "Width" y el resultado de su tratamiento en todos los lenguajes. Como se puede apreciar en la Figura 55, en todos los lenguajes se mantiene el mismo valor en cualquier proceso de traducción y el tratamiento solo se realiza sobre el nombre del atributo.

<b>DSL</b>	<b>XAML</b>	<b>XML Android</b>
width="150"	Width="150"	android:layout_width="150"

Figura 58 Traducción de atributo "width"

A continuación, se muestra la configuración realizada para la configuración del DSL:

<b>XAML</b>	<b>DSL</b>	<b>XML Android</b>
Height	height	android:layout_height
Width	width	android:layout_width
HorizontalAlignment	HorizontalAlignment	
VerticalAlignment	VerticalAlignment	

FontSize	size	android:textSize//size
Text	textContent	android:text
Content		
	marginLeft	android:layout_marginLeft
	marginUp	android:layout_marginTop
	marginRight	android:layout_marginRight
	marginDown	android:layout_marginBottom

Figura 59 Relación de traducción directa de atributos

Varias consideraciones a tener en cuenta:

- Debido a las reglas de XAML, el atributo “textContent” en DSL tiene un tratado especial como ya se ha visto en apartados anteriores. Debido a que hace referencia al texto que acompaña a un elemento (el texto de los labels, el texto de los botones, el texto que acompaña a los CheckBox...) su traducción en XAML puede ser el atributo “Content” o “Text” en función del tipo de elemento que lo contiene. Respecto a XML Android, toda esta funcionalidad está representado por un único atributo, “android:text”. La configuración realizada en el fichero “diccionarioDSLXAML.propertie” (donde se realiza la especificación de traducción entre DSL y XAML) es la vista en la figura 26.

Según esta especificación, el atributo “textContent” en DSL será traducido al atributo “Content” si el elemento que lo contiene es “button”, “checkBox”, “radioButton” o “button”. En los otros casos, su traducción será el atributo “Text”.

- Para los atributos DSL “HorizontalAlignment” y “VerticalAlignment” no existe traducción directa para XML Android debido a que Android trata el alineamiento de elementos mediante otro atributo. La traducción de este atributo para XML Android es tratada en el apartado “4.4.2.2.2 Traducción de atributos y valor”.
- Para los atributos DSL “marginLeft”, “marginUp”, “marginRight” y “marginDown” no existe traducción directa para XAML debido a que se trata de un caso especial y no parametrizable.

#### 4.5.2.2.2 Traducción de atributos y valor

En este apartado se va a tratar la traducción de aquellos atributos que han de ser traducido junto a su valor. Principalmente, esto se debe a que los valores para estos atributos son un conjunto cerrado de opciones y no textos libres como pueden ser la anchura o altura de un campo.

XAML	DSL	XML Android
IsChecked="False"	checked="false"	android:checked="false"
IsChecked="True"	checked="true"	android:checked="true"
IsSelected="False"	selected="false"	
IsSelected="True"	selected="true"	
	HorizontalAlignment="Left" VerticalAlignment="Bottom"	android:layout_gravity="bottom"
	HorizontalAlignment="Stretch" VerticalAlignment="Stretch"	android:layout_gravity="center"
	HorizontalAlignment="Center" VerticalAlignment="Center"	
	HorizontalAlignment="Left" VerticalAlignment="Top"	android:layout_gravity=""
	HorizontalAlignment="Center" VerticalAlignment="Center"	android:layout_gravity="center"
	HorizontalAlignment="Stretch" VerticalAlignment="Stretch"	android:gravity="center"
	HorizontalAlignment="Left" VerticalAlignment="Top"	android:gravity=""

Figura 60 Relación de traducción de atributos y valores

Varias consideraciones a tener en cuenta:

- Las flechas indican que dicha traducción no es bidireccional, si no que esos casos son solo realizados en función del lenguaje origen o lenguaje distinto. Por ejemplo, en la traducción de DSL a XML Android se puede llegar al atributo y valor “android:layout\_gravity="center"” desde dos casos; pero en la traducción de XML Android a DSL desde “android:layout\_gravity="center"” solo se llega a un conjunto de pares de atributos y valores en DSL.
- Los casos donde el valor del atributo es vacío sirve como casos por defecto cuando el atributo es encontrado pero el valor que contiene no se encuentra en la especificación como valor posible a traducir. Por ejemplo, con la configuración de la Figura 57 y en la traducción de XML Android a DSL, si el prototipo se encuentra con android:gravity="bottom" al no encontrarse esta traducción definida se aplica el caso por defecto, es decir, la traducción:

XML Android	DSL
android:gravity=""	HorizontalAlignment="Left" VerticalAlignment="Top"

Figura 61 Ejemplo de atraducción por defecto para “android:gravity”

- Para XML Android no existe traducción para los atributos y valores en DSL: “selected=”false”” y “selected=”true””. Esto es debido a que estos atributos son para elementos de tipo <option> y la traducción de estos elementos no se encuentran en XML Android ya que las opciones de los combos selects en XML Android se crean dinámicamente y no mediante código estático XML.

#### 4.4.3 Procesos no parametrizables

Como se ha comprobado en el análisis realizado en el apartado “4.3 Análisis de XAML y XML Android” de las características de los lenguajes XAML y XML Android, tanto en el proceso de traducir de DSL a código fuente como en el proceso inverso puede haber ocasiones donde la traducción no es parametrizable. Aunque no sea parametrizable, si es modificable. Esto es logrado mediante la orientación de objetos de Java. Este proceso puede ser modificado en implementaciones que hereden de las clases del prototipo y así conseguir configurar y añadir un procesado definido y creado por el usuario según sus necesidades.

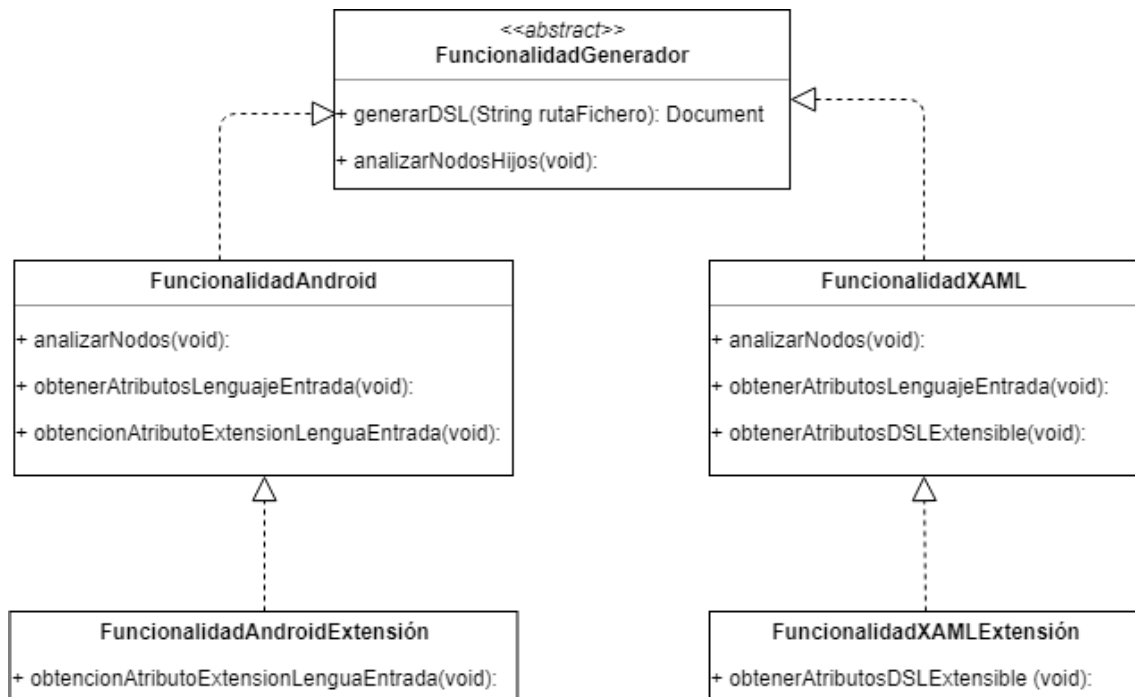


Figura 62 Diagrama de ejemplo de extensión de funcionalidad

## 4.6 Traducción de proyectos

En los anteriores apartados se ha detallado la solución desarrollada para los dos principales procesos para conseguir tanto la generación de ficheros a partir de código DSL como la traducción entre distintos lenguajes. Estos procesos son aplicados a ficheros de forma individual. Para dar más potencia a la herramienta, se ha desarrollado un proceso que permite traducir proyectos enteros.

Estos procesos no son parametrizables. Esto se debe a que se ha aplicado conocimiento específico de la estructura de ficheros de los proyectos. La herramienta ha sido programada para que busque y genere ficheros en rutas concretas según el tipo de proyecto a tratar, Windows o Android.

### 4.6.1 Traducción de Windows a Android

El proceso tomará la ruta de entrada del proyecto Windows y recorrerá todos los directorios buscando ficheros con extensión “.xaml”. Al encontrarlo, obtendrá el nombre del fichero y procederá a la generación de DSL a partir del contenido del fichero. A partir del DSL generado y con el nombre de la clase, generará la traducción a XML Android. El fichero generado se almacenará en el subdirectorio de la ruta del proyecto Android indicado como entrada del proceso: “\app\src\main\res\layout\”. En esta ruta se encuentra los ficheros de definición de interfaces XML Android para los proyectos Android.

### 4.6.2 Traducción de Android a Windows

El proceso tomará la ruta de entrada del proyecto Android y tratará el subdirectorio “\app\src\main\res\layout\” de la ruta indicada, ruta donde se encuentra los ficheros de definición de interfaces XML Android. Para todos los ficheros con extensión “.xml”, se generarán contenido DSL y se realizará la traducción a XAML. Los ficheros generados, con el mismo nombre que los ficheros XML Android, se crearán en el subdirectorio de la ruta del proyecto Windows: “\nombre del proyecto\”, siendo “nombre del proyecto” el nombre del proyecto que se obtendrá de la ruta de entrada del proyecto Window.

## 5 Caso Práctico

En este apartado se va a proceder a aplicar el prototipo en varios procesos para demostrar su correcto funcionamiento. Prioritariamente, se validará de forma gráfica que los resultados obtenidos, independientemente de la plataforma, son idénticos y puedan ser utilizados. Los códigos fuentes de las entradas y salidas del prototipo en cada una de las pruebas se pueden encontrar en el apartado Anexo, dentro del apartado “Ejemplos de generación de páginas”.

### 5.1 Generación de páginas

La generación de páginas de forma individual es posible realizarse con el prototipo mediante dos vías distintas: mediante la generación a partir de código DSL y mediante la traducción de un fichero de otra plataforma. En este apartado se va a proceder a utilizar el prototipo para demostrar tanto la utilidad como el buen funcionamiento del prototipo desarrollado en la generación de páginas XAML y XML Android desde código DSL y la traducción de XAML a XML Android y viceversa. Para lograrlo, se va a utilizar el prototipo desarrollado y detallado en el trabajo sobre un conjunto de páginas pertenecientes a una aplicación sencilla. Salvo el aspecto visual de los elementos que pueden diferir por la plataforma, los resultados no deberían variar entre una plataforma y otra.

Las pantallas generadas en los siguientes apartados podrían formar parte de una aplicación de gestión de usuarios. Por ello, cuenta con las siguientes páginas:

- Página de login: El usuario podrá entrar con su cuenta o bien seleccionar darse de alta y registrarse.
- Página de registro: El usuario podrá introducir una serie de datos personales necesario para poder darse de alta en el sistema.

#### 5.1.1 Página de Login

Se trata de la primera pantalla al entrar en la aplicación. Cuenta con los campos necesarios para iniciar sesión: usuario y contraseña; y dos botones: uno para realizar el inicio y otro para registrarse en caso de que el usuario no cuente con una cuenta. Un boceto visual de la página buscada sería el siguiente:



Figura 63 Boceto de página Login

#### 5.1.1.1 Generación a partir de DSL

Para su definición DSL, es necesario ordenar los elementos en varias capas layout para lograr una distribución adecuada. Además, es necesario la utilización de atributos para especificar alineación, anchura, altura o textos entre otros para lograr el resultado esperado. De esta forma, se comprobará si se realiza la traducción de estos atributos de forma esperada.

