Universidad Zaragoza

1542

Trabajo Fin de Grado

Desarrollo de una aplicación móvil multiplataforma para la creación de documentos XML utilizando XML Schema y Schematron

Autor

Sergio Frago Criado

Director

Álvaro Alesanco Iglesias

Universidad de Zaragoza / Escuela de Ingeniería y Arquitectura
2015

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Sergio Frago Criado,

con nº de DNI 17767224T en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Desarrollo de una aplicación móvil multiplataforma para la creación de documentos XML utilizando XML Schema y Schematron,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 13 de febrero de 2015

Fdo:

SERGIO FRAGO CRIADO

A mis padres y al resto de personas que me han ayudado a hacer realidad este proyecto.

Agradecimientos

Quiero dar las gracias Álvaro Alesanco por su ayuda, por su paciencia y sobre todo por darme la oportunidad de trabajar en un proyecto tan interesante. También quiero mostrar mi agradecimiento a todas aquellas personas que me han mostrado su apoyo y me han animado a completar este proyecto, especialmente a mi padre.

Desarrollo de una aplicación móvil multiplataforma para la creación de documentos XML utilizando XML Schema y Schematron

Resumen

Este trabajo plantea el diseño y la implementación de una aplicación móvil multiplataforma mediante JavaScript, HTML5 y CSS3 que permita al usuario crear documentos XML válidos de manera asistida a partir de XML Schema y opcionalmente Schematron. Para ello, utilizando la información proporcionada por el XML Schema, la aplicación genera de forma dinámica un formulario que reproduce la estructura de un documento XML válido, y el usuario interacciona con los elementos de dicho formulario de manera que el estado y el propio formulario van evolucionando hasta obtener un documento XML que es válido respecto del Schema puesto que el formulario se ha construido a partir de él.

El uso de Schematron es una de las características que hacen novedosa a la aplicación desarrollada frente a otras. Dicho documento define restricciones y relaciones entre diferentes elementos del documento XML y la aplicación utiliza esta información no solo para informar de restricciones que no se cumplen, sino también para prevenir errores de manera dinámica y anticipada, evitando que el usuario seleccione opciones que causarían errores.

En la aplicación también se han tenido en cuenta las necesidades de los usuarios desde el punto de vista visual y del tipo de documento que quieren obtener, por lo que se permite utilizar hojas XSLT para transformar el documento XML a otro tipo de documento más adecuado a las necesidades del usuario, como JSON, HTML o incluso PDF.

La aplicación se ha adaptado para ser utilizada en el ámbito de la dermatología mediante el uso de documentos XML Schema y Schematron que modelan patologías dermatológicas comunes y que han sido desarrollados en un Trabajo de Fin de Grado anterior [Bue14]. No obstante, la aplicación es genérica y se adaptará de manera dinámica a los documentos XML Schema y Schematron proporcionados por el usuario aunque fuesen otros diferentes.

Índice general

Capítulo 1

Introducción

Las virtudes del lenguaje XML (*EXtensible Markup Language*) [BPSM$^+$06], especialmente su simplicidad y su flexibilidad, lo convierten en un lenguaje idóneo para la transmisión y el almacenamiento de datos en muchos ámbitos. En la mayoría de dichos ámbitos no solo se requiere que los documentos XML estén bien formados, sino que además se precisa que sean válidos, es decir que cumplan unas determinadas condiciones que restringen su estructura, contenidos, tipos de datos etc.

El presente TFG se centra en la concepción y el desarrollo de una aplicación multiplataforma que permita al usuario generar documentos XML válidos de forma dinámica y asistida, a partir de XML Schema 1.0 [xsd04a] y opcionalmente Schematron [sch06], que son dos tecnologías muy potentes dentro de las muchas alternativas disponibles en el contexto de la validación XML.

XML Schema es probablemente el lenguaje de validación XML más ampliamente utilizado, ya que permite especificar de manera una manera sencilla pero muy precisa la estructura la de los documentos XML, así como restringir su contenidos. XML Schema define qué elementos pueden aparecer en un documento XML, en qué orden, cuántas veces, qué tipo de datos y restricciones debe cumplir el valor de cada elemento a nivel individual, etc.

El uso de documentos Schematron está menos extendido pero supone una herramienta muy potente de validación XML. Los documentos Schematron proporcionan validación de documentos XML basada en reglas mediante el uso de XPath [xpa04]. Schematron se suele usar en conjunción con la tecnología XML Schema, ya que ambas se complementan: XML Schema hace simple definir la estructura general del documento XML y proporciona validación avanzada para contenido simple, mientras que Schematron permite definir reglas que el documento XML debe satisfacer y, lo que es más importante, relaciones entre los valores de diferentes elementos del documento, es decir validaciones complejas de contenidos cruzadas, lo cual no puede lograrse con XML Schema.

El resultado principal de este proyecto ha sido la implementación de una aplicación multiplataforma, llamada Formulatron, que permite al usuario la creación de documentos XML válidos a partir de XML Schema y opcionalmente Schematron. Esta aplicación analiza el XML Schema y presenta de forma dinámica un formulario que reproduce la estructura del Schema y guía al usuario de manera dinámica y asistida en el proceso de completado del formulario. Cuando el usuario finaliza el proceso obtiene un documento

XML que por construcción es válido, ya que el formulario se había creado reproduciendo la estructura del XML Schema. Por otra parte la aplicación utiliza la información proporcionada por el documento Schematron para asistir al usuario aun más en el proceso de completado del formulario, informándole de fallos y evitando de manera anticipada que seleccione ciertas opciones del formulario que, dado el estado actual, causarían un error en caso de ser seleccionadas. Cabe señalar que la aplicación desarrollada permite además aplicar transformaciones XSLT [Cla99] al documento XML válido, lo que permite que el usuario final pueda obtener como resultado otro tipo de documentos, como HTML o incluso documentos de tipo PDF a través de un servicio web que se ha creado para tal fin.

La aplicación desarrollada es genérica en el sentido de que no espera unos XML Schema y Schematron concretos sino que se adapta de forma dinámica a los documentos proporcionados. No obstante, para comprobar el funcionamiento de la aplicación se han utilizado XML Schemas y Schematrones de uso médico que modelan la formulación magistral de patologías dermatológicas comunes, desarrollados en otro Trabajo de Fin de Grado [Bue14]. Además, se ha hecho que la aplicación incluya estos documentos como ficheros predefinidos, y se ha adaptado para dicho uso por parte de dermatólogos.

1.1. Introducción a XML Schema

Esta sección ofrece una breve y básica introducción a XML Schema, una tecnología de validación XML. Un documento XML Schema especifica al estructura y los tipos de contenidos que debe tener un documento XML para ser válido. Algunos de los elementos más importantes de XML Schema y su funcionalidad se resumen en la lista siguiente:

- *xs:schema*: es el raíz de los documentos XML Schema. Está asociado al espacio de nombres `http://www.w3.org/2001/XMLSchema` cuyo prefijo preferido es *xs*.
- *xs:element*: permite definir un elemento del documento XML. Puede ser de tipo simple si no contiene otros elementos, o de tipo complejo si los contiene. El tipo del elemento se especifica en el atributo *type* o utilizando un hijo *xs:simpleType* o *xs:complexType*.
- *xs:attribute*: especifica un atributo de un elemento del documento XML.
- *xs:simpleType*: permite definir un tipo simple, es decir, el contenido de texto de un elemento del documento XML. Puede definirse utilizando los tipos predefinidos en XML Schema [xsd04c], mediante restricciones, o bien a través de listas y o uniones.
- *xs:restriction*: permite restringir los valores de un tipo simple mediante el uso de elementos de restricciones tales como *xs:maxInclusive*, *xs:maxExclusive*, *xs:maxLength*, *xs:enumeration*, *xs:pattern*, etc.
- *xs:complexType*: define un tipo complejo, es decir que puede contener otros elementos y atributos. Existen tres tipos complejos principales: *xs:sequence*, *xs:all* y *xs:choice*.
- *xs:sequence*: especifica una serie de elementos que debe aparecer en el documento XML en el orden en el que se especifican en la secuencia.
- *xs:all*: especifica una serie de elementos que pueden aparecer en el documento XML en cualquier orden.

- *xs:choice*: especifica una serie de elementos de los que solamente uno de ellos puede aparecer en el documento XML.
- Attributos *maxOccurs* y *minOccurs*: Son atributos de los elementos que permiten definir el número de veces que puede aparecer como máximo y como mínimo dicho elemento en el documento XML. Su valor es por defecto es 1.

La figura 1.1 muestra un ejemplo básico de un XML Schema que modela la estructura de un documento XML que contiene la información de una persona. Según este Schema, en el documento XML un elemento *persona* tiene que tener el siguiente contenido y en este orden puesto que se ha utilizado una secuencia (elemento *xs:sequence*):

- Elemento obligatorio *nombre* de tipo *string*.
- Elemento *edad* de tipo *int* que puede no aparecer puesto que el valor su atributo *minOccurs* es 0.
- Elemento obligatorio *sexo* cuyo valor puede ser *Hombre* o *Mujer*.
- Elemento obligatorio *embarazo* de tipo *boolean*.
- Elemento *info* de tipo *string* es opcional.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persona">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string"/>
        <xs:element name="edad" type="xs:int" minOccurs="0" />
        <xs:element name="sexo">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Hombre"/>
              <xs:enumeration value="Mujer"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="embarazo" type="xs:boolean"/>
        <xs:element name="info" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 1.1: Ejemplo de documento XML Schema que modela la estructura de un documento XML que contiene la información de una persona.

La figura 1.2 muestra un ejemplo de documento XML válido respecto del XML Schema de la figura 1.1. A continuación, para facilitar la comprensión de XML Schema, se explican otros casos que podrían darse con un XML Schema y un documento XML similares a los de las figuras 1.1 y 1.2 pero ligeramente modificados:

- El documento XML no sería válido si el valor del elemento *embarazo* fuera *"no"* ya que no es de tipo *boolean*.
- El documento XML seguiría siendo válido si el elemento *edad* y/o el elemento *info* no estuvieran presentes, puesto que en el Schema ambos tienen el atributo *minOccurs* con valor 0.
- El documento XML no sería válido si el elemento *edad* apareciera antes que el elemento *nombre* puesto que no se estaría respetando el orden de la secuencia. Sin embargo, si que sería válido si en lugar de un elemento *xs:sequence* se utilizase un elemento *xs:all*, ya que sería válido independientemente del orden en el que aparecieran sus hijos.
- El documento XML no sería válido si en lugar de utilizar *xs:sequence* hubiésemos utilizado el elemento *xs:choice*. En ese caso, para que el documento fuese válido solo debería aparecer uno de sus hijos.

```
<?xml version="1.0"?>
<persona>
  <nombre>Sergio Frago</nombre>
  <edad>22</edad>
  <sexo>Hombre</sexo>
  <embarazo>false</embarazo>
  <info>Nacido en Zaragoza en 1992</info>
</persona>
```

Figura 1.2: Ejemplo de documento XML válido respecto del XML Schema de la figura 1.1.

El XML Schema presentado como ejemplo en la figura 1.1 modela una estructura relativamente simple, no obstante, la estructura modelada podría ser arbitrariamente compleja y recursiva si, por ejemplo, añadiéramos otros elementos complejos dentro del elemento complejo *persona*.

Puede encontrarse toda la información relativa a XML Schema 1.0 en las tres partes de la especificación del *W3C* [xsd04a] [xsd04b] [xsd04c].

1.2. Acrónimos utilizados

La lista siguiente muestra los acrónimos que se utilizan en este documento y su significado:

- AJAX: *Asynchronous JavaScript And XML.*
- API: *Application Programming Interface.*
- CSS: *Cascading Style Sheets.*
- CSV: *Comma-Separated Values.*
- DOM: *Document Object Model.*

- HTML: *HyperText Markup Language.*
- IEC: *International Electrotechnical Commission.*
- ISO: *International Organization for Standardization.*
- JSON: *JavaScript Object Notation.*
- PHP: *PHP: Hypertext Preprocessor.*
- TFG: Trabajo de Fin de Grado.
- W3C: *World Wide Web Consortium.*
- XML: *eXtensible Markup Language.*
- XPath: *XML Path Language.*
- XSLT: *Extensible Stylesheet Language Transformations.*

1.3. Estructura del documento

El resto del documento presenta la siguiente estructura y contenidos:

- **Paradigma de validación XML tradicional frente al paradigma de la aplicación.** En este capítulo se explica qué es lo que hace diferente y novedosa a la aplicación desarrollada frente al paradigma de validación XML clásico. Se exponen otras aplicaciones con un funcionamiento similar que también estén basadas en generar un formulario a partir de un XML Schema para obtener un documento XML válido.
- **Arquitectura general del sistema.** Se da una visión de alto nivel del sistema desarrollado, se explican las diferentes alternativas de desarrollo de la aplicación, la solución que se escogió, y las tecnologías utilizadas para desarrollar este proyecto.
- **El uso de XML Schema en la aplicación.** En este capítulo se detallan los usos de XML Schema en la aplicación y se explican los elementos que componen el formulario.
- **El uso del Schematron en la aplicación.** Este capítulo da una introducción a Schematron y detalla los usos que hace la aplicación del proporcionado por el usuario.
- **Transformaciones en la aplicación.** Explica el uso de las transformaciones y de XSLT en la aplicación, muestra la hoja XSLT creada para su uso en dermatología y explica el servicio web creado para la transformación de HTML a PDF.
- **Conclusiones y trabajo futuro.** Detalla los resultados y conclusiones obtenidos del desarrollo de este Trabajo de Fin de Grado así como posibles futuras vías de ampliación y uso de los resultados del Trabajo.