Tras la ejecución del prototipo, se puede apreciar que en la ruta de salida especificada en la entrada al prototipo encontramos los dos ficheros generados. Un fichero con el código generado XAML para la plataforma Window y otro fichero en lenguaje XML Android para la plataforma Android. Tras abrir su contenido en los IDEs correspondientes para cada plataforma, se puede comprobar que visualmente los resultados son iguales, salvo por los estilos propios de cada plataforma para los elementos de interfaz de usuario. Por ejemplo, para las cajas de texto, si no se indica lo contrario, en XAML aparecerán con borde y en XML Android no.

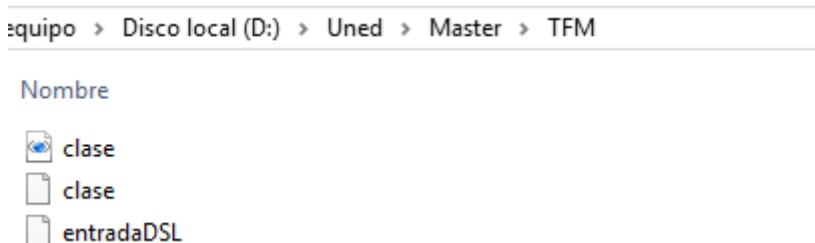


Figura 64 Imagen con fichero XML Android y XAML generado a partir de DSL

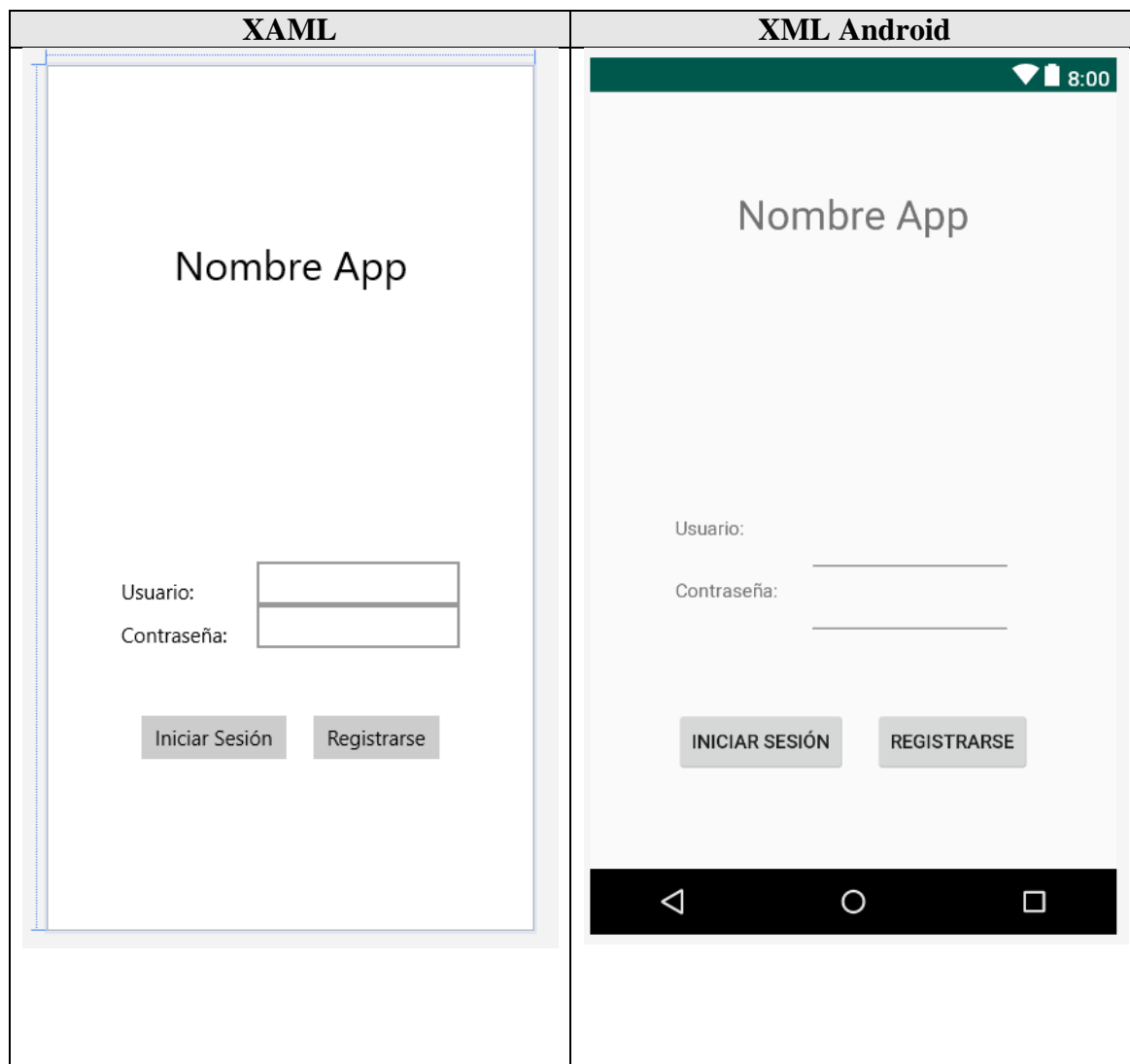


Figura 65 Páginas de Login generadas a partir de DSL

### 5.1.1.2 Generación como traducción

Partiendo de los resultados obtenidos en el apartado anterior, un fichero en lenguaje XAML y otro en XML Android, se va a aplicar el prototipo sobre ellos para realizar traducciones a otros lenguajes. Se comprobará que, visualmente, no hay variación en el resultado obtenido independientemente del proceso de generación: generando el fichero a partir de un fichero en lenguaje DSL o a partir de la traducción de un fichero en el lenguaje de otra plataforma.



### 5.1.1.2.1 Traducción XML Android a XAML

A continuación, se compara que el resultado visual obtenido por los siguientes procesos son los mismos:

- Página de login en XAML obtenido por generación a partir de DSL.
- Página de login en XAML obtenido al realizar la traducción del fichero XML Android obtenido mediante la generación a partir de DSL.

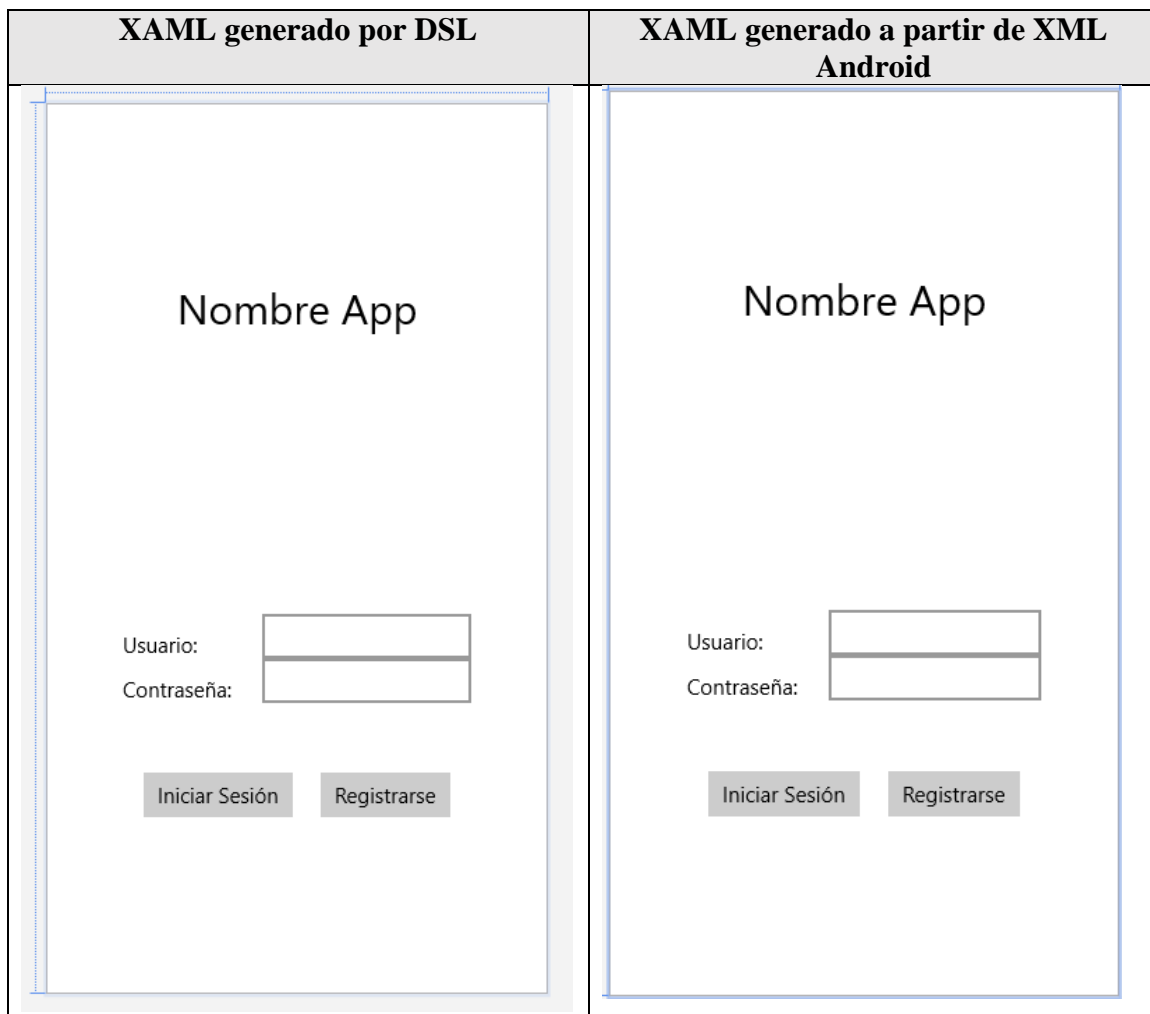


Figura 66 Página de Login XAML generada como traducción de XML Android

### 5.1.1.2.1 Traducción XAML a XML Android

A continuación, se compara que el resultado visual obtenido por los siguientes procesos son los mismos:

- Página de login en XML Android obtenido por generación a partir de DSL.
- Página de login en XML Android obtenido al realizar la traducción del fichero XAML obtenido mediante la generación a partir de DSL

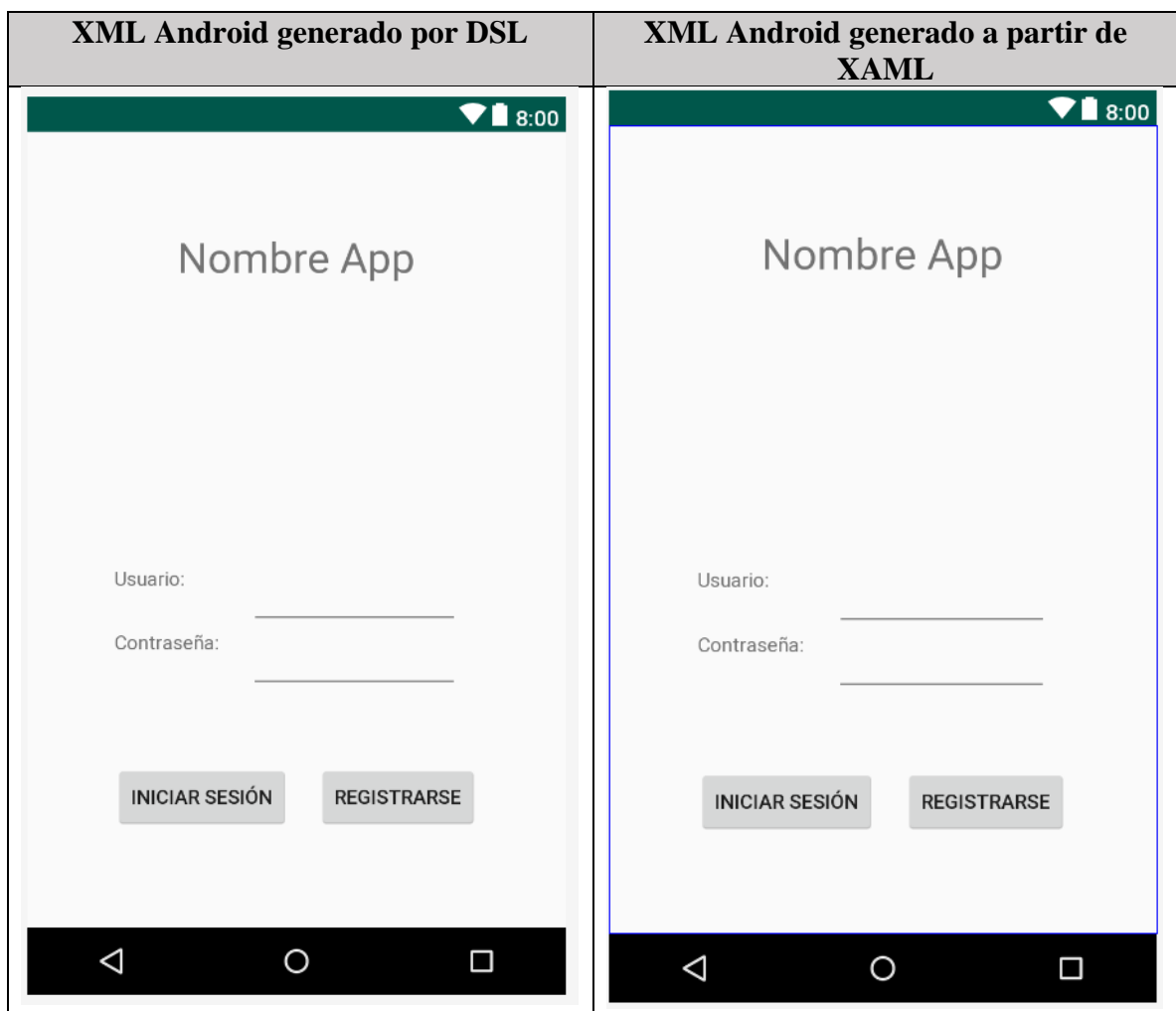


Figura 67 Página de Login XML Android generada como traducción de XAML

## 5.1.2 Página de Registro

En el uso de la aplicación, si el usuario no se encuentra registrado en la aplicación, en la pantalla de “Login” deberá pulsar sobre Registrar. Esta pantalla deberá ser un formulario que recoja los datos necesarios para ser almacenados en el sistema del usuario. El diseño de la pantalla cuenta con una serie de textos, cajas de textos, radioButtons y un botón para desencadenar la acción de dar de alta al usuario. Un boceto visual de la página buscada sería el siguiente:

Registro

Usuario:

Contraseña:

Repetir Contraseña:

Nombre:

Apellidos:

Correo:

Sexo:  M  H

Figura 68 Boceto de página Registro

### 5.1.2.1 Generación a partir de DSL

Para su definición DSL, es necesario ordenar los elementos en varias capas layout para lograr una distribución adecuada indicando de forma correcta la orientación de los elementos. Además, es necesario la utilización de atributos “margin” y “width” entre otros para lograr el resultado esperado.

Tras ejecutarse el prototipo, se comprueba en los IDEs de desarrollo Android Studio y Visual Basic el resultado visual de los ficheros generados. La distribución de los elementos guarda bastante parecido, aunque los estilos visuales cambien. Esto se debe a que se trata de código nativo para cada plataforma y cada una cuenta con estilos propios.

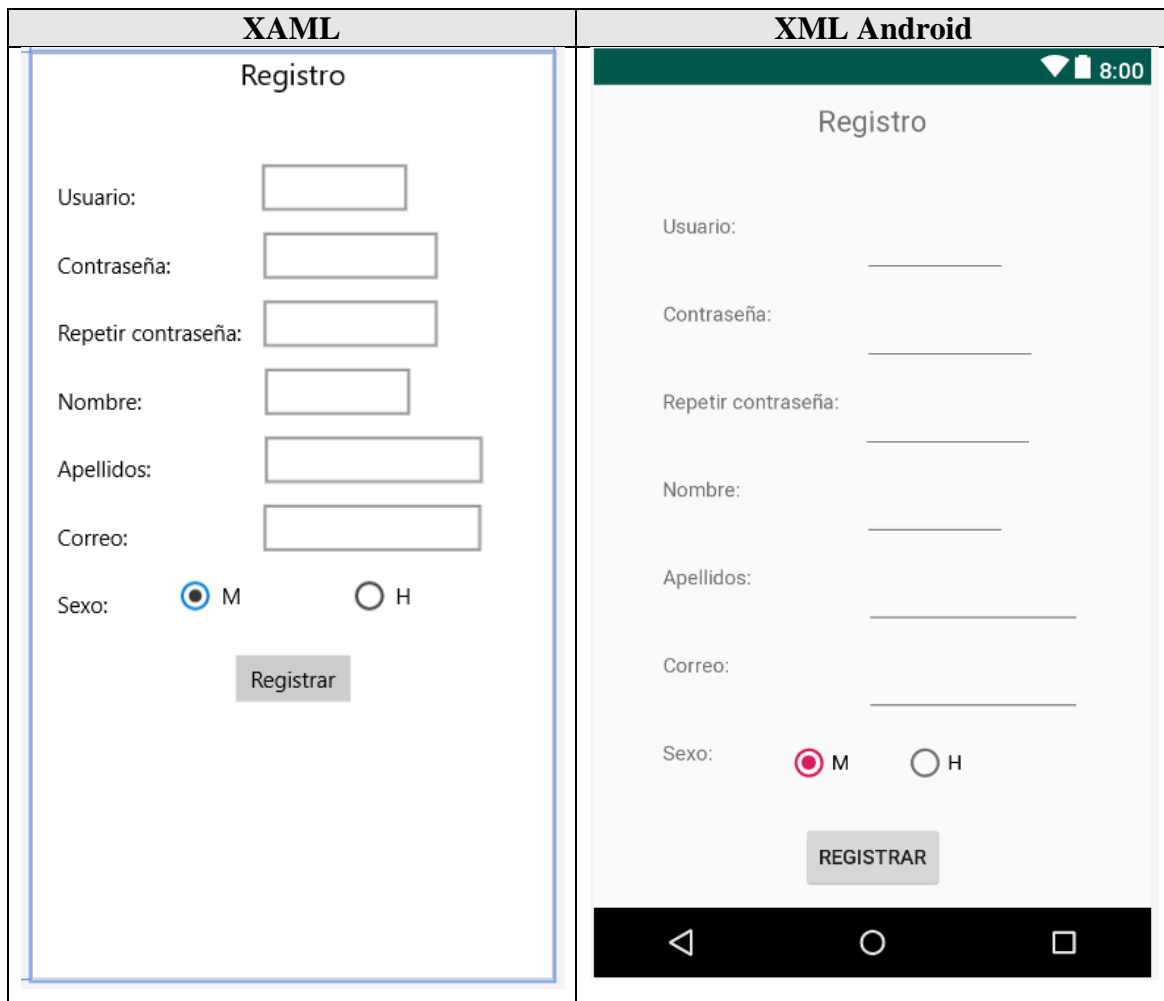


Figura 69 Páginas de Registro generadas a partir de DSL

### 5.1.2.2 Generación como traducción

Partiendo de los resultados obtenidos en el apartado anterior, se va a aplicar el prototipo sobre ellos para realizar traducciones entre los ficheros. Se comprobará que, visualmente, no hay variación en el resultado en comparación de la obtenida en generación de código a partir de un fichero en lenguaje DSL.

#### 5.1.2.2.1 Traducción XML Android a XAML

A continuación, se compara que el resultado visual obtenido por los siguientes procesos son los mismos:

- Página de registro en XAML obtenido por generación a partir de DSL.
- Página de registro en XAML obtenido al realizar la traducción del fichero XML Android obtenido mediante la generación a partir de DSL.

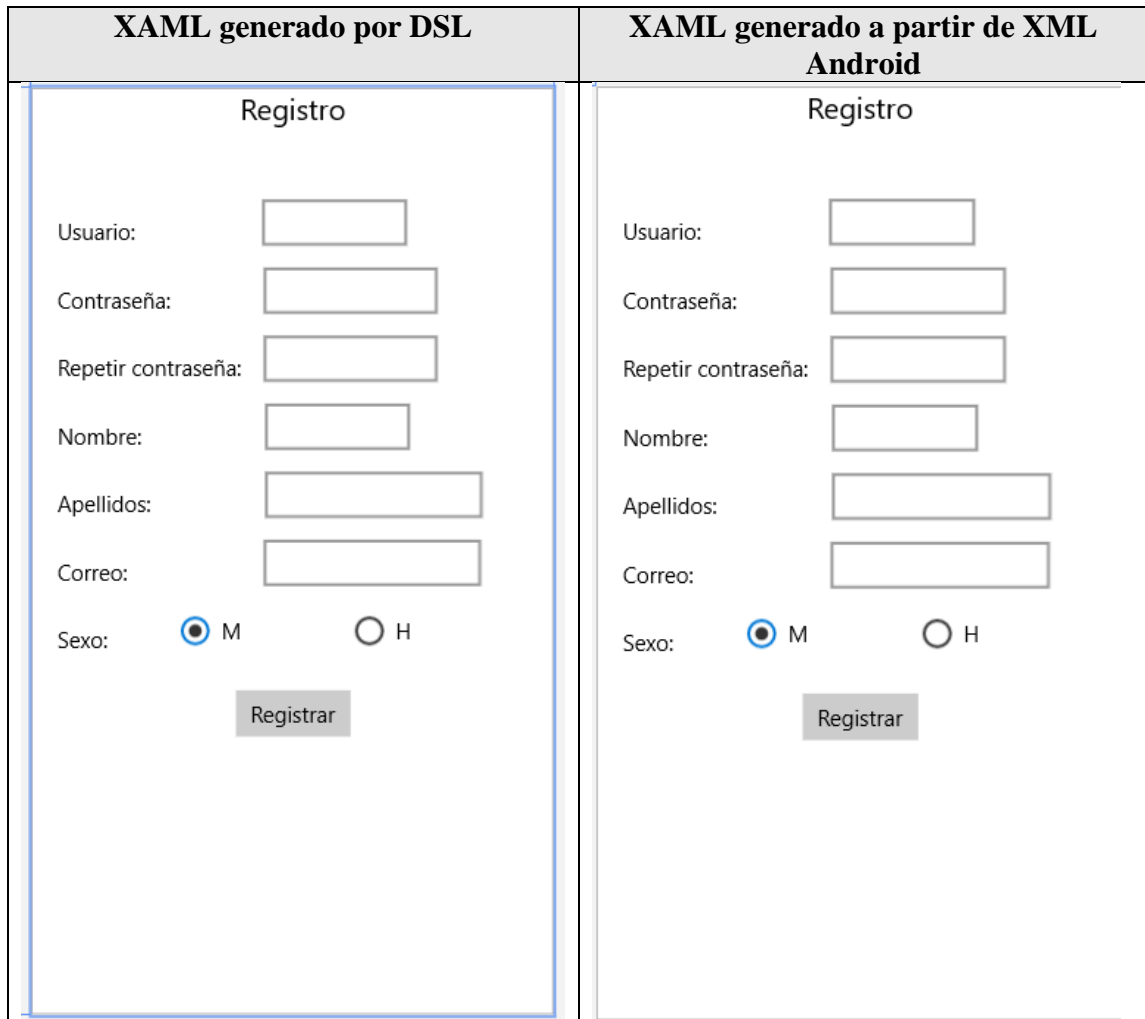


Figura 70 Página de Registro XAML generada como traducción de XML Android

#### 5.1.1.2.1 Traducción XAML a XML Android

A continuación, se compara que el resultado visual obtenido por los siguientes procesos son los mismos:

- Página de registro en XML Android obtenido por generación a partir de DSL.
- Página de registro en XML Android obtenido al realizar la traducción del fichero XAML obtenido mediante la generación a partir de DSL