Capítulo 2

Paradigma de validación XML tradicional frente al paradigma de la aplicación

La aplicación desarrollada plantea un enfoque novedoso y completamente diferente al de la validación XML tradicional. Las diferencias entre ambos enfoques se dan principalmente en estos cuatro aspectos:

1. Los datos de entrada: cuáles son los documentos o elementos de los que partimos.
2. El objetivo: qué es lo que se quiere conseguir.
3. El flujo de procesos: qué pasos se siguen desde que se tienen los datos de entrada hasta que se obtienen los resultados.
4. Resultados: qué se obtiene como resultado del proceso seguido.

El enfoque de la validación XML tradicional y el enfoque de la aplicación, así como sus características desde los cuatro puntos de vista citados anteriormente, se muestran en la figura 2.1 y se explican a continuación:

1. Enfoque de la validación XML tradicional:
   - Datos de entrada: se parte de un Documento XML, un XML Schema y opcionalmente un Schematron. Aunque es posible utilizar otro tipo de documentos diferentes de XML Schema y Schematron, el uso de estos documentos es relativamente común.
   - Objetivo: comprobar si el documento XML es válido respecto al XML Schema y al Schematron si lo hubiera.
   - Flujo de procesos: se valida el documento XML inicial utilizando el XML Schema, si es válido se valida también con el Schematron.
   - Resultado: el resultado del proceso es una afirmación, podríamos decir que es un dato *booleano*, ya que el resultado es simplemente una conclusión sobre si el documento XML inicial es o no válido respecto al XML Schema y al Schematron si lo hubiera.
2. Enfoque de la aplicación desarrollada:

- Datos de entrada: en la aplicación desarrollada se parte de un XML Schema y opcionalmente un Schematron. También se puede añadir un documento XSLT aunque no tiene relevancia en cuanto a la generación y validación del documento XML.
- Objetivo: que el usuario obtenga de manera dinámica y asistida un documento XML válido respecto del XML Schema que ha proporcionado y respecto del Schematron si lo hubiera.
- Flujo de procesos: la información del XML Schema inicial se usa para presentar al usuario un formulario que reproduce la estructura que debe cumplir un documento XML válido. Dicha información, así como la del Schematron si lo hubiera, se utiliza para asistir al usuario en el proceso de creación del documento.
- Resultado: el resultado del proceso es un documento XML que es válido respecto del XML Schema y respecto del Schematron si lo hubiera, puesto que el formulario presentado al usuario se ha construido a partir del XML Schema de forma que el documento resultante es válido por construcción.

Además de todo eso, una vez acabado el proceso, en la propia aplicación se permite al usuario transformar el documento XML resultante mediante XSLT, obteniéndose como resultado otro documento XML u otro tipo de documento (como por ejemplo HTML que puede resultar mucho más atractivo visualmente), y se permite también transformarlo a PDF como paso posterior a la transformación HTML, lo cual aporta una gran ventaja de cara al usuario final, ya que le permite tener la información en un formato más adecuado a sus necesidades y mejor desde el punto de vista de la visualización y presentación. Aunque esto también sería posible hacerlo por otros medios después del proceso de validación XML tradicional, es algo que está completamente fuera de dicho proceso, mientras que está completamente integrado en la aplicación desarrollada.

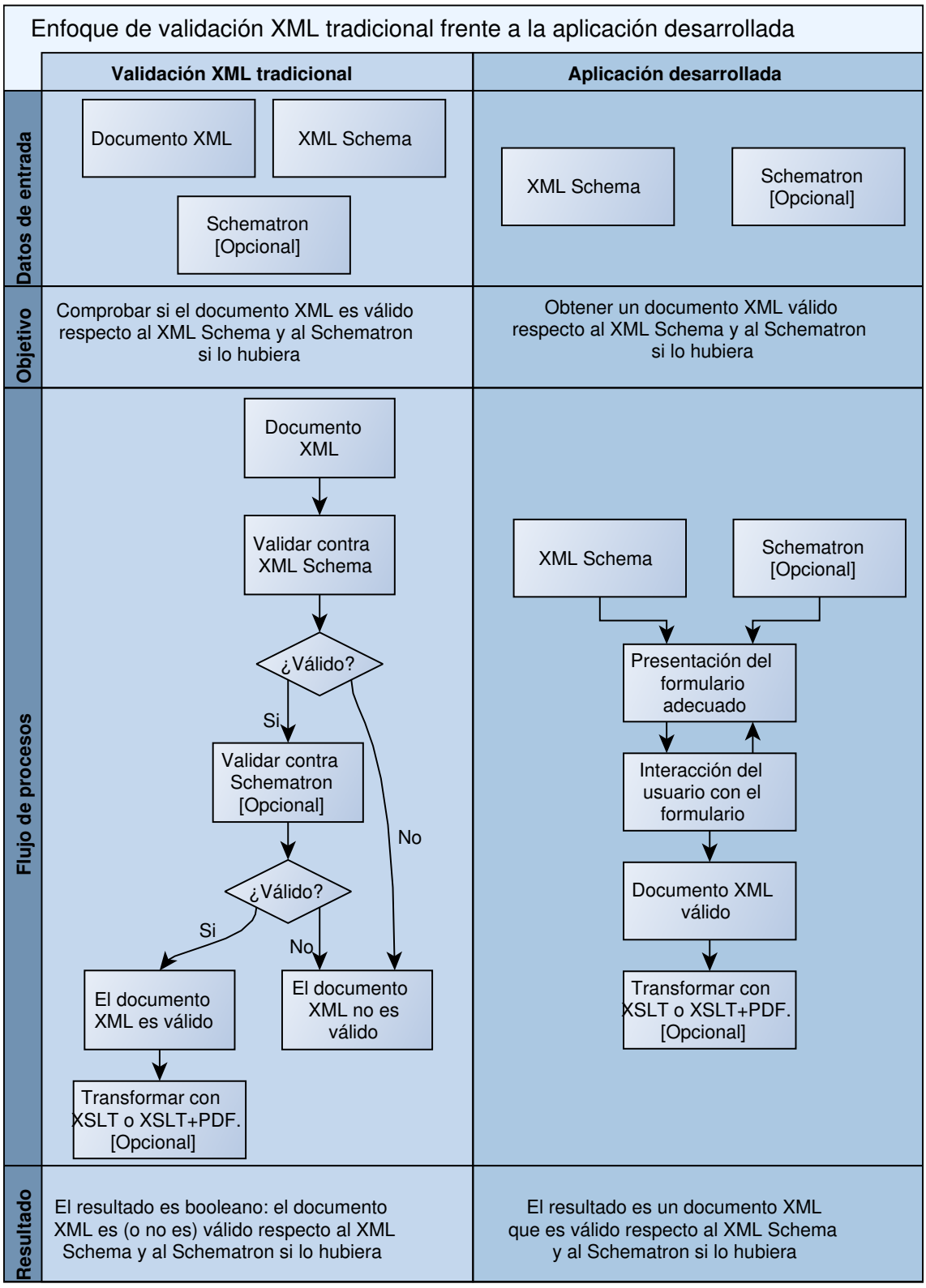


Figura 2.1: Diferencias entre el enfoque tradicional de validación XML y el enfoque de la aplicación desde el punto de vista de los datos de entrada, el objetivo, el flujo de procesos y el resultado.

2.1. Aplicaciones similares

Aunque Formulatron presenta funcionalidades que la hacen novedosa, existen aplicaciones similares, en el sentido de que siguen un flujo de procesos parecido. Algunas de las aplicaciones similares encontradas, así como algunas de sus características se describen brevemente en la lista siguiente:

Xsd-forms [dav14]
: Es un generador de formularios web (JavaScript, HTML, CSS) a partir de XML Schema. Está implementado utilizando Scala y Java. Permite un subconjunto de los elementos de XML Schema aunque no soporta atributos. Permite añadir información para mejorar la presentación mediante la edición del XML Schema con unas anotaciones determinadas. Permite también sobrescribir estilos y enviar el XML resultante al servidor que el usuario elija. La aplicación puede usarse también desde un servidor que proporciona el autor. El proyecto está hospedado en la dirección `https://github.com/davidmoten/xsd-forms`.

XSDForm [Sof]
: Es una aplicación comercial cuya licencia cuesta 50$, aunque cuentan con versión de prueba gratuita durante 20 días. Está basada en PHP y Ajax y es capaz de generar un formulario web a partir de XML Schema. Cuenta con entradas de datos adaptadas al tipo de datos, estilos configurables y permite que el documento XML se pueda enviar a un servidor. Puede obtenerse de la dirección `http://www.ilerian.com/xsd-web-form-overview`.

DynaForm [Rau10]
: Esta aplicación está implementada en Java y basada en el framework web Aranea. Es capaz de generar dinámicamente un formulario web a partir de un XML Schema y opcionalmente una instancia XML usada para la inicialización de datos. Es altamente configurable utilizando un lenguaje denominado DynaData creado por el autor que permite cambiar la distribución de los elementos, su estilo etc. Permite realizar cambios en el Schema o el XML que se reflejan de manera dinámica en la aplicación. El proyecto se encuentra hospedado en la dirección `https://github.com/reinra/dynaform`.

Hay que resaltar que ninguna de las aplicaciones similares encontradas presentan una sola de las siguientes características y funcionalidades, que sí que están presentes en la aplicación desarrollada:

- Soporte de un lenguaje de validación basado en reglas (Schematron).
- Prevención dinámica de errores.
- Navegación multipágina a través de los elementos del formulario.
- Transformaciones del documento XML a otro tipo de documentos mediante XSLT y XSLT->HTML->PDF.
- Aplicación concebida y desarrollada como aplicación móvil y multiplataforma.

Capítulo 3

Arquitectura general del sistema

Este capítulo explica el diseño a alto nivel del sistema desarrollado, entendiendo por sistema tanto la aplicación implementada como el servicio web externo que se ha puesto en marcha para su uso en Formulatron, cuya única funcionalidad es la obtención de documentos PDF a partir de documentos HTML que puedan contener CSS. En los capítulos posteriores se explican con mayor detalle la mayoría de las ideas que aquí se presentan.

3.1. Paradigma de desarrollo

En esta sección se introducen los dos paradigmas más importantes que existen en el desarrollo de aplicaciones multiplataforma, así como las ventajas e inconvenientes de cada uno de ellos. Finalmente se comenta la solución elegida y los motivos de escogerla.

3.1.1. Análisis de aproximaciones

En el ámbito de los Sistemas Operativos móviles existen básicamente dos alternativas bien diferenciadas para el desarrollo de aplicaciones multiplataforma:

1. Desarrollar una aplicación diferente para cada plataforma. Esta aproximación consiste en utilizar el lenguaje de programación específico de cada plataforma para obtener cada aplicación.

2. Utilizar frameworks que permiten crear aplicaciones para varias plataformas con un mismo código. Estas herramientas ahorran gran cantidad de trabajo si se quiere obtener una aplicación multiplataforma, puesto que solamente hay que desarrollar un único código fuente para todas las plataformas y el framework se encarga de generar las aplicaciones nativas.

La primera alternativa tiene como principal ventaja el rendimiento de la aplicación final, puesto que está directamente desarrollada en el lenguaje de la plataforma objetivo, mientras que la segunda aproximación tiene como principal ventaja la simplicidad de tener que desarrollar un único código fuente.

3.1.2. Solución adoptada

Para el presente proyecto se optó por utilizar un framework por la enorme ventaja que supone tener el mismo código fuente para todas las plataformas. No hubiera sido viable

en el tiempo programado para un TFG implementar la aplicación en un lenguaje de programación distinto para cada plataforma objetivo.

La herramienta elegida ha sido el framework Phonegap [NS14], que tiene como lenguajes de entrada JavaScript, HTML5 y CSS3, y a partir de ellos genera aplicaciones nativas para varias plataformas. Phonegap cuenta con varias ventajas que lo hacen más atractivo que otros frameworks similares, como la cantidad de plataformas soportadas y los múltiples Plugins disponibles, que dan acceso común a *APIs* del Sistema Operativo móvil, como por ejemplo al Sistema de ficheros.