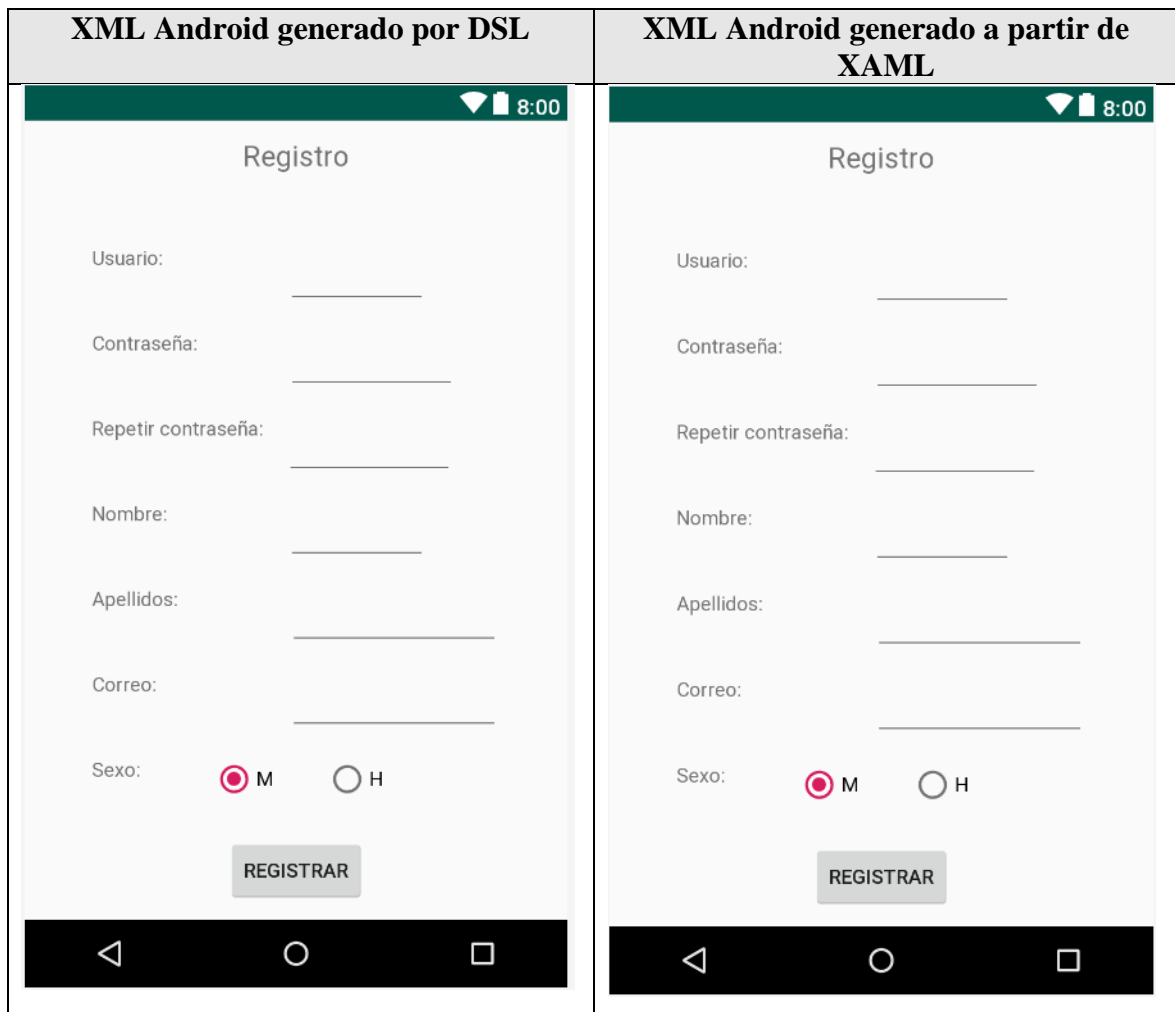


Figura 71 Página de Registro XML Android generada como traducción de XAML

## 5.2 Traducción de proyectos

En este apartado se va a proceder a utilizar el prototipo para demostrar la facilidad y productividad que puede aportar al aplicarlo para traducir el conjunto de ficheros visuales de un proyecto y generar otros ficheros en el lenguaje de otra plataforma. Esto puede ser útil para la migración de una aplicación ya finalizada a otra plataforma para ahorrar recursos en la generación de nuevos ficheros nativos de interfaz de usuario.

### 5.2.1 Traducción de Window a Android

Se va a proceder a traducir un proyecto Windows que contiene como interfaz de usuario dos páginas XAML detalladas de en el apartado “5.1 Generación de páginas”. Posteriormente, se comprobará que el resultado generado en el proyecto de salida

Android cumple con lo esperado. A continuación, se muestra de forma visual los resultados obtenidos tras utilizar el prototipo para realizar la traducción.


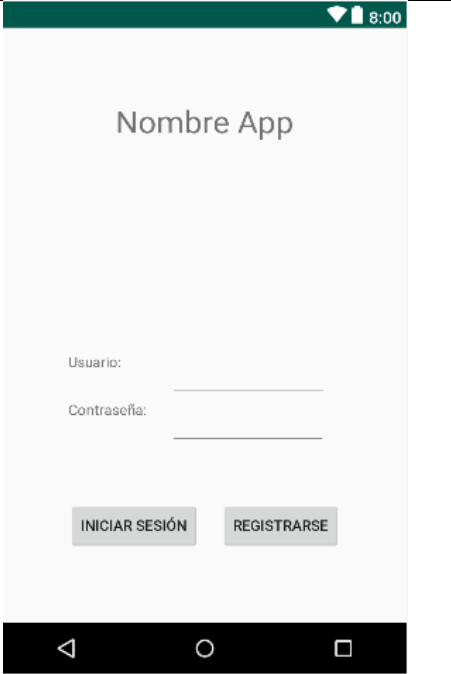
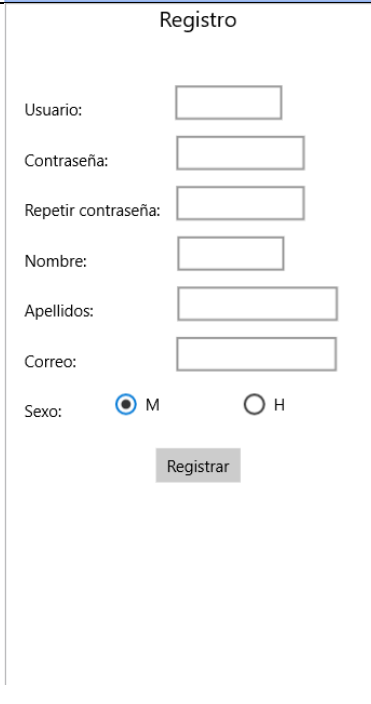
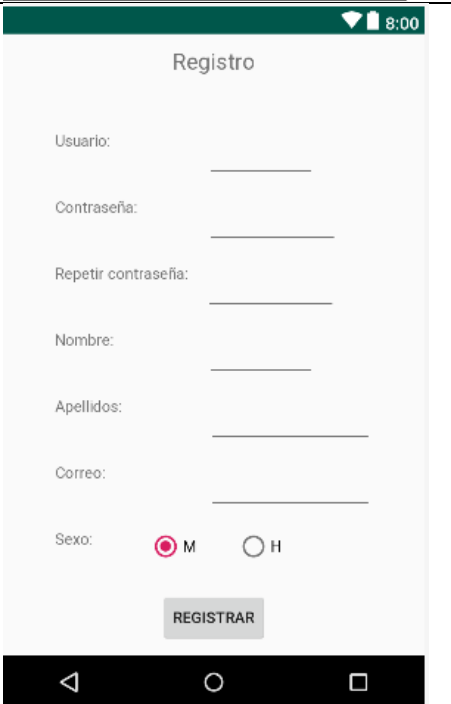
Pagina	Plataforma origen: Window	Plataforma destino: Android
<b>Login</b>		
<b>Registro</b>		

Figura 72 Traducción de proyecto Window a Android

Después de la ejecución del prototipo, se valida si el listado de ficheros XAML con elementos identificables para generar DSL por el prototipo del proyecto Window de entrada son generados en el proyecto de salida. Estos ficheros en este caso son los ficheros Login.xaml y Registro.xaml (figura 69).

HelloWorld > HelloWorld				
Nombre	Fecha de modifica...	Tipo	Tamaño	
Assets	01/12/2018 16:00	Carpeta de archivos		
bin	14/06/2019 23:53	Carpeta de archivos		
obj	23/12/2018 14:30	Carpeta de archivos		
Properties	01/12/2018 16:00	Carpeta de archivos		
App	01/12/2018 16:00	Archivo de marca...	1 KB	
App.xaml.cs	01/12/2018 16:00	Visual C# Source F...	5 KB	
HelloWorld	15/06/2019 0:14	Visual C# Project f...	8 KB	
HelloWorld_TemporaryKey	01/12/2018 16:00	Personal Informati...	3 KB	
Login	15/06/2019 0:36	Archivo de marca...	3 KB	
Login.xaml.cs	15/06/2019 0:13	Visual C# Source F...	1 KB	
Package.appxmanifest	01/12/2018 16:00	Archivo APPXMA...	2 KB	
Registro	15/06/2019 0:36	Archivo de marca...	5 KB	
Registro.xaml.cs	01/12/2018 20:10	Visual C# Source F...	1 KB	

Figura 73 Imagen de ficheros origen en la traducción de XML Android a XAML

Los ficheros son generados en la carpeta “app\src\main\res\layout”, localización de directorios donde se deben de encontrar los ficheros XML Android en los proyectos Android (figura 70).

HelloWorld > app > src > main > res > layout				
Nombre	Fecha de modifica...	Tipo	Tamaño	
MainPage	15/06/2019 0:39	Documento XML	1 KB	
Login	15/06/2019 0:36	Documento XML	4 KB	
Registro	15/06/2019 0:36	Documento XML	7 KB	
activity_main	10/06/2019 20:25	Documento XML	4 KB	

Figura 74 Imagen de ficheros generados en la traducción de XML Android a XAML

### 5.2.2 Traducción de Android a Window

Se va a proceder a traducir los ficheros con definición de interfaz de usuario de un proyecto Android a un proyecto Window. Las dos páginas que contendrá el proyecto de entrada serán las descritas en el apartado “5.1 Generación de páginas”. Se comprobará que el resultado generado en el proyecto indicado de salida es visualmente igual. A continuación, se muestra de forma comparativa los resultados obtenidos tras utilizar el prototipo para realizar la traducción.