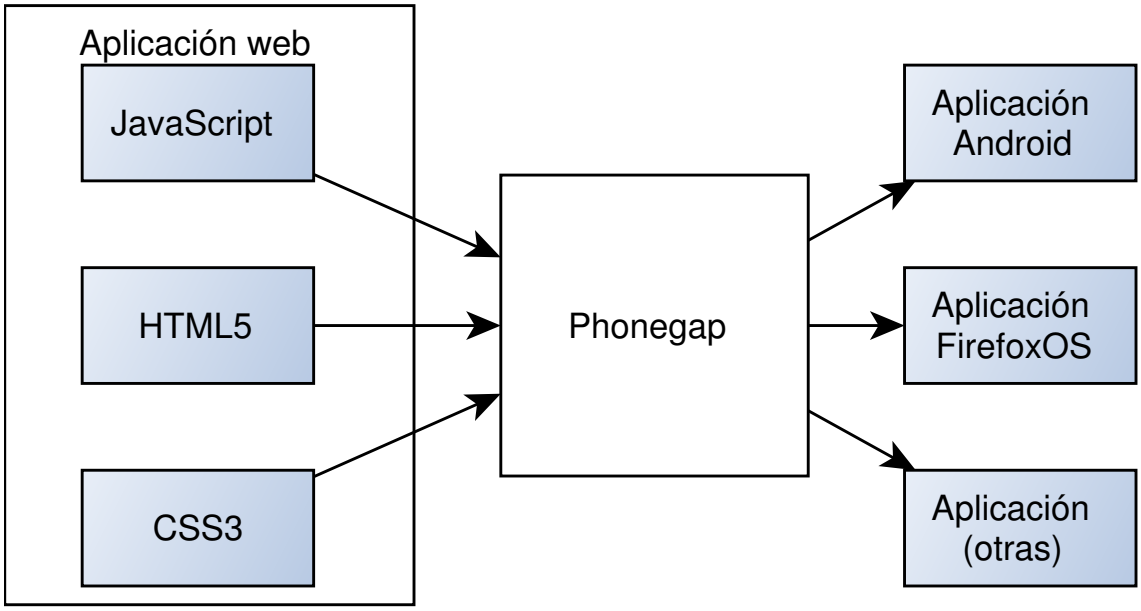


Figura 3.1: Obtención de aplicaciones a partir del código JavaScript, HTML5 y CSS3. El propio código constituye en si una aplicación web que puede ser usada en los navegadores comunes sin necesidad de Phonegap, por otro lado, se esquematiza el uso de Phonegap para la obtención de las aplicaciones para las plataformas móviles.

Las herramientas de desarrollo de Phonegap, y por tanto de la aplicación desarrollada, son JavaScript, HTML5 y CSS3, que son las tecnologías más utilizadas en la web y en el desarrollo web. El proceso de obtención de aplicaciones móviles a partir de JavaScript, HTML5 y CSS3 mediante el uso de Phonegap se encuentra esquematizado en la figura 3.1. El uso de cada una de estas tecnologías es el siguiente:

HTML5 permite definir y diseñar las páginas web especificando su arquitectura y estructura, por tanto es el punto de entrada para diseñar la interfaz gráfica de la aplicación.

CSS3 nos permite crear hojas de estilo, que contienen propiedades que especifican el estilo de visualización que deben tener los elementos de las páginas HTML5.

JavaScript es un lenguaje de programación interpretado orientado a objetos cuyo uso es habitual en las páginas web aportando dinamismo y permitiendo realizar muchas funciones entre las que se incluye la interacción de manera sencilla con los elementos de las páginas HTML5.

Ya que la aplicación se ha desarrollado utilizando estos lenguajes que son la base de la web, el desarrollo de la aplicación se puede considerar equivalente al desarrollo de una página web, si bien la aplicación desarrollada es una aplicación que puede utilizarse *offline* salvo que se quiera utilizar la funcionalidad de transformar el documento a PDF, para lo cual se utiliza un servicio externo creado. En este sentido, la aplicación no solo funciona como aplicación nativa en plataformas móviles sino que también funciona en los navegadores de las plataformas de escritorio.

3.2. Componentes del Sistema

En esta sección se introducen los componentes o módulos en los que están divididas la aplicación y el servicio web, así como las relaciones que existen entre ellos. La figura 3.3 proporciona una visión general de los principales componentes del sistema así como de las comunicaciones entre ellos.

3.2.1. Componentes de la aplicación

Ya que el lenguaje en el que se desarrolla el código de la aplicación es JavaScript, el paradigma de programación es el de orientación a objetos, lo que ha permitido dotar de modularidad a la arquitectura de la aplicación. A continuación se explican brevemente los módulos que componen a alto nivel la aplicación. El orden de descripción de los módulos es similar al orden en el que se usan en el proceso global de funcionamiento de la aplicación, tal y como se esquematiza en la figura 3.2.

SchemaParser (parseador del Schema). Parsea el contenido del fichero o ficheros XML Schema que el usuario haya seleccionado y genera dinámicamente un formulario cuyos elementos reproducen la estructura definida por el XML Schema.

SchematronParser (parseador del Schematron). Parsea las restricciones del documento Schematron.

FormElement (elemento del formulario). Este módulo representa los elementos del formulario, el parseado de cada elemento del XML Schema genera un elemento del formulario diferente. Estos elementos evolucionan cambiando de estado con la interacción del usuario, y cambiando a su vez el estado del documento XML asociado.

Paginador. Permite al usuario realizar una navegación basada en páginas a través de los elementos del formulario, de esta forma el proceso de completado del formulario puede estructurarse de una manera mucho más lógica y visual.

Estado. Contiene el estado del formulario, y es por tanto un estado intermedio del documento XML, que va evolucionando con la interacción del usuario.

Comprobador de las restricciones Schema. Comprueba que se cumplen las restricciones del Schema en un FormElement simple determinado. Este módulo actúa en tiempo real en cuanto el usuario interacciona con el FormElement modificando su estado.

Comprobador de las restricciones de Schematron. Comprueba que las restricciones del Schematron se cumplen en el estado intermedio del documento XML. Este módulo actúa cada vez que el usuario ha interaccionado con el formulario produciendo un cambio que ha hecho evolucionar el estado y el documento XML.

Módulo de transformaciones. Permite transformar el documento XML a otro tipo de documentos (tales como HTML u otro XML) utilizando el XSLT seleccionado por el usuario. Permite también transformar el documento XML a PDF utilizando la hoja de estilos XSLT para transformar inicialmente a HTML y posteriormente obtener el PDF a parir del servicio web externo creado.

Sistema de ficheros. Este módulo es un explorador de ficheros que se ha implementado e integrado en la aplicación y que proporciona funcionalidades para leer o guardar ficheros del sistema de ficheros del dispositivo.

Ficheros predefinidos. Este módulo permite al usuario no tener que buscar y cargar los ficheros de entrada (XML Schema y opcionalmente Schematron y/o XSLT) cada vez que inicia la aplicación, ya que le habilita para guardar los ficheros y poder comenzar la edición en sucesivos inicios de la aplicación. Este módulo incluye precargados los ficheros XML Schema y Schematron de patologías dermatológicas desarrollados en [Bue14] así como un XSLT que permite transformar el documento XML resultante a HTML y posteriormente a PDF.

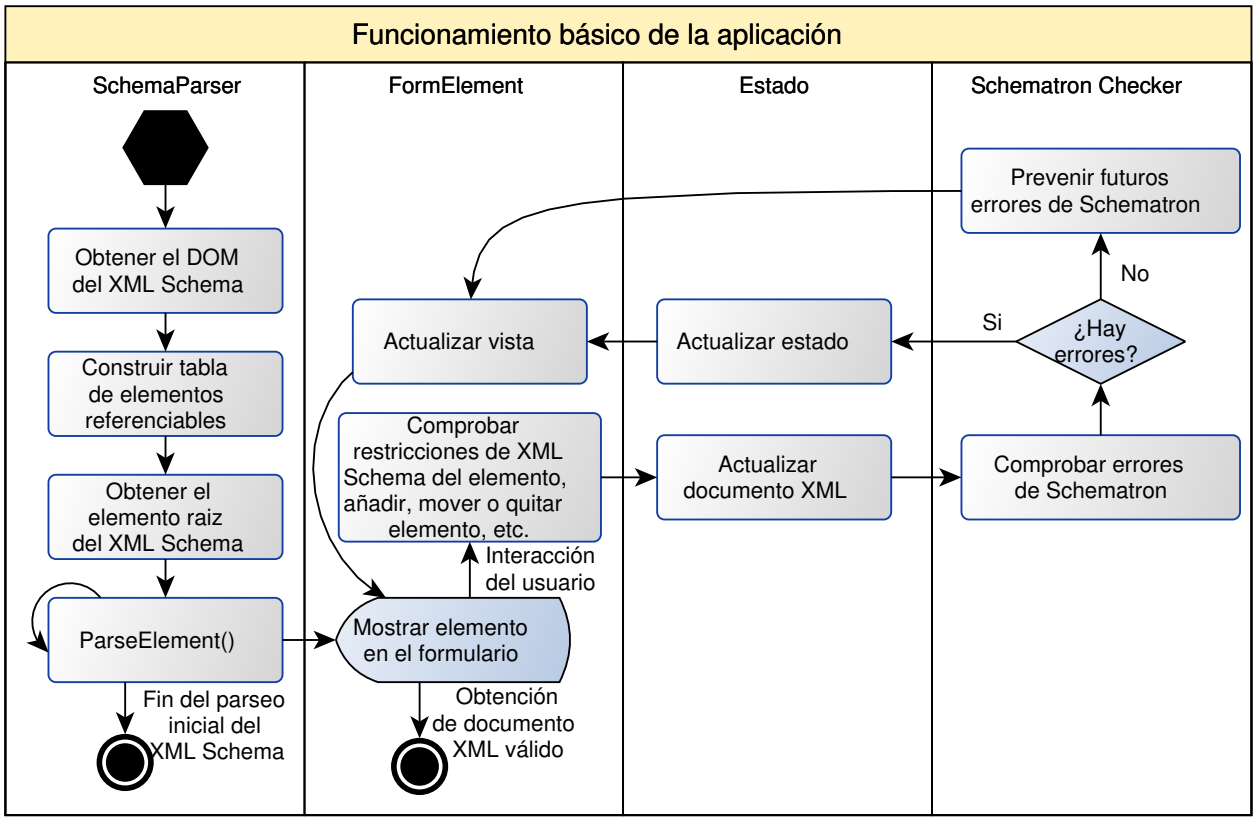


Figura 3.2: Proceso de funcionamiento básico de la aplicación. Esquematiza el trabajo que realizan los principales módulos en la secuencia básica de pasos del parseado inicial del XML Schema y en el proceso iterativo en el que el usuario va interactuando con el formulario hasta que obtiene como resultado un documento XML válido.

3.2.2. Componentes del servicio web

El servicio web creado tiene como funcionalidad proporcionar documentos PDF a partir de documentos HTML con CSS. Se ha creado un servicio externo con esta funcionalidad dado que a menudo es necesario o útil obtener un PDF como resultado, pero no śe han encontrado bibliotecas JavaScript adecuadas para realizar la transformación en la propia aplicación.

A continuación se enumeran y describen brevemente las partes en las que se divide este servicio:

Procesador de parámetros. Este componente simplemente lee los parámetros de la petición POST.

Transformador. Este módulo se encarga de transformar el documento HTML a PDF y proporciona el PDF resultante.

El servicio creado está implementado en PHP y presenta la información necesaria accesible en la dirección `http://formulatron.hol.es/htmlToPdf/index.html`.