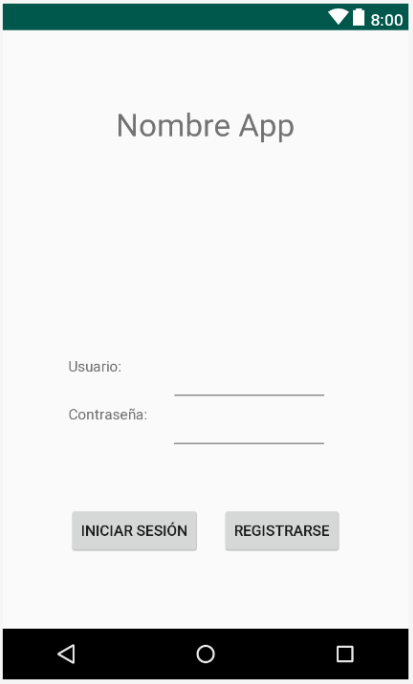


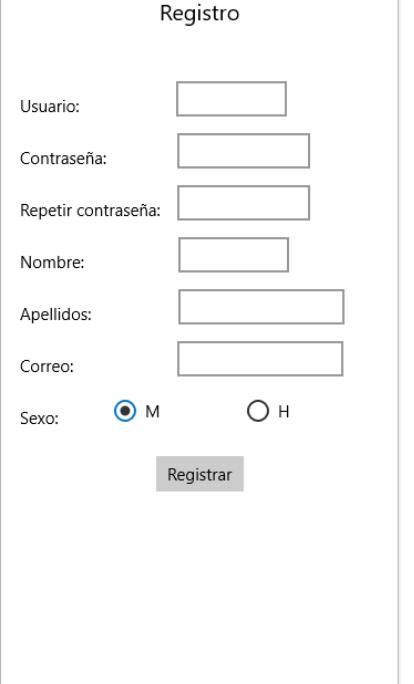
Pagina	Plataforma origen: Android	Plataforma destino: Window
Login		
Registro		

Figura 75 Traducción de proyecto Android a Window

Después de la ejecución del proceso de traducción, se puede validar si el listado de ficheros de la plataforma Android de entrada son generados en el proyecto de salida Window. En este ejemplo. Los ficheros han sido login.xml, registro.xml, MainPage.xml y activity\_main.xml (figura 72)

HelloWorld > app > src > main > res > layout

Nombre	Fecha de modifica...	Tipo	Tamaño
MainPage	15/06/2019 0:39	Documento XML	1 KB
registro	14/06/2019 19:37	Documento XML	7 KB
activity_main	10/06/2019 20:25	Documento XML	4 KB
login	08/06/2019 14:52	Documento XML	3 KB

Figura 76 Imagen de ficheros origen en la traducción de XAML a XML Android

Los ficheros son generados en la carpeta “HelloWorld”, donde se deben de encontrar los ficheros XAML para el correcto funcionamiento en las aplicaciones Windows (figura 73).

s > HelloWorld > HelloWorld

Nombre	Fecha de modifica...	Tipo	Tamaño
Assets	01/12/2018 16:00	Carpeta de archivos	
bin	14/06/2019 23:53	Carpeta de archivos	
obj	23/12/2018 14:30	Carpeta de archivos	
Properties	01/12/2018 16:00	Carpeta de archivos	
activity_main	15/06/2019 13:15	Archivo de marca...	3 KB
App	01/12/2018 16:00	Archivo de marca...	1 KB
App.xaml.cs	01/12/2018 16:00	Visual C# Source F...	5 KB
HelloWorld	15/06/2019 0:14	Visual C# Project f...	8 KB
HelloWorld_TemporaryKey	01/12/2018 16:00	Personal Informati...	3 KB
login	15/06/2019 13:15	Archivo de marca...	3 KB
MainPage	15/06/2019 13:15	Archivo de marca...	1 KB
Package.appxmanifest	01/12/2018 16:00	Archivo APPXMA...	2 KB
registro	15/06/2019 13:15	Archivo de marca...	4 KB

Figura 77 Imagen de ficheros generados en la traducción de XAML a XML Android

## 6 Conclusiones

Se ha podido demostrar que el prototipo desarrollado cumple la función que inicialmente se esperaba de él. El objetivo era que el prototipo sirviese de herramienta de apoyo y se pudiera integrar en el proceso de desarrollo software de una aplicación multiplataforma en dos etapas principales del desarrollo software. Por un lado, en el inicio del desarrollo para que el resultado final fuera una aplicación multiplataforma; y en una etapa final, en la que se desea migrar una aplicación desarrollada a otra plataforma.

Respecto a la integración del prototipo en el inicio del desarrollo, se ha conseguido generar código XAML y XML Android desde DSL. De esta forma, es posible destinar un desarrollo paralelo de la capa de interfaz de usuario mediante el DSL especificado de manera independiente a otros procesos de desarrollo.

Por otro lado, también es posible la aplicación del prototipo una vez finalizada la aplicación y al iniciar el proyecto de migración. Se ha demostrado que el prototipo desarrollado cuenta con la finalidad de recorrer un proyecto Windows o Android, encontrar los ficheros visuales y generar en otro proyecto los ficheros traducidos al lenguaje deseado.

El último objetivo logrado es la parametrización. Como se ha demostrado, la mayoría de los elementos y atributos se pueden traducir mediante la configuración de ficheros properties y el lenguaje DSL es totalmente parametrizable por el usuario. El resto de caso menores no parametrizables es posible cubrirlos mediante la expansión de la funcionalidad del prototipo mediante la sobrescritura de métodos gracias a la herencia de la programación orientada a objetos. Por ejemplo, en la figura 59 se muestra un ejemplo de la extensión de funcionalidad en nuevas clases que hereden de las clases del prototipo.

En función del desempeño realizado en el desarrollo del prototipo, se llega a la conclusión de que el desarrollo de un prototipo similar requiere de ser desarrollado en paralelo a las necesidades. La traducción entre lenguajes requiere de un conocimiento total de los lenguajes a traducir. Por ejemplo, ha sido necesario realizar muchas pruebas para encontrar errores y proceder a su resolución. La forma más sencilla de dar robustez a un prototipo de estas características es desarrollarlo junto al desarrollo de una aplicación multiplataforma. De esta forma, se irá corrigiendo y mejorando de forma más eficiente. Es muy probable que en la aplicación del prototipo desarrollado en un proyecto real se encuentren fallos no contemplados por falta de conocimiento técnico de los lenguajes a tratar.

Por otro lado, los resultados de la sencilla configuración realizada son bastante aceptables. Con la especificación realizada en los ficheros “.propertie” de una cantidad pequeña de elementos y atributos se han podido generar y traducir un par de páginas básicas de cualquier aplicación. Además, los elementos traducidos son los más utilizados en formularios de recogida de datos de un usuario. Esto nos lleva a la conclusión de que una herramienta que permita reconocer una cantidad mayor de elementos y atributos puede ser bastante productiva.

Asimismo, se puede facilitar más aún si cabe el diseño y generación de interfaces. El DSL puede ser generado por una herramienta con interfaz de usuario y herramientas de diseño. De esta forma, el diseño lo pueda hacer una persona sin conocimiento técnico de lenguajes de definición de interfaces y aplicar la salida de esta herramienta de diseño al prototipo desarrollado.

En conclusión, la tarea de realizar el aspecto visual de una herramienta multiplataforma puede volverse muy sencillo con la utilización de las herramientas correctas. Esta automatización también podría aplicarse a otros módulos, como la generación de un esquema de datos o validaciones de entidades. Procesos que, indiferentemente de la plataforma donde se ejecute una aplicación guardan un patrón de ejecución y desarrollo. El objetivo de estas herramientas debería ser automatizar trabajo que puede ser resultado con los patrones correctos y enfocar los esfuerzos en otras tareas que sí que necesiten un análisis, es decir, más hacia ámbito funcional de una aplicación.

## 7 Líneas futuras

Como se ha remarcado en toda la memoria, el software desarrollado se trata de un prototipo. Aunque se haya conseguido cumplir unos objetivos mínimos, la versión desarrollada es una buena base de la que partir y sobre la cual desarrollar mejoras. Entre ellas:

- Dar la posibilidad de externalizar los ficheros “.propertie” de configuración de las traducciones. En el prototipo desarrollado, los ficheros que contienen las especificaciones de las traducciones entre los distintos lenguajes se encuentran dentro del compilado “.jar”. Estos tipos de ficheros puede ser tratados como un directorio de ficheros con las herramientas adecuadas y acceder a su contenido, pudiendo modificar los ficheros “.propertie” de su interior. Aun así, es quizás un proceso incomodo y sería más cómodo para el usuario indicar al prototipo un directorio externo al prototipo que contenga los ficheros de configuración. Para lograr esto, sería necesario modificar las interfaces de los métodos y añadir como parámetro la ruta del directorio que contenga los ficheros.
- Tener en consideración más elementos “layouts” o capas para los procesos de traducción, es decir, tipos de elementos que contienen otros elementos.
- Profundizar en la búsqueda de procesos comunes y poder generalizar el prototipo para que sirva de traductor para cualquier lenguaje basado en XML. Esto sería posible debido a la naturaleza de Java y la orientación de objetos. Solo sería necesario crear clases que extiendan de las utilidades comunes del prototipo para desarrollar las traducciones necesarias para cada lenguaje.

## Referencias bibliográficas

1. H. Jack. "Code Generation in Action". Part 2: Code generation solutions.
2. <http://gs.statcounter.com/os-market-share/mobile/worldwide>. Mobile Operating System Market Share Worldwide. [Consulta Julio 2019]
3. <https://metacpan.org/pod/DSL::HTML>. Documentación de DSL::HTML. [Consulta Julio 2019]
4. <http://hackage.haskell.org/package/lucid>. Documentación de Lucid. [Consulta Julio 2019]
5. <http://www.perl6.org/>. Documentación de Perl. [Consulta Julio 2019]
6. <https://expertise.jetruby.com/android-anko-layouts-goodbye-xml-85b75936b26d>. Documentación de Anko. [Consulta Julio 2019]
7. <https://mockflow.com/>. Documentación de MockFlow. [Consulta Julio 2019]
8. <https://www.fluidui.com/>. Documentación de FluidUi. [Consulta Julio 2019]
9. <https://docs.microsoft.com/es-es/visualstudio/designers/creating-a-ui-by-using-xaml-designer-in-visual-studio?view=vs-2019>. Definición de interfaz de usuario en Visual Studio de forma interactiva. [Consulta Julio 2019]
10. <https://balsamiq.cloud/>. Documentación de Balsamiq web. [Consulta Julio 2019]
11. <https://glade.gnome.org/>. Documentación de Glade. [Consulta Julio 2019]
12. <https://balsamiq.com/wireframes/desktop/>. Documentación de Balsamiq como aplicación de escritorio. [Consulta Julio 2019]
13. <https://support.mockflow.com/article/84-how-to-export-my-wireframe-project>. Formatos de exportación en MockFlow. [Consulta Julio 2019]
14. <https://guide.fluidui.com/exporting-projects/>. Formatos de exportación en FluidUi. [Consulta Julio 2019]
15. <https://www.gtk.org/language-bindings.php>. Formatos de exportación en Glade. [Consulta Julio 2019]
16. <https://github.com/ccywch/UI2code>. Repositorio Git de UI2Code. [Consulta Julio 2019]
17. <https://hackernoon.com/introducing-ui2code-an-automatic-flutter-ui-code-generator-7e0a575c193>. Artículo sobre UI2Code. [Consulta Julio 2019]
18. <https://github.com/tonybeltramelli/pix2code>. Repositorio Git de Pix2Code. [Consulta Julio 2019]
19. <https://github.com/ngundotra/code2pix>. Repositorio Git de Code2pix. [Consulta Julio 2019]
20. <https://towardsdatascience.com/code2pix-deep-learning-compiler-for-graphical-user-interfaces-1256c346950b>. Artículo sobre code2pix. [Consulta Julio 2019]

21. <https://github.com/tonybeltramelli/pix2code/tree/master/compiler/assets>. Repositorio con la especificación de traducciones para Pix2Code. [Consulta Julio 2019]
22. <http://www.cshtml5.com/>. Documentación de CSHTML5. [Consulta Julio 2019]
23. <https://xaml2html.net/>. Documentación de xaml2html. [Consulta Julio 2019]
24. <https://docs.microsoft.com/es-es/xamarin/>. Documentación de Xamarin. [Consulta Julio 2019]
25. <https://flutter.dev/>. Documentación de Flutter. [Consulta Julio 2019]
26. <https://cordova.apache.org/>. Documentación de Apache Cordoba. [Consulta Julio 2019]
27. <https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html>. Documentación de Package org.w3c.dom. [Consulta Julio 2019]

## Siglas

GUI: graphical user interface (Interfaz Gráfica de Usuario)

DSL: Domain Specific Language (Lenguaje Específico de Dominio)

SQL: Structured Query Language (lenguaje de Consulta Estructurada)

IDE: Integrated Development Environment (Entorno de Desarrollo Integrado)

UI: User Interface (Interfaz de Usuario)

XAML: eXtensible Application Markup Language (Lenguaje Extensible de Formato para Aplicaciones)

XML: Extensible Markup Language (Lenguaje de Marcado Extensible)

WPF: Windows Presentation Foundation



# Anexos

## Manual de usuario

En el apartado actual se va a detallar como hacer uso del prototipo desarrollado. Debido a que el prototipo a utilizar es un fichero “.jar” es posible utilizarlo mediante dos vías. La primera de ellas es mediante un terminal de texto o una consola de comandos si se tiene instalado un intérprete de Java. La otra forma de uso a tratar en este apartado es como una librería que ofrece métodos públicos. Para poder utilizarla como librería es necesario asociarla a otro proyecto de desarrollo, ya sea mediante importación al proyecto o como dependencia.

Mediante consola de comando, es posible mostrar en el intérprete un resumen con todas las opciones disponibles. Para ello, basta con llamar al prototipo sin argumentos, o bien con un único parámetro con valor “0” o “Menu”.

```
C:\Users\Desktop>java -jar GeneradorXAMLXMLAndroid.jar
Menú
====
1.Generar fichero único mediante dsl: 1 // FicheroUnicoDesdeDSL
  Parámetros:
    1-Ruta fichero DSL
    2-Ruta salida
    3-Formato salida. Opciones:
    3-Formato salida. Opciones:
      -XAML: 1//XAML
      -XML ANDROID: 2//XMLAndroid
      -XAML y XML Android: 3//XAMLXMLAndroid
    4-(Opcional) Nombre de clase y fichero de salida
    5-(Opcional para XAML) Nombre de proyecto
2.Traducir fichero único XAML a XML Android: 2 // TraducirFicheroXAML
  Parámetros:
    1-Ruta fichero XAML
    2-Ruta salida
    3-(Opcional) Nombre de clase y fichero de salida
3.Traducir fichero único XML Android a XAML: 3 // TraducirFicheroXMLAndroid
  Parámetros:
    1-Ruta fichero XML Android
    2-Ruta salida
    3-(Opcional) Nombre de clase y fichero de salida
    5-(Opcional) Nombre de proyecto
4.Traducir proyecto Window a Android: 4 // TraducirProyectoWindow
  Parámetros:
    1-Ruta proyecto Window
    2-Ruta proyecto Android
5.Traducir proyecto Android a Window: 5 // TraducirProyectoAndroid
  Parámetros:
    1-Ruta proyecto Android
    2-Ruta proyecto Window
```

Figura 78 Imagen con el menú mostrado en consola de comandos

## Uso del prototipo para generación de páginas a partir de DSL

En este apartado se detallará cómo se ha de usar el prototipo desarrollado con el fin de generar ficheros XML Android o XAML a partir de un fichero DSL. Esto es posible realizarlo mediante una consola de comando o desde otro proyecto importando el fichero “.jar” como una librería o por algún otro mecanismo de dependencia.

### Mediante consola de comandos

Mediante la consola de comando es posible la generación de ficheros a partir de ficheros con código DSL. Para lograrlo, los parámetros introducidos al intérprete de java deberán ser:

1. Los valores “1” o “FicheroUnicoDesdeDSL”.
2. Ruta de directorio donde se encuentra el fichero DSL.
3. Ruta donde se generará el fichero de salida.
4. Opción de salida. Existen tres posibles valores:
  - a. Para generar un fichero XAML los valores pueden ser: “1” o “XAML”.
  - b. Para generar un fichero XML Android los valores pueden ser: “2” o “XMLAndroid”.
  - c. Para generar un fichero XAML y XML Android los valores pueden ser: “3” o “XAMLXMLAndroid”.
5. Opcional: Nombre de fichero de salida. Corresponderá al nombre de la clase asociada tanto en XAML como en XML Android. El valor por defecto en caso de no ser indicado es “clase”.
6. Opcional: Nombre de proyecto. Corresponderá al nombre del proyecto asociado en XAML. El valor por defecto en caso de no ser indicado es “proyecto”.

### Mediante uso como librería

La clase `GeneradorXAMLXMLAndroid` dispone de dos métodos públicos para utilizar la generación de ficheros a partir de código DSL:

- `generarFicheroXAMLdesdeFicheroDSL`: los parámetros necesarios son la ruta del fichero DSL, la ruta de directorio donde se quiere generar el fichero de salida en lenguaje XAML, el nombre de la clase (que en caso de pasarse un valor nulo será “clase”) y el nombre de proyecto asociado (que en caso de pasarse un valor nulo será “proyecto”).

- `generarFicheroXMLAndroidDesdeFicheroDSL`: los parámetros necesarios son la ruta del fichero DSL, la ruta de directorio donde se quiere que se genere el fichero de salida en lenguaje XML Android y el nombre de la clase (que en caso de pasarse un valor nulo será “clase”).

## Ejemplos

A continuación, se muestran dos ejemplos del uso principal del prototipo para generar ficheros de lenguaje final a partir de DSL. Los ejemplos serán detallados y complementados con imágenes. Además, se detallará tanto la utilización mediante consola de comandos como mediante una librería “.jar” de utilidades.

### Generación de ficheros XAML

Para realizar este proceso mediante una consola de comandos, al intérprete se le debe especificar los siguientes parámetros en el siguiente orden: la opción “1” como funcionalidad principal a utilizar; la ruta del directorio donde se encuentra el fichero DSL de entrada; la ruta de directorio donde se generarán los ficheros de salida; el formato de salida XAML, “login” como nombre de la clase asociada y fichero; y “miAplicacion” como nombre de la aplicación.

```
java -jar GeneradorXAMLXMLAndroid.jar 1 D:\Uned\Master\TFM\entradaDSL.xml D:\Uned\Master\TFM\ 1 login miAplicacion
```

Figura 79 Ejemplo de generación XAML mediante consola de comandos

Para lograr el mismo resultado importando el generador como una librería en un proyecto de desarrollo, es necesario invocar al siguiente método con los parámetros indicados:

```
String rutaEntradaDSL = "D:\\Uned\\Master\\TFM\\entradaDSL.xml";
String rutaSalida = "D:\\Uned\\Master\\TFM\\";
GeneradorXAMLXMLAndroid.generarFicheroXAMLdesdeFicheroDSL
(rutaEntradaDSL, rutaSalida, null, null);
```

Figura 80 Ejemplo de generación XAML mediante uso del prototipo como librería externa

## Generación de ficheros XAML y XML Android

Es posible generar un fichero XAML y XML Android a partir de un único fichero DSL mediante una consola de comando. Para ello, los parámetros necesarios son: la opción “1” como funcionalidad función principal a utilizar; la ruta del directorio donde se encuentra el fichero de entrada; la ruta de directorio donde se generarán los ficheros de salida y los formatos de salida: XAML y XML Android. Como se puede comprobar, tanto el nombre de la clase asociado como el nombre del proyecto no se detallan por lo que se generará con la opción por defecto.

```
java -jar GeneradorXAMLXMLAndroid.jar 1 D:\Uned\Master\TFM\entradaDSL.xml D:\Uned\Master\TFM\ 3
```

Figura 81 Ejemplo de generación XAML y XML Android mediante consola de comandos. Opciones numéricas

El resultado es el mismo que sustituir las opciones numéricas a las textuales.

```
java -jar GeneradorXAMLXMLAndroid.jar FicheroUnicoDesdeDSL D:\Uned\Master\TFM\entradaDSL.xml D:\Uned\Master\TFM\ XAMLXMLAndroid
```

Figura 82 Ejemplo de generación XAML y XML Android mediante consola de comandos. Opciones textuales

Para lograr el mismo resultado utilizando el generador como una librería, es necesario invocar a los siguientes métodos con los parámetros indicados:

```
String rutaEntradaDSL = "D:\\Uned\\Master\\TFM\\entradaDSL.xml";  
String rutaSalida = "D:\\Uned\\Master\\TFM\\";  
GeneradorXAMLXMLAndroid.generarFicheroXAMLdesdeFicheroDSL  
(rutaEntradaDSL, rutaSalida, null, null);  
GeneradorXAMLXMLAndroid.generarFicheroXMLAndroiddesdeFicheroDSL  
(rutaEntradaDSL, rutaSalida null);
```

Figura 83 Ejemplo de generación XAML y XML Android mediante uso del prototipo como librería externa

## Uso del prototipo para traducción de páginas

En este apartado se detallará cómo se ha de usar el prototipo desarrollado con el fin de traducir ficheros XML Android a ficheros XAML y viceversa. Este proceso es posible realizarlo mediante una consola de comando o desde otro proyecto importando el fichero “.jar” como una librería y utilizando sus métodos públicos.

## Mediante consola de comandos

Mediante una consola de comando es posible la traducción de ficheros a otros lenguajes. A continuación, se listan los parámetros a introducir según la traducción a realizar.

1. En función de la traducción a realizar, el valor del primer parámetro será:
  - a. Para traducir de XAML a XML Android, el primer parámetro deberá ser el valor “2” o “TraducirFicheroXAML”.
  - b. En caso de traducir de XML Android a XAML, el valor será “3” o “TraducirFicheroXMLAndroid”.
2. Ruta de directorio donde se encuentra el fichero de entrada.
3. Ruta donde se generará el fichero de salida.
4. Opcional: Nombre de fichero de salida. Corresponderá al nombre de la clase asociada tanto en XAML como en XML Android. El valor por defecto en caso de no ser indicado es “clase”.
5. Opcional: Nombre de proyecto. Corresponderá al nombre del proyecto asociado en XAML. El valor por defecto en caso de no ser indicado es “proyecto”.

## Mediante uso como librería

La clase `GeneradorXAMLXMLAndroid` dispone de dos métodos públicos para utilizar la traducción de ficheros a partir de un fichero de entrada:

- `generarXMLAndroidDesdeXAML`: los parámetros necesarios son la ruta del fichero XAML, la ruta de directorio donde se quiere que se genere el fichero XML Android de salida y el nombre de la clase (que en caso de pasarse un valor nulo será “clase”).
- `generarFicheroXAMLDesdeXMLAndroid`: los parámetros necesarios son la ruta del fichero XML Android, la ruta de directorio donde se quiere que se genere el fichero XAML, el nombre de la clase (que en caso de pasarse un valor nulo será “clase”) y el nombre de proyecto asociado (que en caso de pasarse un valor nulo será “proyecto”).

## Ejemplos

A continuación, se muestran dos ejemplos del uso del prototipo para realizar la traducción entre ficheros XAML y XML Android. Los ejemplos detallan tanto la utilización del prototipo mediante consola de comandos y como se debería de invocar a los métodos necesarios en caso de realizarse la importación de la librería en un proyecto de desarrollo.

## Traducir un fichero XAML a XML Android

Para realizar el proceso de traducción de un fichero XAML a XML Android mediante una consola de comandos, al intérprete se le debe especificar los siguientes parámetros en el siguiente orden: la opción 2 como funcionalidad principal a utilizar (o bien el valor “TraducirFicheroXAML”); la ruta del directorio donde se encuentra el fichero XAML de entrada; la ruta de directorio donde se generará el fichero de salida en formato XML Android; y “pagina” como nombre de la clase asociada.

```
java -jar GeneradorXAMLXMLAndroid.jar 2 D:\Uned\Master\TFM\entrada.xml D:\Uned\Master\TFM\ pagina
```

Figura 84 Ejemplo de traducción XAML a XML Android mediante consola de comandos. Opciones numéricas

```
java -jar GeneradorXAMLXMLAndroid.jar TraducirFicheroXAML D:\Uned\Master\TFM\entrada.xml D:\Uned\Master\TFM\ pagina
```

Figura 85 Ejemplo de traducción XAML a XML Android mediante consola de comandos. Opciones textuales

Este mismo resultado se puede conseguir utilizando el prototipo como librería importada en otro proyecto.

```
String rutaEntradaXAML = "D:\\Uned\\Master\\TFM\\entrada.xml";  
String rutaSalida = "D:\\Uned\\Master\\TFM\\";  
String nombreClaseSalida = "pagina";  
GeneradorXAMLXMLAndroid.generarXMLAndroidDesdeXAML(rutaEntradaXAML,  
rutaSalida, nombreClaseSalida);
```

Figura 86 Ejemplo de traducción XAML a XML Android mediante uso del prototipo como librería externa

## Traducir un fichero XML Android a XAML

Para realizar el proceso de traducción de un fichero XML Android a XAML mediante una consola de comandos, al intérprete se le debe especificar los siguientes parámetros en el siguiente orden de entrada: la opción 3 como funcionalidad principal a utilizar (o bien el valor “TraducirFicheroXMLAndroid”); la ruta del directorio donde se encuentra el fichero XML Android de entrada; la ruta de directorio donde se generará el fichero de salida en formato XML Android; “pagina” como nombre de la clase asociada; y “proyecto” como nombre del proyecto que contendrá el fichero XAML.

```
java -jar GeneradorXAMLXMLAndroid.jar 3 D:\Uned\Master\TFM\entrada.xml D:\Uned\Master\TFM\ pagina proyecto
```

Figura 87 Ejemplo de traducción XML Android a XAML mediante consola de comandos. Opciones numéricas

```
java -jar GeneradorXAMLXMLAndroid.jar TraducirFicheroXMLAndroid D:\Uned\Master\TFM\entrada.xml D:\Uned\Master\TFM\ pagina proyecto
```

Figura 88 Ejemplo de traducción XML Android a XAML mediante consola de comandos. Opciones textuales

Este mismo resultado se puede conseguir al ejecutar el siguiente código importando el prototipo como librería en otro proyecto.

```
String rutaEntradaXAML = "D:\\Uned\\Master\\TFM\\entrada.xaml";  
String rutaSalida = "D:\\Uned\\Master\\TFM\\";  
String nombreClaseSalida = "pagina";  
String nombreProyecto = "proyecto";  
GeneradorXAMLXMLAndroid.generarFicheroXAMLDesdeXMLAndroid  
(rutaEntradaXAML, rutaSalida, nombreClaseSalida, nombreProyecto);
```

Figura 89 Ejemplo de traducción XML Android a XAML mediante uso del prototipo como librería externa

## Uso del prototipo para traducción de proyectos

En este apartado se explicará cómo se ha de usar el prototipo desarrollado con el fin de traducir todos los ficheros nativos de definición de interfaz de usuario de un proyecto Android a Window y viceversa. Es decir, la traducción de los ficheros XAML que pueda contener un proyecto de plataforma Window a XML Android; o bien, el proceso contrario. La entrada principal necesaria de este proceso son las rutas raíz de directorio donde se encuentran ambos proyectos.