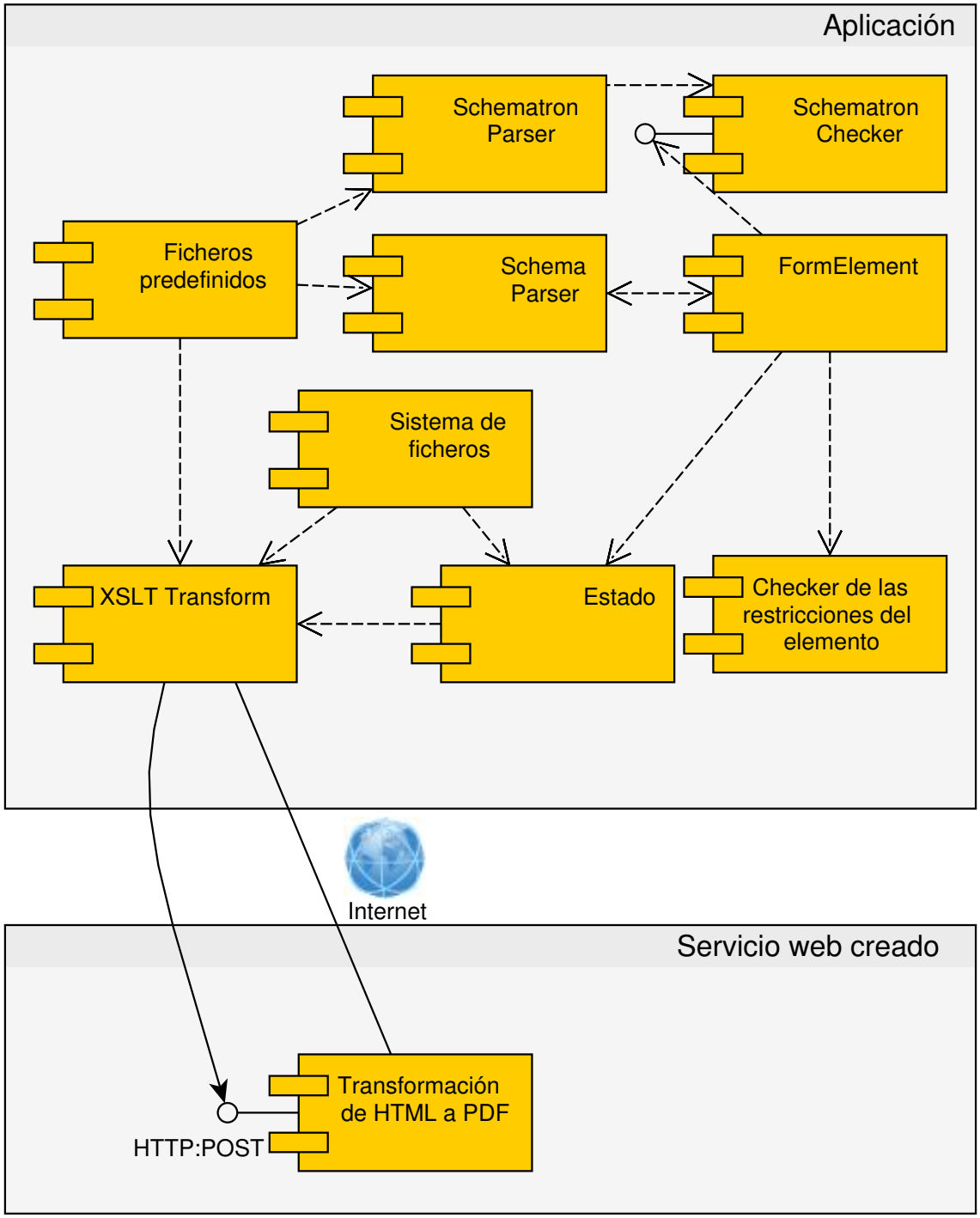


Figura 3.3: Diagrama de los principales componentes de la aplicación y del servicio web desarrollados y relaciones de comunicación entre ellos.

Capítulo 4

El uso de XML Schema en la aplicación: estructura y elementos del formulario

Ya que el objetivo principal es obtener un documento XML válido respecto a un XML Schema, la aplicación desarrollada presenta al usuario un formulario que reproduce la estructura definida en un XML Schema. Cada uno de los elementos que internamente componen el formulario se denomina elemento del formulario o FormElement. Los FormElements y su estructura se van construyendo en el proceso de parseado de los elementos del XML Schema, de este modo lo que se logra es que por definición el documento XML final cumpla con la estructura especificada en el Schema. Ya que los FormElements siguen la estructura dada por el XML Schema, son elementos recursivos, es decir, dentro de cada FormElement puede haber otros FormElements.

Dada esta estructura recursiva del Schema y su posible gran complejidad la aplicación no parsea todo el Schema de una sola vez, sino que realiza podas inteligentes para no parsear los elementos que puedan no aparecer. Por ejemplo, si un elemento es opcional, inicialmente solo se parseará su nombre y se le presentará al usuario la posibilidad de activarlo, y solamente en el caso de que el usuario decida activarlo se parseará el elemento asociado en el XML Schema y su estructura arbitrariamente compleja. Esta optimización de parseado dinámico se ha implementado en varios FormElements y ha permitido a la aplicación ahorrar un gran trabajo de parseado de elementos que no se van a utilizar.

En la lista siguiente se describen muy brevemente los tipos de FormElements que existen. Existe una descripción completa de cada uno de los tipos de FormElement y los elementos del XML Schema a partir de los que se generan disponible en el Anexo B.

- ValueEnum: elemento que permite seleccionar un valor entre un conjunto de valores.
- ValueInput: entrada de texto que puede tener un slider asociado.
- XMLElement: elemento que representa un nodo de tipo elemento del documento XML.
- Attribute: elemento que representa un atributo de un elemento XML.
- Sequence: secuencia de otros elementos que deben aparecer en un orden determinado.

- Choice: elemento que permite elegir un elemento de entre todos los que lo componenen.
- All: elemento que contiene otros elementos que pueden aparecer en cualquier orden.
- List: elemento que contiene un FormElement que puede aparecer varias veces.

Aunque cada uno de estos elementos presenta propiedades y operaciones diferentes, existen propiedades y operaciones que son comunes a todos ellos tal y como se puede ver en la figura 4.1. Dichas propiedades y su funcionalidad son las siguientes:

- Propiedad *id*: identificador del FormElement.
- Propiedad *childs*: lista de hijos FormElement.
- Propiedad *parent*: FormElement padre.
- Propiedad *viewNode*: contiene la vista, es decir, el nodo HTML DOM asociado.
- Propiedad *xmlObject*: contiene la información del FormElement en el documento XML.

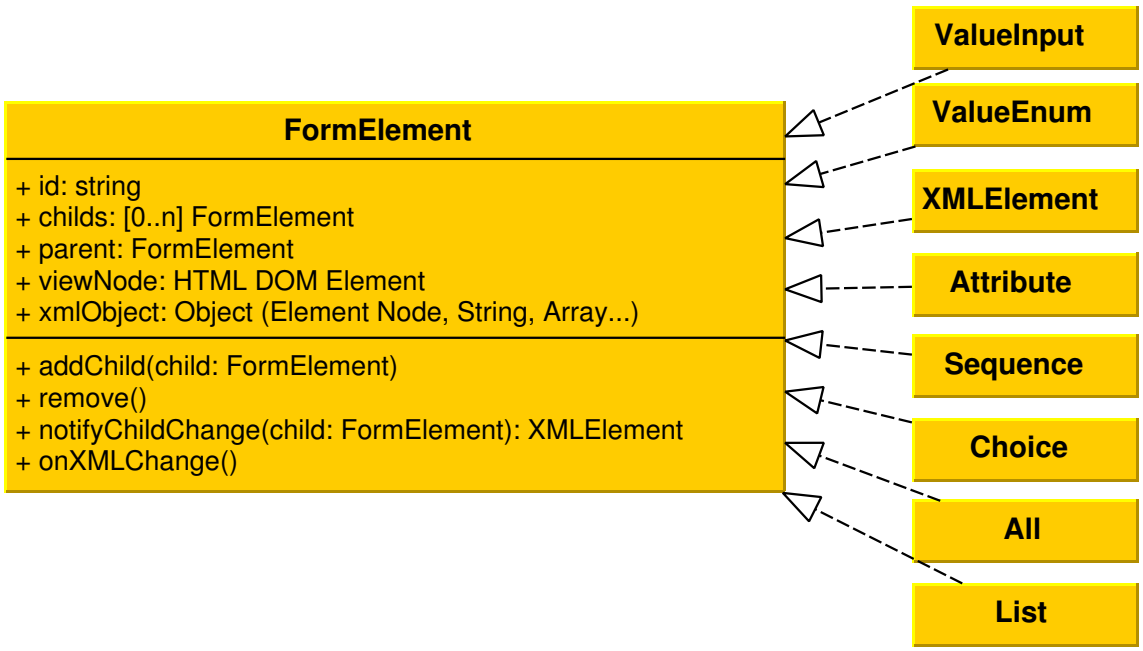


Figura 4.1: Diagrama de clase que muestra las propiedades y operaciones comunes a todos los FormElements, asi como los tipos de FormElements existentes.

En la figura 4.2 se da un ejemplo de un XML Schema y los FormElements que se generarían a partir de él. Con ello se intenta facilitar la comprensión de la estructura recursiva de los FormElements y las ideas recientemente expuestas.

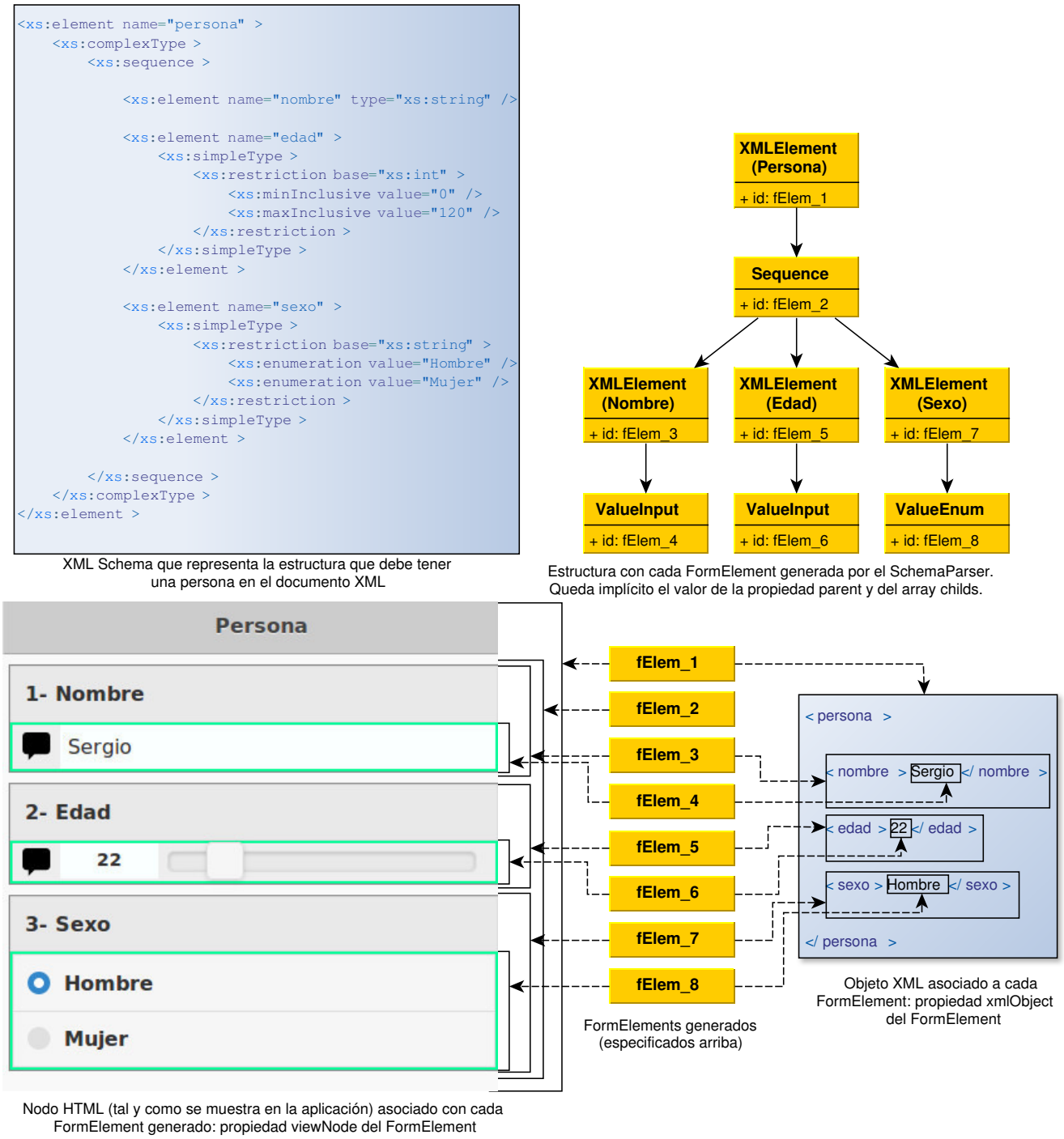


Figura 4.2: Ejemplo de un XML Schema y la estructura que generaría al ser parseado. Permite ver la estructura de los FormElements asi como entender sus propiedades. Nótese que los FormElements amarillos de arriba y de abajo son los mismos, simplemente se han utilizado los identificadores para simplificar el diagrama.

4.1. Evolución del formulario

Cuando el usuario proporciona su Schema a la aplicación, obtiene un formulario que inicialmente no es válido, salvo en el caso extraño de que el XML Schema sea tan simple que no requiera ningún tipo de interacción con el usuario. Por tanto, el formulario es un formulario vivo, en tanto en cuanto va evolucionando y cambiando de estado conforme el usuario interacciona con él para completarlo. Así mismo, cada cambio en el formulario es reflejado como cambio en el documento XML que contiene el estado actual.