### Mediante consola de comandos

Mediante una consola de comando es posible realizar la traducción de los ficheros con IU que pueda contener un proyecto de una plataforma a otra plataforma distinta. A continuación, se plantea los parámetros a utilizar según el tipo de proyecto fuente:

1. En función de la traducción de plataforma a realizar, el valor del primer parámetro será:
  - a. Para traducir un proyecto de la plataforma Window a Android, el primer parámetro deberá ser el valor “4” o “TraducirProyectoWindow”.
  - b. En caso de traducir un proyecto de la plataforma Android a Window, el primer parámetro deberá ser el valor “5” o “TraducirProyectoAndroid”.
2. Ruta de directorio donde se encuentra el proyecto de la plataforma de entrada.
3. Ruta de directorio se generarán los ficheros en el lenguaje de la plataforma de salida.

## Mediante uso como librería

La clase `GeneradorXAMLXMLAndroid` dispone de dos métodos públicos para realizar la traducción de ficheros visuales de una plataforma al lenguaje de otra plataforma:

- `transformarWindowAndroid`: los parámetros necesarios son la ruta del proyecto de la plataforma Window y la ruta de directorio de la plataforma Android donde se generarán los ficheros traducidos.
- `transformaAndroidWindow`: los parámetros necesarios son la ruta del proyecto de la plataforma Android y la ruta de directorio de la plataforma Window donde se generarán los ficheros traducidos.

## Ejemplos

A continuación, se muestran dos ejemplos del uso del prototipo para realizar la traducción entre proyectos entre distintas plataformas.

### Traducir un proyecto de plataforma Window a Android

Mediante consola de comandos, para realizar el proceso de traducción de los ficheros visuales de un proyecto de la plataforma Window a ficheros XML Android y que ocupen el lugar correcto en un proyecto Android se debe especificar los siguientes parámetros en el siguiente orden: la opción 4 como funcionalidad principal a utilizar (o bien el valor “TraducirProyectoWindow”); la ruta del directorio donde se encuentra el proyecto de entrada Window; y la ruta del directorio donde se encuentra el proyecto de la plataforma Android donde se desea que se generen los ficheros XML Android.



```
java -jar GeneradorXAMLXMLAndroid.jar 4 D:\Uned\Master\Window\HelloWorld D:\Uned\Master\Android\HelloWorld
```

Figura 90 Ejemplo de traducción de proyecto Window a Android mediante consola de comandos. Opciones numéricas

```
java -jar GeneradorXAMLXMLAndroid.jar TraducirProyectoWindow D:\Uned\Master\Window\HelloWorld D:\Uned\Master\Android\HelloWorld
```

Figura 91 Ejemplo de traducción de proyecto Window a Android mediante consola de comandos. Opciones textuales

Este mismo resultado se puede conseguir utilizando el prototipo como librería importada en otro proyecto

```
String rutaEntradaWindow = "D:\\Uned\\Master\\Window\\HelloWorld";  
String rutaSalidaAndroid = "D:\\Uned\\Master\\Android\\HelloWorld";  
GeneradorXAMLXMLAndroid.transformarWindowAndroid(rutaEntradaWindow,  
rutaSalidaAndroid);
```

Figura 92 Ejemplo de traducción de proyecto Window a Android mediante uso del prototipo como librería externa

## Traducir un proyecto de plataforma Android a Window

Mediante consola de comandos, es posible realizar el proceso de traducción de los ficheros XML Android de un proyecto de la plataforma Android a ficheros XAML para aplicaciones Windows. Para realizar este proceso, y que los ficheros generados ocupen el lugar indicado en un proyecto, se debe especificar los siguientes parámetros en el siguiente orden: la opción 5 como funcionalidad principal a utilizar (o bien el valor “TraducirProyectoAndroid”); la ruta del directorio donde se encuentra el proyecto de entrada Android; y la ruta del directorio donde se encuentra el proyecto de la plataforma Window donde se desea que se generarán los ficheros XAML.

```
java -jar GeneradorXAMLXMLAndroid.jar 5 D:\Uned\Master\Android\HelloWorld D:\Uned\Master\Window\HelloWorld
```

Figura 93 Ejemplo de traducción de proyecto Android a Window mediante consola de comandos. Opciones numéricas

```
java -jar GeneradorXAMLXMLAndroid.jar TraducirProyectoAndroid D:\Uned\Master\Android\HelloWorld D:\Uned\Master\Window\HelloWorld
```

Figura 94 Ejemplo de traducción de proyecto Android a Window mediante consola de comandos. Opciones textuales

Este mismo resultado se puede conseguir utilizando el prototipo como librería importada en otro proyecto.

```
String rutaEntradaAndroid = "D:\ \Uned\ \Master\ \Android\ \HelloWorld";  
String rutaSalidaWindow = "D:\ \Uned\ \Master\ \Window\ \HelloWorld";  
GeneradorXAMLXMLAndroid.transformaAndroidWindow (rutaEntradaAndroid,  
rutaSalidaWindow);
```

*Figura 95 Ejemplo de traducción de proyecto Android a Window mediante uso del prototipo como librería externa*

## Templates

Los dos siguientes apartados contienen el contenido de los templates que sirven como base para la generación de ficheros en el prototipo desarrollado.

### templateXAMLVisual.template

El siguiente código muestra el template a partir del cual el prototipo genera los ficheros en lenguaje XAML. El contenido generado por la herramienta sustituirá al texto “contenido\_generado”. Otras variables que también serán sustituidas son “nombre\_proyecto” y “nombre\_clase”. Este contenido es modificable externamente, permitiendo ser útil en caso de querer cambiarse la referencia a las librerías XAML por nuevas versiones.

```
<?xml version="1.0" encoding="utf-8"?>
<Page
  x:Class="nombre_proyecto.nombre_clase"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:nombre_proyecto"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

contenido_generado

</Page>
```

### templateAndroidVisual.template

El siguiente código muestra el template a partir del cual el prototipo genera los ficheros en lenguaje XML Android. En comparación al template de XAML, no cuenta más que con la especificación de que se tratará de un fichero que cumplirá con el estándar de XML.

```
<?xml version="1.0" encoding="utf-8"?>
contenido_generado
```

## Ejemplos de generación de páginas

En los siguientes apartados se va a mostrar el código de entrada y el código generado en el apartado “5.1 Generación de páginas”.

### Generación de página Login

La página de Login descrita en el apartado “5.1.1 Página de Login” se puede generar con el prototipo a partir de DSL o bien mediante la traducción de otro fichero nativo de definición de interfaz de usuario. En los siguientes apartados, se mostrarán tanto el contenido de entrada al prototipo como el resultado obtenido en los distintos procesos.

#### Entrada en lenguaje DSL

La entrada en formato DSL al prototipo para la generación de ficheros en lenguaje XAML y XML Android es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
  <layout HorizontalAlignment="Center" VerticalAlignment="Center" orientation="V">
    <layout HorizontalAlignment="Center" VerticalAlignment="Center" orientation="H">
      <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="200"
textContent="Nombre App" size="30" />
    </layout>
    <layout HorizontalAlignment="Left" VerticalAlignment="Bottom" orientation="V">
      <layout HorizontalAlignment="Left" VerticalAlignment="Bottom" orientation="H">
        <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginRight="46"
textContent="Usuario:" />
        <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="150" />
      </layout>
      <layout HorizontalAlignment="Left" VerticalAlignment="Bottom" orientation="H">
        <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginRight="21"
textContent="Contraseña:" />
        <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="150" />
      </layout>
    </layout>
    <layout HorizontalAlignment="Center" VerticalAlignment="Center" marginUp="50"
orientation="H">
      <button HorizontalAlignment="Left" VerticalAlignment="Top" textContent="Iniciar
Sesión" />
      <button HorizontalAlignment="Left" VerticalAlignment="Top" marginLeft="20"
textContent="Registrarse" />
    </layout>
  </layout>
```

## Generación a partir de DSL

Los resultados obtenidos al realizar la generación de la página Login mediante DSL con el prototipo son los contenidos en los siguientes apartados.

### Fichero XAML de página de Login desde DSL

```
<?xml version="1.0" encoding="utf-8"?>
<Page
  x:Class="proyecto.login"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:proyecto"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

  <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
    HorizontalAlignment="Center" >
    <StackPanel Orientation="Vertical" VerticalAlignment="Center"
      HorizontalAlignment="Center" >
      <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
        HorizontalAlignment="Center" >
        <TextBlock FontSize="30" VerticalAlignment="Bottom" Text="Nombre App"
          HorizontalAlignment="Left" Margin="0,0,0,200" />
        </StackPanel>
        <StackPanel Orientation="Vertical" VerticalAlignment="Bottom"
          HorizontalAlignment="Left" >
          <StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" >
            <TextBlock VerticalAlignment="Bottom" Text="Usuario:" HorizontalAlignment="Left"
              Margin="0,0,46,0" />
            <TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
          </StackPanel>
          <StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" >
            <TextBlock VerticalAlignment="Bottom" Text="Contraseña:"
              HorizontalAlignment="Left" Margin="0,0,21,0" />
            <TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
          </StackPanel>
        </StackPanel>
      </StackPanel>
      <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
        HorizontalAlignment="Center" Margin="0,50,0,0" >
        <Button VerticalAlignment="Top" Content="Iniciar Sesión" HorizontalAlignment="Left"
          />
        <Button VerticalAlignment="Top" Content="Registrarse" HorizontalAlignment="Left"
          />
      </StackPanel>
    </StackPanel>
  </Page>
```

```
Margin="20,0,0,0" />
</StackPanel>
</StackPanel>
</StackPanel>
</Page>
```

Fichero XML Android de página de Login desde DSL

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical" android:layout_gravity="center"
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=".login"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:textSize="30" android:layout_marginBottom="200"
android:layout_gravity="bottom" android:text="Nombre App"
android:layout_width="match_parent" android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="vertical" android:layout_gravity="bottom"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_gravity="bottom"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="46" android:layout_gravity="bottom"
android:text="Usuario:" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_gravity="bottom"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="21" android:layout_gravity="bottom"
android:text="Contraseña:" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginTop="50"
android:layout_gravity="center" android:layout_width="wrap_content"
android:layout_height="wrap_content" >
<Button android:text="Iniciar Sesión" android:layout_width="match_parent"
android:layout_height="match_parent" />
<Button android:text="Registrarse" android:layout_marginLeft="20"
android:layout_width="match_parent" android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
```

## Generación a partir de traducción

Los resultados obtenidos al realizar la generación de la página Login mediante el proceso de traducción con el prototipo se encuentran en los siguientes apartados.

### Traducción XML Android a XAML

```
<?xml version="1.0" encoding="utf-8"?>
<Page
  x:Class="HelloWorld.login"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:HelloWorld"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

  <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
    HorizontalAlignment="Center" >
    <StackPanel Orientation="Vertical" VerticalAlignment="Center"
      HorizontalAlignment="Center" >
      <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
        HorizontalAlignment="Center" >
        <TextBlock FontSize="30" VerticalAlignment="Bottom" Text="Nombre App"
          HorizontalAlignment="Left" Margin="0,0,0,200" />
        </StackPanel>
        <StackPanel Orientation="Vertical" VerticalAlignment="Bottom"
          HorizontalAlignment="Left" >
          <StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" >
            <TextBlock VerticalAlignment="Bottom" Text="Usuario:" HorizontalAlignment="Left"
              Margin="0,0,46,0" />
            <TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
          </StackPanel>
          <StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
            HorizontalAlignment="Left" >
            <TextBlock VerticalAlignment="Bottom" Text="Contraseña:" HorizontalAlignment="Left"
              Margin="0,0,21,0" />
            <TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
          </StackPanel>
        </StackPanel>
      </StackPanel>
    </StackPanel>
  </StackPanel>
  <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
    HorizontalAlignment="Center" Margin="0,50,0,0" >
    <Button VerticalAlignment="Top" Content="Iniciar Sesión" HorizontalAlignment="Left" />
    <Button VerticalAlignment="Top" Content="Registrarse" HorizontalAlignment="Left"
      Margin="20,0,0,0" />
  </StackPanel>
</Page>
```

```
</StackPanel>
</StackPanel>
</StackPanel>
</Page>
```

### Traducción XAML a XML Android

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="horizontal" android:layout_gravity="center"
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=".login"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="vertical" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="0" android:textSize="30"
android:layout_marginBottom="200" android:layout_marginTop="0"
android:layout_gravity="bottom" android:text="Nombre App"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="vertical" android:layout_gravity="bottom"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_gravity="bottom"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="46" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Usuario:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_gravity="bottom"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="21" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom"
android:text="Contraseña:" android:layout_marginLeft="0"
android:layout_width="match_parent" android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="0" android:layout_marginTop="50"
android:layout_gravity="center" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<Button android:text="Iniciar Sesión" android:layout_width="match_parent"
android:layout_height="match_parent" />
```



```

<Button android:layout_marginRight="0" android:layout_marginBottom="0"
android:layout_marginTop="0" android:text="Registrarse" android:layout_marginLeft="20"
android:layout_width="match_parent" android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
</LinearLayout>

```

## Generación de página Registro

La página de Registro descrita en el apartado “5.1.2 Página de Registros” se puede generar con el prototipo mediante DSL o bien mediante la traducción de otro fichero nativo de definición de interfaz de usuario. En los siguientes apartados, se mostrará tanto el contenido de entrada al prototipo como el resultado obtenido en los distintos procesos.

### Entrada en lenguaje DSL

La entrada en formato DSL al prototipo para la generación de ficheros en lenguaje XAML y XML Android es el siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<layout HorizontalAlignment="Stretch" VerticalAlignment="Stretch" orientation="V">
  <layout HorizontalAlignment="Center" VerticalAlignment="Center" orientation="H">
    <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="50"
marginLeft="0" marginRight="0" marginUp="0" size="20" textContent="Registro" />
  </layout>
  <layout HorizontalAlignment="Center" VerticalAlignment="Bottom" orientation="V">
    <layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
      <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="87" marginUp="0" textContent="Usuario:" />
      <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="100" />
    </layout>
    <layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
      <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="63" marginUp="0" textContent="Contraseña:" />
      <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="120" />
    </layout>
    <layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
      <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="15" marginUp="0" textContent="Repetir contraseña:" />
      <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="120" />
    </layout>
  </layout>
</layout>

```

```

</layout>
<layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
  <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="84" marginUp="0" textContent="Nombre:" />
  <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="100" />
</layout>
<layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
  <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="78" marginUp="0" textContent="Apellidos:" />
  <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="150" />
</layout>
<layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
  <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="93" marginUp="0" textContent="Correo:" />
  <textBox HorizontalAlignment="Left" VerticalAlignment="Bottom" width="150" />
</layout>
<layout HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="15"
marginLeft="0" marginRight="0" marginUp="0" orientation="H">
  <text HorizontalAlignment="Left" VerticalAlignment="Bottom" marginDown="0"
marginLeft="0" marginRight="50" marginUp="0" textContent="Sexo:" />
  <radioButton checked="true" textContent="M" width="80" />
  <radioButton checked="false" textContent="H" width="80" />
</layout>
</layout>
<layout HorizontalAlignment="Center" VerticalAlignment="Center" marginDown="0"
marginLeft="0" marginRight="0" marginUp="10" orientation="H">
  <button HorizontalAlignment="Left" VerticalAlignment="Top" textContent="Registrar" />
</layout>
</layout>

```

## Generación a partir de DSL

Los resultados obtenidos al realizar la generación de la página Login mediante DSL a través del prototipo son los contenidos en los siguientes apartados.

## Fichero XAML de página de Registro desde DSL

```

<?xml version="1.0" encoding="utf-8"?>
<Page
  x:Class="proyecto.registro"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:proyecto"

```

```

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<StackPanel Orientation="Vertical" VerticalAlignment="Stretch"
HorizontalAlignment="Stretch" >
<StackPanel Orientation="Horizontal" VerticalAlignment="Center"
HorizontalAlignment="Center" >
<TextBlock FontSize="20" VerticalAlignment="Bottom" Text="Registro"
HorizontalAlignment="Left" Margin="0,0,0,50" />
</StackPanel>
<StackPanel Orientation="Vertical" VerticalAlignment="Bottom"
HorizontalAlignment="Center" >
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Usuario:" HorizontalAlignment="Left"
Margin="0,0,87,0" />
<TextBox Width="100" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Contraseña:" HorizontalAlignment="Left"
Margin="0,0,63,0" />
<TextBox Width="120" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Repetir contraseña:"
HorizontalAlignment="Left" Margin="0,0,15,0" />
<TextBox Width="120" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Nombre:" HorizontalAlignment="Left"
Margin="0,0,84,0" />
<TextBox Width="100" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Apellidos:" HorizontalAlignment="Left"
Margin="0,0,78,0" />
<TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Correo:" HorizontalAlignment="Left"
Margin="0,0,93,0" />
<TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>

```

```

<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Sexo:" HorizontalAlignment="Left"
Margin="0,0,50,0" />
<RadioButton Width="80" IsChecked="True" Content="M" />
<RadioButton Width="80" IsChecked="False" Content="H" />
</StackPanel>
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Center"
HorizontalAlignment="Center" Margin="0,10,0,0" >
<Button VerticalAlignment="Top" Content="Registrar" HorizontalAlignment="Left" />
</StackPanel>
</StackPanel>
</Page>

```

Fichero XML Android de página de Registro desde DSL

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical" android:layout_gravity="center"
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=".registro"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="0" android:textSize="20"
android:layout_marginBottom="50" android:layout_marginTop="0"
android:layout_gravity="bottom" android:text="Registro" android:layout_marginLeft="0"
android:layout_width="match_parent" android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="vertical" android:layout_width="wrap_content"
android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="87" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Usuario:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="100" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="63" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Contraseña:"

```

```

android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="120" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="15" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Repetir
contraseña:" android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="120" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="84" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Nombre:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="100" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="78" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Apellidos:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="93" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Correo:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"

```

```

android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="50" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Sexo:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<RadioButton android:layout_width="80" android:checked="true" android:text="M"
android:layout_height="match_parent" />
<RadioButton android:layout_width="80" android:checked="false" android:text="H"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="0" android:layout_marginTop="10"
android:layout_gravity="center" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<Button android:text="Registrar" android:layout_width="match_parent"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>

```

## Generación a partir de traducción

Los resultados obtenidos al realizar la generación de la página Registro mediante el proceso de traducción con el prototipo se pueden encontrar en los siguientes apartados.

## Traducción XML Android a XAML

```

<?xml version="1.0" encoding="utf-8"?>
<Page
  x:Class="HelloWorld.registro"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:HelloWorld"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  Background="{ ThemeResource ApplicationPageBackgroundThemeBrush }">

  <StackPanel Orientation="Vertical" VerticalAlignment="Center"
  HorizontalAlignment="Center" >
  <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
  HorizontalAlignment="Center" >

```

```

<TextBlock FontSize="20" VerticalAlignment="Bottom" Text="Registro"
HorizontalAlignment="Left" Margin="0,0,0,50" />
</StackPanel>
<StackPanel Orientation="Vertical" VerticalAlignment="Top" HorizontalAlignment="Left" >
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Usuario:" HorizontalAlignment="Left"
Margin="0,0,87,0" />
<TextBox Width="100" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Contraseña:" HorizontalAlignment="Left"
Margin="0,0,63,0" />
<TextBox Width="120" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Repetir contraseña:"
HorizontalAlignment="Left" Margin="0,0,15,0" />
<TextBox Width="120" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Nombre:" HorizontalAlignment="Left"
Margin="0,0,84,0" />
<TextBox Width="100" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Apellidos:" HorizontalAlignment="Left"
Margin="0,0,78,0" />
<TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Correo:" HorizontalAlignment="Left"
Margin="0,0,93,0" />
<TextBox Width="150" VerticalAlignment="Bottom" HorizontalAlignment="Left" />
</StackPanel>
<StackPanel Orientation="Horizontal" VerticalAlignment="Bottom"
HorizontalAlignment="Left" Margin="0,0,0,15" >
<TextBlock VerticalAlignment="Bottom" Text="Sexo:" HorizontalAlignment="Left"
Margin="0,0,50,0" />
<RadioButton Width="80" IsChecked="True" VerticalAlignment="Top" Content="M"
HorizontalAlignment="Left" />
<RadioButton Width="80" IsChecked="False" VerticalAlignment="Top" Content="H"
HorizontalAlignment="Left" />
</StackPanel>
</StackPanel>

```

```

<StackPanel Orientation="Horizontal" VerticalAlignment="Center"
HorizontalAlignment="Center" Margin="0,10,0,0" >
<Button VerticalAlignment="Top" Content="Registrar" HorizontalAlignment="Left" />
</StackPanel>
</StackPanel>

</Page>

```

## Traducción XAML a XML Android

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical" android:layout_gravity="center"
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools" tools:context=".registro"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="0" android:textSize="20"
android:layout_marginBottom="50" android:layout_marginTop="0"
android:layout_gravity="bottom" android:text="Registro" android:layout_marginLeft="0"
android:layout_width="match_parent" android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="vertical" android:layout_width="wrap_content"
android:layout_height="wrap_content" >
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="87" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Usuario:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="100" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="63" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom"
android:text="Contraseña:" android:layout_marginLeft="0"
android:layout_width="match_parent" android:layout_height="match_parent" />
<EditText android:layout_width="120" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>

```



```

<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="15" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Repetir
contraseña:" android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="120" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="84" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Nombre:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="100" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="78" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Apellidos:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="93" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Correo:"
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<EditText android:layout_width="150" android:layout_gravity="bottom"
android:layout_height="match_parent" />
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="15" android:layout_marginTop="0"
android:layout_gravity="bottom" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<TextView android:layout_marginRight="50" android:layout_marginBottom="0"
android:layout_marginTop="0" android:layout_gravity="bottom" android:text="Sexo:"

```

```
android:layout_marginLeft="0" android:layout_width="match_parent"
android:layout_height="match_parent" />
<RadioButton android:layout_width="80" android:checked="true" android:text="M"
android:layout_height="match_parent" />
<RadioButton android:layout_width="80" android:checked="false" android:text="H"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
<LinearLayout android:orientation="horizontal" android:layout_marginRight="0"
android:layout_marginBottom="0" android:layout_marginTop="10"
android:layout_gravity="center" android:layout_marginLeft="0"
android:layout_width="wrap_content" android:layout_height="wrap_content" >
<Button android:text="Registrar" android:layout_width="match_parent"
android:layout_height="match_parent" />
</LinearLayout>
</LinearLayout>
```