La evolución del formulario, guiada por la interacción del usuario con los FormElements, es la que permitirá obtener el documento XML válido cuando todos los elementos se encuentren en un estado válido. Dado que los elementos del formulario pueden cambiar de estado, se consigue un estado válido (documento XML válido) cuando se cumplen las siguientes premisas:

1. Todas las enumeraciones (FormElements de tipo ValueEnum) tienen un valor seleccionado.
2. Todos los sliders e inputs (FormElements de tipo ValueInput) están inicializados y tienen un valor correcto respecto a las restricciones del Schema.
3. Todos los elementos de selección (FormElements de tipo Choice) tienen uno de sus elementos seleccionado.
4. No hay errores de Schematron en el caso de que haya un Schematron seleccionado. Véase el capítulo Capítulo 5.

Una vez que se cumplan estas condiciones el estado del formulario será válido y también será válido el estado del documento XML asociado. Esto permitirá al usuario guardar el documento XML, aplicarle transformaciones XSLT, o transformarlo a HTML mediante XSLT y después a PDF. No obstante, aunque el estado no sea correcto, el usuario puede ver en cualquier momento el estado del XML, el XML transformado a texto o el XML transformado como HTML. Así mismo, también se le muestran comentarios sobre el estado de los valores de los tipos simples y errores relativos al Schematron si los hubiera.

La figura 4.3 muestra las distintas interacciones que puede hacer el usuario con cada tipo de FormElement así como el proceso que desencadenan en la aplicación. Como se puede comprobar, cada interacción del usuario con los FormElemnts provoca cambios en el documento XML, la comprobación de los errores de Schematron con ese estado del documento XML, y la prevención de futuros errores de Schematron. La figura 4.4 muestra los procesos desencadenados por las notificaciones internas que puede recibir cada tipo de FormElement mientras trata las notificaciones generadas en las interacciones del usuario especificadas en la figura 4.3.

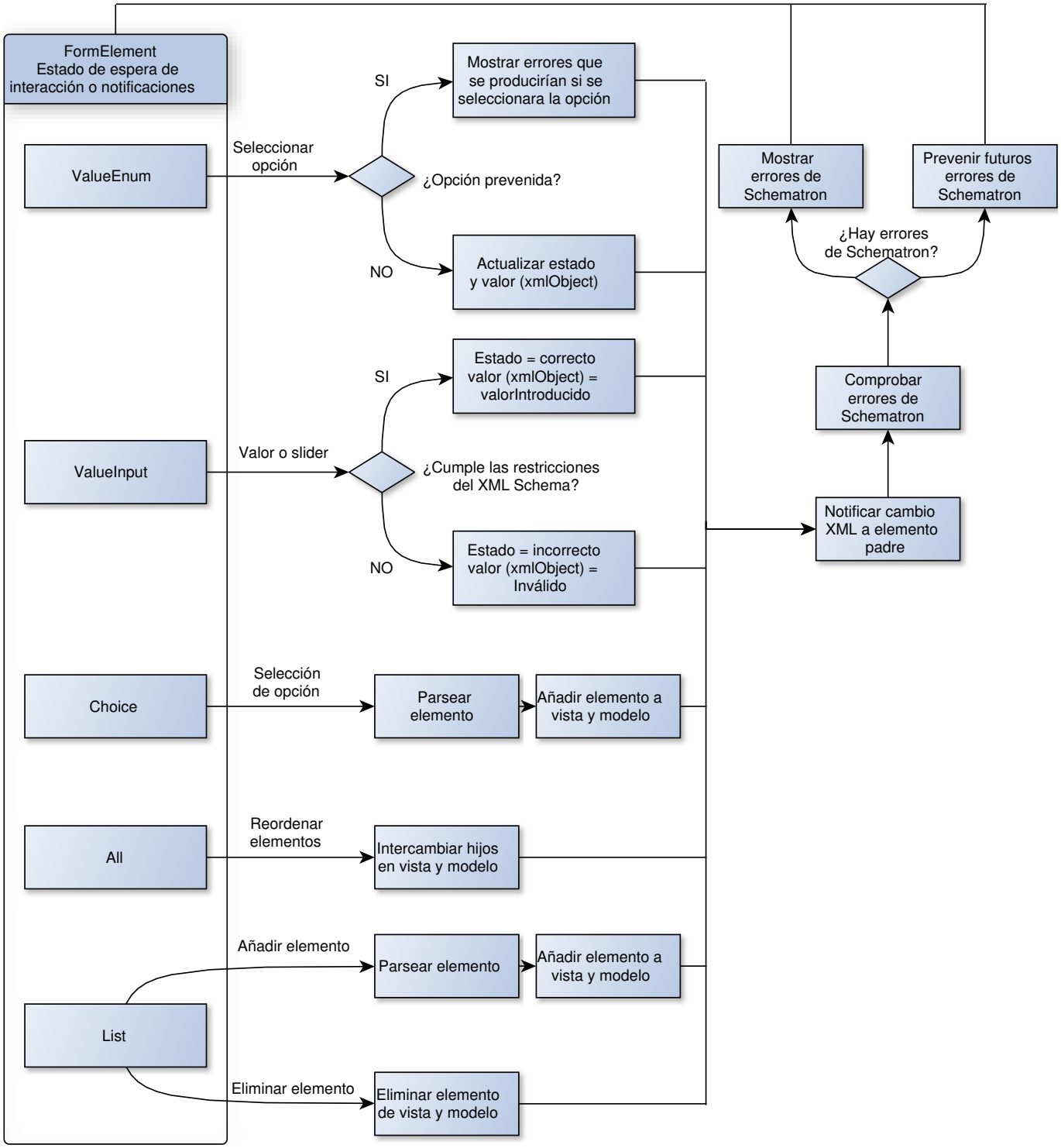


Figura 4.3: Posibles interacciones del usuario con cada tipo de FormElement y proceso que se sigue para tratarlas.

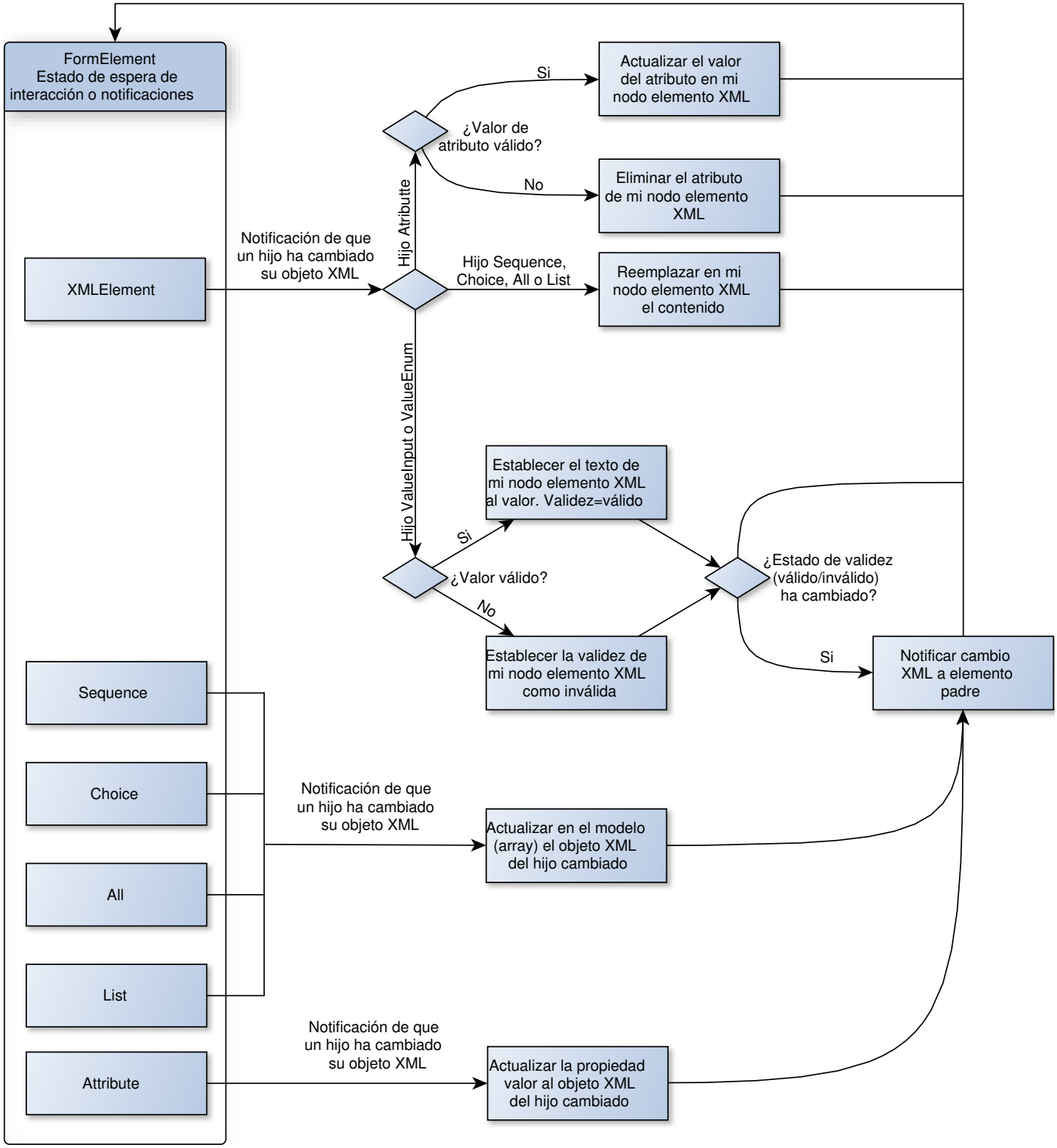


Figura 4.4: Posibles notificaciones que puede recibir cada tipo de FormElement y proceso que se sigue para tratarlas.

4.2. Namespace destinado a las mejoras

Más allá de los *namespaces* de XML Schema y del documento XML final (atributo *targetNamespace* del elemento raíz del XML Schema), se ha creado un espacio de nombres para permitir añadir mejoras y nuevas funcionalidades a la aplicación a través de atributos en los elementos del XML Schema. Este espacio de nombres tiene como URI *http://ehealthz.unizar.es/formulatron/improvements* y como prefijo preferido *imp*.

La especificación de XML Schema permite que un documento XML Schema sea extendido utilizando espacios de nombres definidos por el usuario para uso propio de las aplicaciones, por lo que el XML Schema extendido que utilice el espacio de nombres creado seguirá siendo válido de cara a otros procesadores y validadores, aunque dicha información solo será utilizada por nuestra aplicación.

Estas mejoras se han implementado como atributos opcionales de los elementos del XML Schema, y en ningún caso alteran el documento XML final, simplemente modifican la vista que se presenta al usuario para facilitar el uso de la aplicación. A continuación se describen brevemente cada una de las mejoras disponibles:

- *imp:inNewPage* si su valor es *true* el elemento se mostrará en una nueva página. Permite una navegación basada en páginas mucho más intuitiva y eficaz de cara al usuario.
- *imp:listInNewPage* si su valor es *true* la lista asociada al elemento se mostrará en una nueva página.
- *imp:image* se mostrará la imagen de la *url* especificada junto al elemento.
- *imp:listImage* se mostrará la imagen de la *url* especificada junto a la lista asociada al elemento.
- *imp:label* se mostrará el texto especificado junto al elemento.
- *imp:listLabel* se mostrará el texto especificado junto a la lista asociada al elemento.

Capítulo 5

El uso del Schematron en la aplicación

Este capítulo contiene una breve introducción a la tecnología Schematron, una explicación de las ventajas que aporta su uso y un resumen del uso que se hace de dicha tecnología en el ámbito de este proyecto.

5.1. Introducción a Schematron, necesidad y ventajas de su uso

Schematron es un lenguaje de validación XML basado en reglas soportadas por XPath que ha sido estandarizado en una norma *ISO/IEC* [sch06]. Es un lenguaje simple pero potente, ya que está basado en XPath, que es un lenguaje de procesamiento de nodos XML. Seguidamente se presentan y explican dos de los elementos más importantes en Schematron:

Elemento *sch:rule*
: Contiene un conjunto de reglas que se evalúan sobre un contexto establecido en el atributo *context*.

Elemento *sch:report*
: Es una regla que contiene una expresión XPath en el atributo *test* y un mensaje. Si la expresión XPath evaluada sobre el contexto se cumple entonces se produce un error y se muestra el mensaje asociado.

Elemento *sch:assert*
: Es también una regla que contiene una expresión XPath en el atributo *test* y un mensaje. Si la expresión XPath evaluada sobre el contexto no se cumple entonces se produce un error y se muestra el mensaje asociado. Es similar a un elemento *sch:report*, aunque en el caso del elemento *sch:assert* el error se genera cuando su condición no se cumple.

A continuación, en la figura 5.1, se presenta un ejemplo sencillo de una regla de Schematron para ayudar a la comprensión de esta tecnología, junto a él se da un ejemplo de documento XML que se intentaría validar contra el Schematron, y que dispararía el error de la regla ya que se cumple.

```
<!-- Ejemplo de regla que contendría el Schematron -->
<rule context="factura">
    <report test="precio < 6 and tipoDePago = 'Con tarjeta'">
        No pueden pagarse con tarjeta facturas de menos de 6 euros.
    </report>
</rule>

<!-- XML con una factura válida respecto del Schema pero no del Schematron-->
<factura>
    <precio>5</precio>
    <tipoDePago>Con tarjeta</tipoDePago>
</factura>

<!-- Documento XML Schema que modela la factura -->
<xs:element name="factura" imp:inNewPage="true">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="precio" type="xs:float" />
            <xs:element name="tipoDePago"><html><b>Se deshabilita un valor ya que si se seleccionara causaría error</b></htm
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Con tarjeta" />
                        <xs:enumeration value="En efectivo" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figura 5.1: Ejemplo de regla de Schematron, XML Schema y documento XML a validar. Se produciría un error de Schematron en la validación porque la regla se cumple, y por tanto, se mostraría el mensaje de error asociado.

Con el ejemplo anterior ha quedado de manifiesto que con Schematron se consigue algo que no podemos hacer con XML Schema 1.0: modelar restricciones entre distintos elementos. Utilizando XML Schema podemos definir de una manera sencilla y precisa la estructura de un documento XML (qué elementos aparecen, en qué orden, qué valores pueden tener a nivel individual, etc.), pero no podemos expresar relaciones ni restricciones entre distintos elementos. XML Schema y Schematron son lenguajes que se complementan muy bien y juntos hacen de la aplicación una herramienta más potente y poderosa.

5.2. Prevención dinámica de errores y aviso de errores

La aplicación permite el uso de Schematron, ya con él podemos expresar las restricciones y relaciones existentes entre elementos distintos. Aunque al comienzo de la aplicación el usuario solamente está obligado a elegir un XML Schema, si elige también un Schematron, la aplicación se encargará de garantizar que el documento XML finalmente generado sea también válido respecto al Schematron. En el ámbito de la aplicación la información proporcionada en el Schematron se usa principalmente para dos propósitos: advertir al usuario cuándo el estado del XML no es válido respecto del Schematron y evitar de manera anticipada que ocurran errores de Schematron cuando sea posible.

La advertencia de errores era el uso más evidente del Schematron en la aplicación, esta funcionalidad simplemente advierte al usuario de las restricciones del Schematron que no se cumplen mostrándole el mensaje asociado. En este sentido se hace un uso del Schematron similar al que se haría en la validación XML tradicional, con la peculiaridad de que

en la aplicación el documento XML va evolucionando con la interacción del usuario, por lo que también cambian o desaparecen los errores relativos al Schematron.

Por otro lado, en esta aplicación se ha querido ir más allá de simplemente mostrar los errores relativos al Schematron por las restricciones que no se cumplen, y se ha añadido una prevención dinámica de errores. La funcionalidad de prevención dinámica de errores trata de evitar errores antes de que se produzcan, para ello, impide que el usuario seleccione opciones de las enumeraciones que causarían un error del Schematron. Se trata de una prevención porque ocurre de manera anticipada a la acción del usuario, impidiendo que cometa errores, y es dinámica puesto que se adapta al estado cambiante del documento XML.

Esta prevención de errores está basada en comprobar qué casos futuros causarían un error. Para ello se prueban los valores de las enumeraciones como si fuesen seleccionados, alterando temporalmente el estado del documento XML, si dado ese estado se generan errores de Schematron, entonces se deshabilita esa opción, impidiendo que el usuario la seleccione, porque en caso de hacerlo llegaría a un estado erróneo. Se ha decidido realizar la prevención de errores analizando solamente los posibles casos futuros provocados por cambios en las enumeraciones y se han descartado otros casos por los siguientes motivos:

- El espacio de casos futuros de una enumeración esta definido y corresponde al conjunto de valores que puede tener. En otros elementos, como por ejemplo los valores numéricos, existen infinitos valores posibles, y por tanto infinitos casos futuros posibles, por lo que no se pueden analizar todos ellos.
- Modificar el valor de una enumeración es un cambio de orden temporal constante (*O(1)*) en un documento XML, ya que simplemente consiste en alterar el contenido de un nodo DOM, por lo que es muy rápido.
- La prevención de errores de Schematron sobre los valores de las enumeraciones es muy útil y sirve de gran ayuda al usuario, ya que su uso es muy común. Todavía es más útil cuando una enumeración tiene muchos posibles valores pero pocos de ellos llevarían a un estado válido.
- Analizar otros posibles casos futuros supondría un deterioro en el rendimiento. Por ejemplo, si analizamos el caso futuro de activar un elemento opcional tendríamos que parsear dicho elemento, lo que podría causar un gran sobrecoste temporal, ya que el elemento puede ser arbitrariamente complejo.
- El interés de otros casos es muy reducido. Por ejemplo podríamos analizar los posibles casos futuros de que un usuario reordenara los elementos de un All, pero este caso no presenta interés ya que es poco probable que el intercambio de posición de dos nodos cause un error, ya que si así fuese no se hubiera utilizado un elemento de tipo *xs:all* en el Schema para definir el elemento.

En la figura 5.2 se esquematiza el proceso de prevención de errores de Schematron que se sigue en la aplicación, y en la figura 5.3 se presenta un ejemplo real de uso de la aplicación utilizando el XML Schema y la regla de Schematron contenida en la figura 5.1.

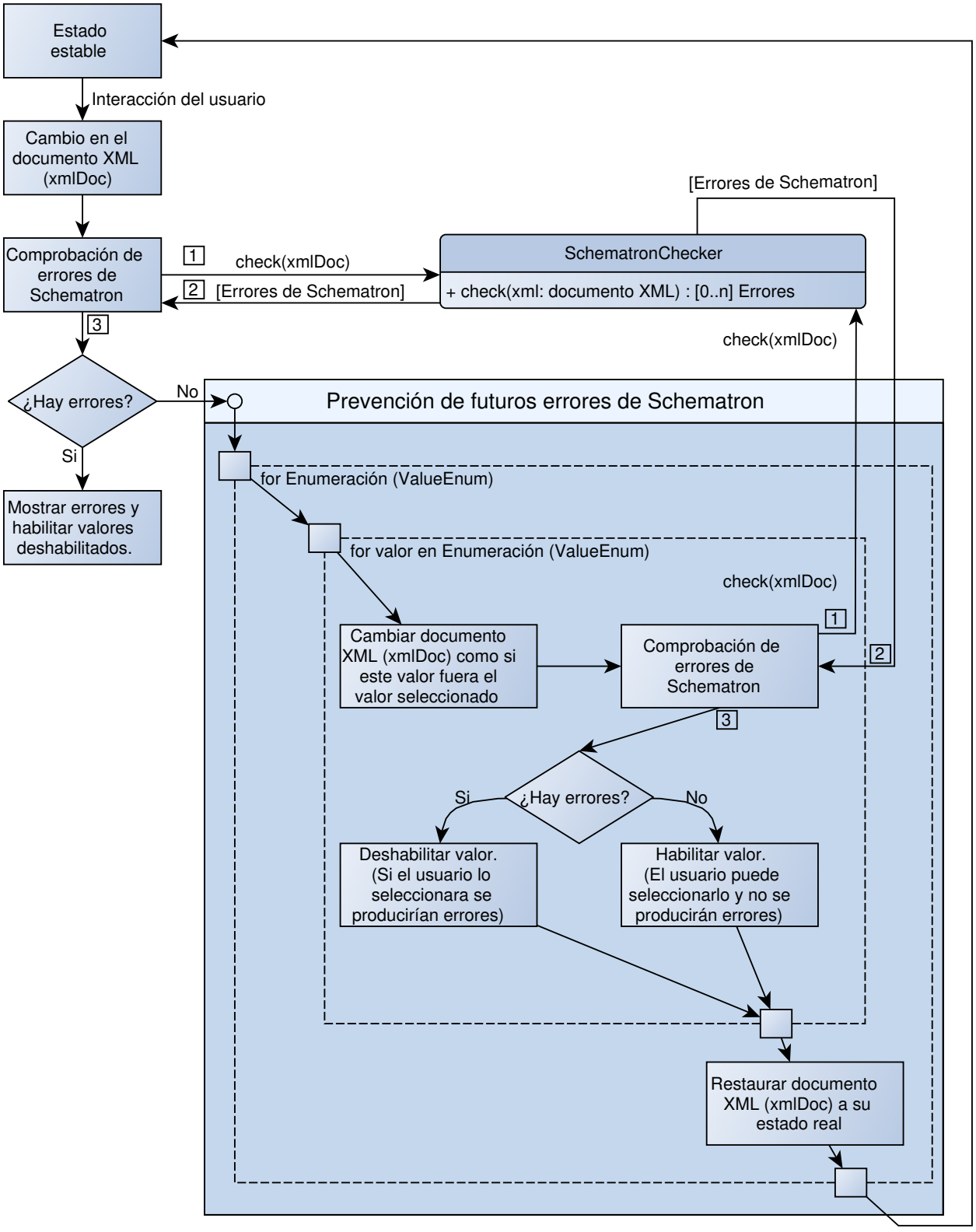


Figura 5.2: Diagrama del proceso que se sigue en la aplicación para evitar futuros errores de Schematron.

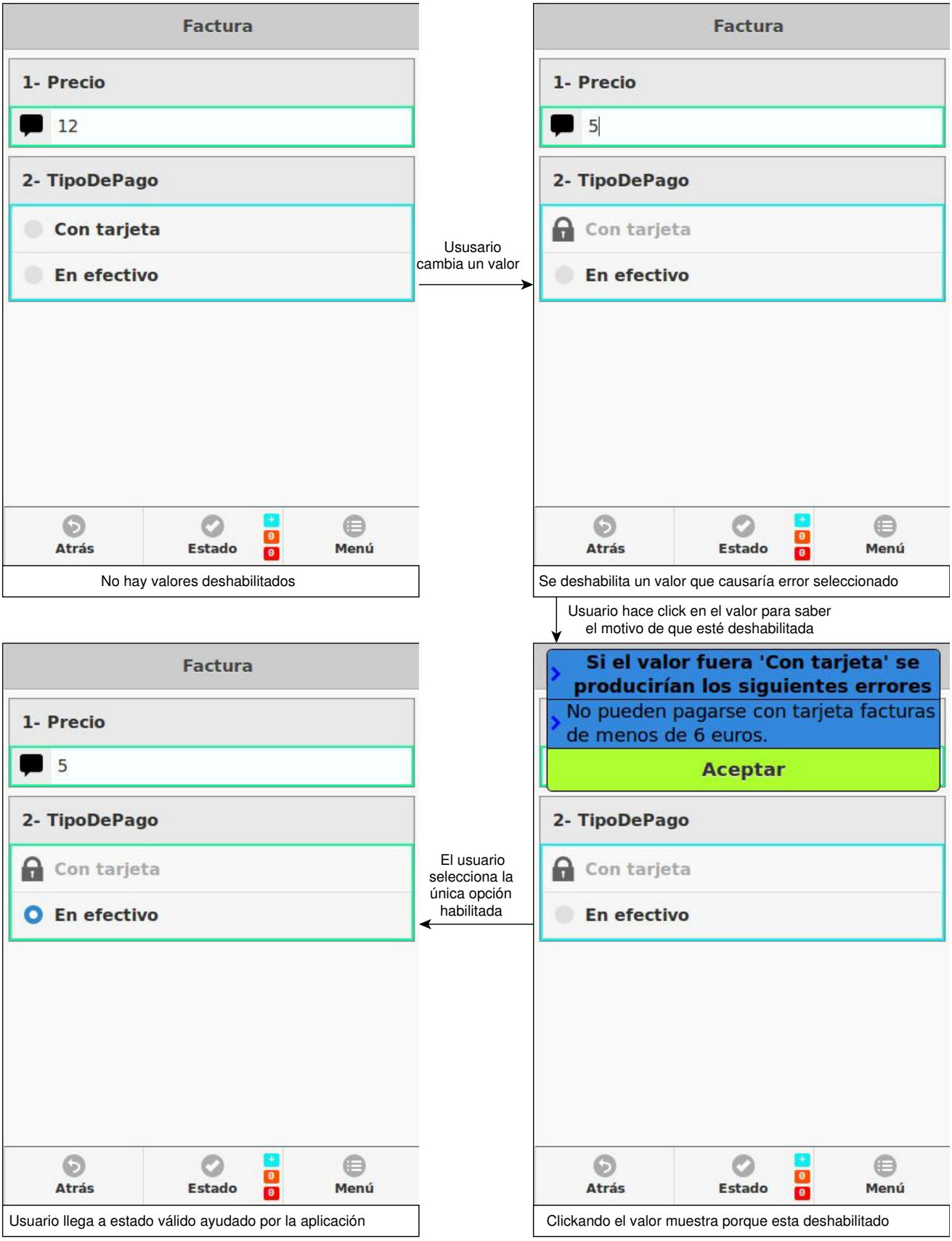


Figura 5.3: Ejemplo real del uso de la prevención dinámica de errores de Schematron en la aplicación. El Schema que da lugar a estas pantallas y la regla de Schematron que da lugar a esta prevención de errores son los de la figura 5.1.

Capítulo 6

Transformaciones en la aplicación

Tal y como se especificó en la propuesta de este trabajo el objetivo de la aplicación a desarrollar debería ser obtener un documento XML válido respecto a un XML Schema y Schematron si lo hubiera, no obstante finalmente se han ampliado las posibilidades de manera que la aplicación permite al usuario obtener otros tipos de documentos, tales como HTML o PDF, utilizando transformaciones y XSLT sobre el documento XML. El presente capítulo explica los motivos del uso de las transformaciones y las posibilidades que ofrece.

XSLT es un lenguaje basado en XPath que permite transformar documentos XML a otros formatos o documentos. La aplicación desarrollada permite al usuario seleccionar un documento XSLT para obtener como resultado el documento XML transformado a otro documento. Esto sirve para que el usuario pueda obtener todo tipo de ficheros sin perder la facilidad de uso y la riqueza semántica que aportan XML Schema y Schematron a la aplicación.

PDF es uno de los formatos de documento más utilizados hoy en día, esto, unido a que muchos usuarios no quieren un documento XML como resultado sino un PDF, ha llevado a la creación de un servicio web que permite transformar documentos HTML a PDF. Por tanto la aplicación puede utilizar el XSLT para transformar el documento XML a un documento HTML con CSS y después realizar una petición al servicio web creado para obtener un PDF.

Teniendo en cuenta que la aplicación plantea un uso real en el ámbito de la dermatología a partir de los documentos XML Schema y Schematron desarrollados en [Bue14], se ha diseñado una hoja XSLT que transforma el documento XML a un documento HTML. Por tanto se puede transformar a PDF usando el servicio web creado, y obteniendo un formato mucho más usable desde el punto de vista del usuario.

La figura 6.1 resume los documentos que se pueden obtener en la aplicación así como su proceso de obtención. La figura 6.2 muestra un ejemplo del documento PDF que se obtiene en la aplicación utilizando el documento XML de la figura 6.4 y la hoja de la figura 6.3, creada para ser usada conjuntamente con los documentos desarrollados en [Bue14].

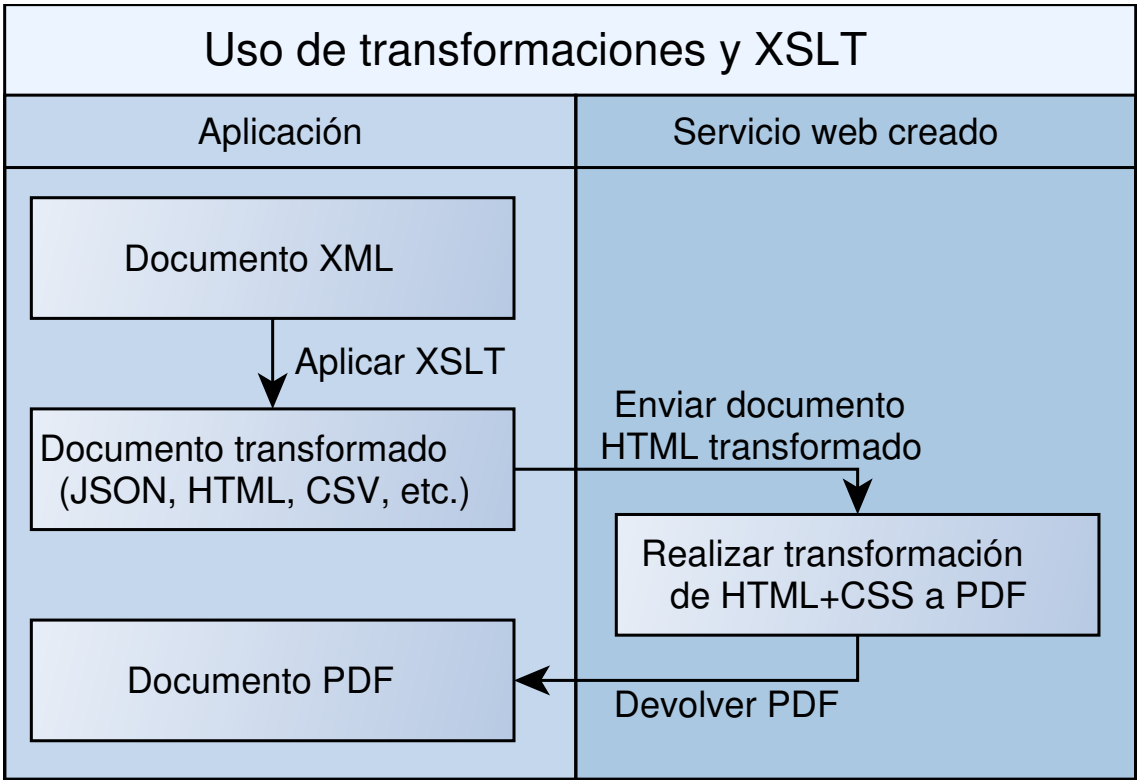


Figura 6.1: Diagrama que muestra los tipos de documentos que se pueden obtener en la aplicación y el proceso de obtención mediante transformaciones XSLT y el uso del servicio web creado.

Farmacia: datos de la farmacia (dirección/teléfono/...)

FÓRMULA MAGISTRAL

DATOS DEL PACIENTE

INFORMACIÓN PERSONAL				
NOMBRE	**EDAD**	**EMBARAZO**	**ALERGIAS**	**INTOLERANCIA_EXCIPIENTES**
Sergio Frago	22	No	No	No
INFORMACIÓN RELATIVA A LA PATOLOGÍA				
LOCALIZACIÓN	**ESTADIO DE LA LESIÓN**			
CueroCabelludo	Androgenetica Masculina			

DATOS DE LA FÓRMULA

VEHÍCULO	
Espuma capilar	
COMPOSICIÓN	
GRUPO - PRINCIPIO ACTIVO	**RANGO_DOSIFICACIÓN_(%)**
Irritantes-Ditranol	0.63
Sensibilizantes-Difenciprona	0.36008

INFORMACIÓN ADICIONAL
Facultativo prescriptor:
Administración:
Conservación:
Caducidad:
Comentarios:

	FECHA		
	DÍA	**MES**	**AÑO**
Firma	·	·	·

Figura 6.2: Ejemplo de documento PDF que se obtiene en la aplicación transformando el documento XML de la figura 6.4 generado usando los documentos creados en [Bue14] y la hoja XSLT de la figura 6.3 desarrollada como parte de este proyecto.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
    <html>

<head>
    <meta charset="utf-8" />
    <title>title</title>
    <style type="text/css">
    #mainHeader {
        font-size: 18pt;
        font-weight: bold;
        text-align: center;
        margin: 0mm auto 10mm
    }
    .superHeader {
        background: #444;
        color: white;
        font-size: 14pt;
        margin-bottom: 4mm;
    }
    .normalHeader {
        background: #808080;
    }
    .bordered, table, th, td {
        border: 1pt solid black;
        border-collapse: collapse;
        text-align: center;
    }
    #pharmacy {
        border-bottom: 1pt dotted black;
        color: #DDD;
        font-size: 12pt;
    }
    table{
        width: 100%;
    }
    </style>
</head>

<body>
    <!-- Zona para los datos de la farmacia -->
    <div id="mainHeader">
        Farmacia: <span id="pharmacy"> datos de la farmacia (dirección/teléfono/...) </span>
        <br/>
        FÓRMULA MAGISTRAL
    </div>

    <!-- Zona para los datos del paciente -->
    <div class="superHeader bordered">DATOS DEL PACIENTE</div>
    <table>
        <tr class="normalHeader"><th colspan="5">INFORMACIÓN PERSONAL</th></tr>
        <tr><th style="width:100%">NOMBRE</th><th>EDAD</th><th>EMBARAZO</th><th>ALERGIAS</th><th>INTOLERANCIA_EXCIPIENTES</t
        <tr>
            <td><xsl:value-of select="/Enfermedad/DatosPaciente/Nombre"/></td>
            <td><xsl:value-of select="/Enfermedad/DatosPaciente/Edad"/></td>
            <td><xsl:value-of select="/Enfermedad/DatosPaciente/Embarazo"/></td>
            <td><xsl:value-of select="/Enfermedad/DatosPaciente/Alergias"/></td>
            <td><xsl:value-of select="/Enfermedad/DatosPaciente/IntoleranciaExcipientes"/></td>
        </tr>
        <tr class="normalHeader"><th colspan="5">INFORMACIÓN RELATIVA A LA PATOLOGÍA</th></tr>
        <tr><th>LOCALIZACIÓN</th><th colspan="4">ESTADIO DE LA LESIÓN</th></tr>
        <tr>
            <td colspan="1"><xsl:value-of select="/Enfermedad/Formula/Localizacion"/></td>
            <td colspan="4"><xsl:value-of select="/Enfermedad/Formula/EstadioLesion"/></td>
        </tr>
    </table>

    <br/>
    <br/>
```

```
    <!-- Zona para los datos de la fórmula -->
    <div class="superHeader bordered">DATOS DE LA FÓRMULA</div>
    <table>
        <tr class="normalHeader"><th colspan="2">VEHÍCULO</th></tr>
        <tr><td colspan="2"><xsl:value-of select="/Enfermedad/Formula/Vehiculo"/></td></tr>
        <tr class="normalHeader"><th colspan="2">COMPOSICIÓN</th></tr>
        <tr><th>GRUPO - PRINCIPIO ACTIVO</th><th>RANGO_DOSIFICACIÓN_(%)</th></tr>
        <!-- EJEMPLO: <tr><td>Nombre grupo</td><td>Cantidad</td></tr> -->
        <!-- Para cada grupo -->
        <xsl:for-each select="/Enfermedad/Formula/Grupos/*">
            <!-- Para cada principio activo. Gr-Pr   0.2 -->
            <xsl:for-each select="./*">
               <tr>
                   <td><xsl:value-of select="name(..)"/>-<xsl:value-of select="name(.)"/></td>
                   <td><xsl:value-of select="."/></td>
               </tr>
           </xsl:for-each>
        </xsl:for-each>
    </table>

    <br/>
    <br/>

    <!-- Zona para información adicional -->
    <table>
        <tr class="normalHeader"><th>INFORMACIÓN ADICIONAL</th></tr>
        <tr><th style="text-align:left">Facultativo prescriptor:</th></tr>
        <tr><th style="text-align:left">Administración:</th></tr>
        <tr><th style="text-align:left">Conservación:</th></tr>
        <tr><th style="text-align:left">Caducidad:</th></tr>
        <tr><th style="text-align:left;height:6em;vertical-align:top">Comentarios:</th></tr>
    </table>

    <br/>

    <!-- Zona para firma (o sello??) y fecha -->
    <table>
        <tr><td rowspan="3" style="width:100%;vertical-align:bottom">Firma</td><th class="normalHeader" colspan="3">FI
        <tr><th>DÍA</th><th>MES</th><th>AÑO</th></tr>
        <tr>
            <td>·
            <!-- <xsl:value-of  select="day-from-date(current-date())"/> -->
            </td>
            <td>·
            <!-- <xsl:value-of  select="month-from-date(current-date())"/> -->
            </td>
            <td>·
            <!-- <xsl:value-of  select="year-from-date(current-date())"/> -->
            </td>
        </tr>
    </table>
</body>

</html>
</xsl:template>
</xsl:stylesheet>
```

Figura 6.3: Hoja XSLT creada para ser usada en el ámbito de la dermatología junto con los XML Schemas y Schematron desarrollados en [Bue14].

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Enfermedad>
   <DatosPaciente>
      <Nombre>Sergio Frago</Nombre>
      <Edad>
         <Adultos>22</Adultos>
      </Edad>
      <Alergias>No</Alergias>
      <Embarazo>No</Embarazo>
      <IntoleranciaExcipientes>No</IntoleranciaExcipientes>
   </DatosPaciente>
   <Formula>
      <Localizacion>CueroCabelludo</Localizacion>
      <EstadioLesion>Androgenetica Masculina</EstadioLesion>
      <Vehiculo>Espuma capilar</Vehiculo>
      <Grupos>
         <Irritantes>
            <Ditranol>0.63</Ditranol>
         </Irritantes>
         <Sensibilizantes>
            <Difenciprona>0.36008</Difenciprona>
         </Sensibilizantes>
      </Grupos>
   </Formula>
</Enfermedad>
```

Figura 6.4: Ejemplo de documento obtenido en la aplicación utilizando los XML Schemas y Schematron desarrollados en [Bue14].

Capítulo 7

Conclusiones y trabajo futuro

Este capítulo se centra en las conclusiones finales del proyecto en cuanto a los resultados obtenidos y al planteamiento de posibles trabajos futuros que utilicen los resultados de este proyecto.

7.1. Resultados y conclusiones

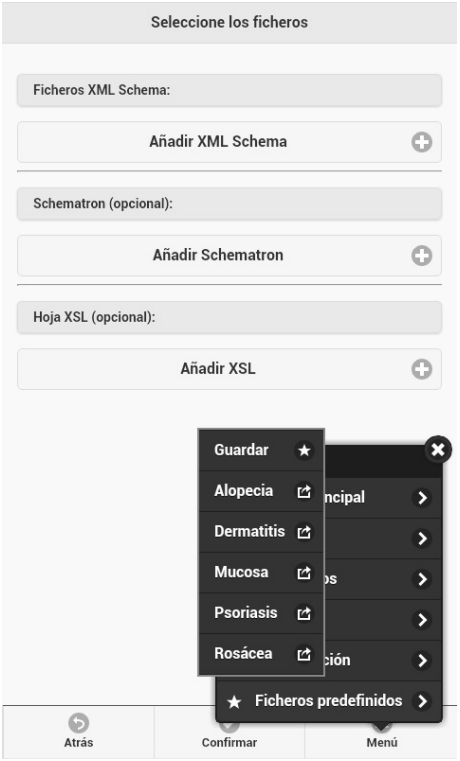
Los resultados de este TFG más importantes son los siguientes:

- Se ha desarrollado una aplicación multiplataforma, que funciona tanto en plataformas móviles como en entornos de escritorio, y que permite crear de documentos XML válidos a partir de XML Schema y opcionalmente Schematron.
- Se ha probado y adaptado el funcionamiento de la aplicación para su uso en dermatología utilizando los Schemas y Schematrones desarrollados en [Bue14].
- Se ha comprobado el funcionamiento de la aplicación en varias plataformas móviles y en navegadores de escritorio, tal y como se explicará más adelante.
- Se ha creado una hoja XSLT para dichas formulaciones magistrales de dermatología que permite transformar el documento XML a un documento HTML con CSS, permitiendo al usuario tener un estilo más adecuado para la visualización.
- Se ha creado un servicio web que permite obtener un documento PDF a partir de un documento HTML con CSS, lo que permite al usuario obtener como resultado un tipo de documento de uso más generalizado y adecuado a sus necesidades de visualización.

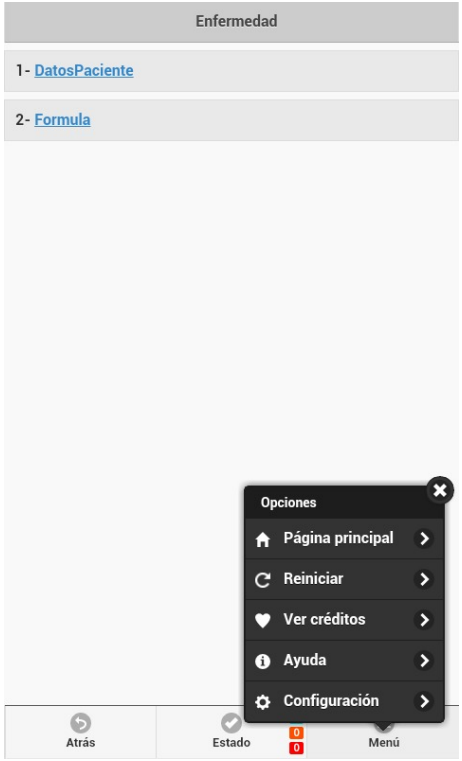
Las plataformas, dispositivos y navegadores en los que se ha comprobado el funcionamiento de la aplicación son las siguientes:

- Plataforma móvil *Android*: comprobado en *Samsung Galaxy Core Plus* con *Android* versión 4.2.2, *OnePlus one* con *Cyanogen 11S* y *BQ Edison2* con *Android* 4.1 *Jelly Bean*.
- Plataforma móvil *FirefoxOS*: comprobado en *Alcatel One Touch Fire c* con *FirefoxOS* versión 1.3.
- Navegadores web de escritorio: comprobado en *Firefox* versión 31, *Google Chrome* versión 38 y *Safari* versión 7.1.3.

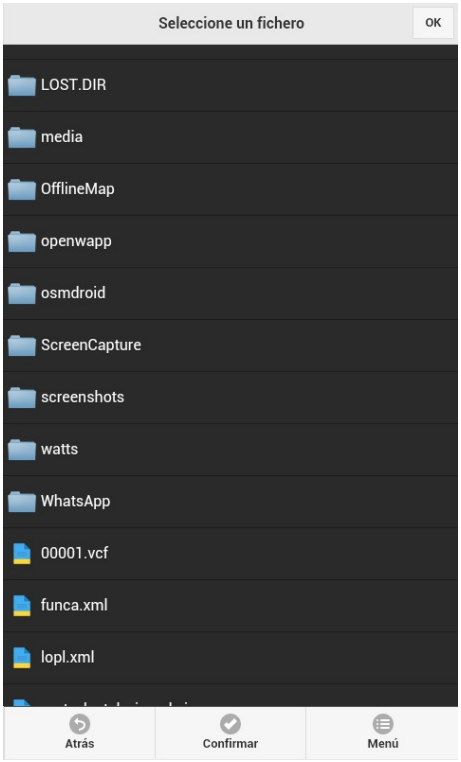
La aplicación mantiene una visualización uniforme en todas las plataformas, siendo el acceso al sistema de ficheros el único aspecto del código (y de la visualización) que cambia entre plataformas. En las figuras 7.1, 7.2 y 7.3 se muestran algunas pantallas reales de la aplicación durante su uso en un *Samsung Galaxy Core Plus* para generar una fórmula magistral para un paciente con *Alopecia*. Como se ha comentado anteriormente, estos Schemas y Schematrones utilizados han sido definidos previamente en [Bue14] y se corresponden a casos reales llevados a cabo en colaboración con dermatólogos y farmacéuticos.



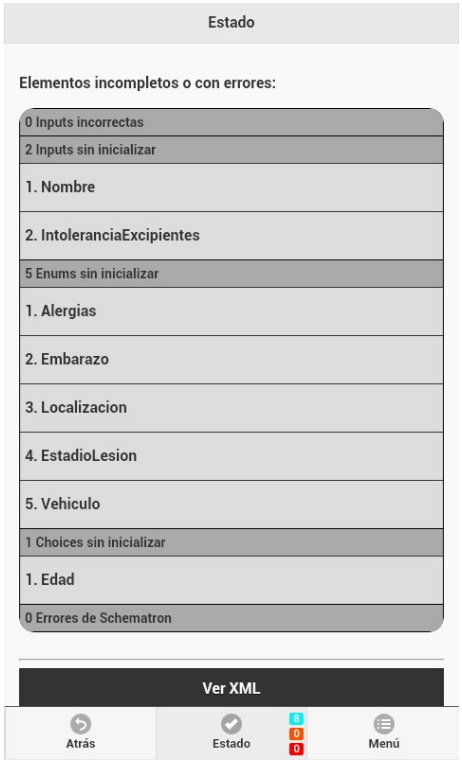
(a) Pantalla inicial de selección de ficheros y sub-menú de ficheros predefinidos.



(b) Menú principal y página inicial del formulario para la opción predefinida de la patología *Alopecia*.

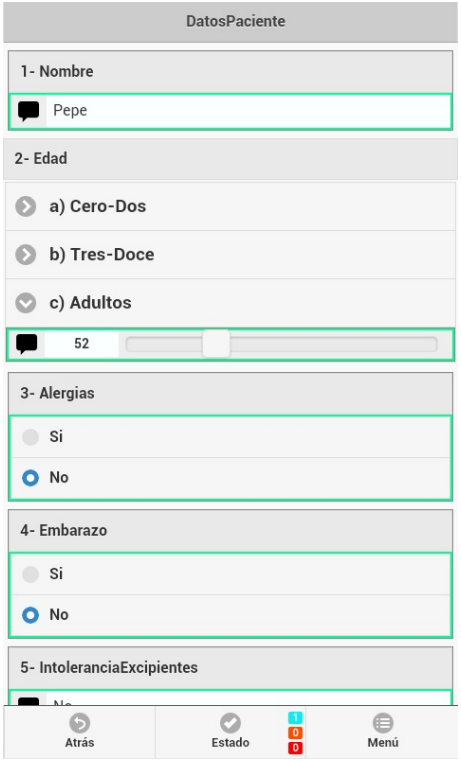


(c) Selector de ficheros integrado en la aplicación.

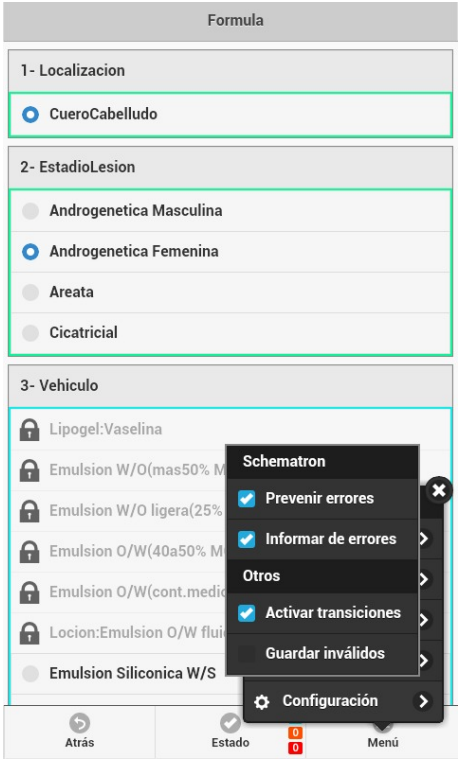


(d) Página de estado: elementos del formulario que quedan por completar.

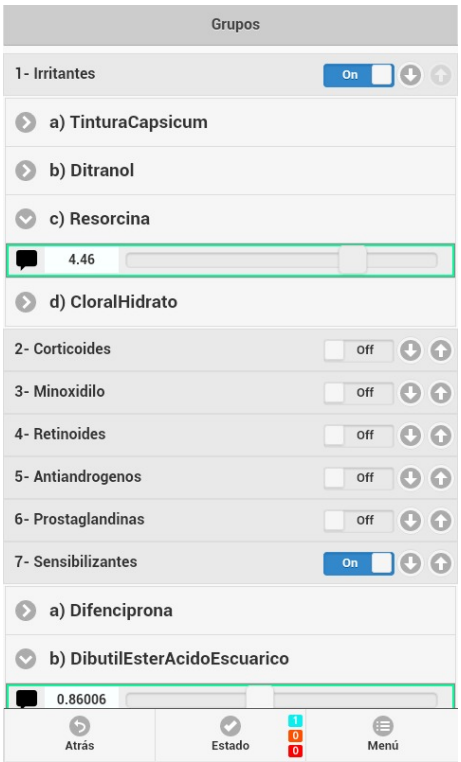
Figura 7.1: Capturas de la aplicación en un *Samsung Galaxy Core Plus*



(a) Página del elemento del formulario relativo a los datos del paciente.



(b) Submenú de configuración y página del elemento del formulario relativo a la fórmula.



(c) Página del elemento del formulario relativo a los grupos de principios activos.

(d) Página de estado: mostrar estado del documento XML.

Figura 7.2: Capturas de la aplicación en un *Samsung Galaxy Core Plus*

(a) Página de estado: mostrar estado del documento XML transformado con XSLT como HTML.

- Muestra los elementos del formulario no completados y da un acceso directo a ellos.
- Muestra los errores de Schematron actuales.
- Permite ver el estado del documento XML.
- Permite guardar el documento XML.
- Permite ver el XML transformado como texto.
- Permite guardar el documento transformado.
- Permite ver el XML transformado como HTML.
- Permite guardar un PDF obtenido a partir del HTML del documento transformado.

(b) Funcionalidades de la página de estado

Figura 7.3: Captura de la aplicación en un *Samsung Galaxy Core Plus* y funcionalidades de la página de estado.

7.2. Trabajo futuro

Un trabajo futuro claro consistiría en añadir soporte en la aplicación para un subconjunto mayor de los elementos de XML Schema y/o Schematron. Dichos subconjuntos a los que da soporte la aplicación detallan en el anexo A. Adicionalmente, dada la genericidad de la aplicación, es evidente que podría ser usada por otros usuarios.

Otro trabajo futuro próximo planificado es la validación clínica por parte de los dermatólogos para su uso en formulación magistral. Aunque el resultado final que se ha conseguido es visualmente atractivo, tendrán que ser los especialistas los que expresen sus opiniones sobre la usabilidad del sistema, posibles mejoras, etc. que se deberían incluir en la aplicación para su uso en la práctica clínica habitual.

Otra vía de ampliación, que ya está puesta sobre la mesa como futuro TFG, consiste en el desarrollo de una aplicación que permite modelar de manera visual Schemas y Schematrones para dermatología, que usada en conjunción con la aplicación aquí desarrollada permitiría al dermatólogo cubrir el proceso completo de creación de fórmulas magistrales, desde la definición de nuevas patologías hasta la obtención de fórmulas magistrales.

Bibliografía

[BPSM+06] Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, and Cowan, *Extensible markup language (xml) 1.1*, W3C recommendation, W3C, 2006, http://www.w3.org/TR/xml11.

[Bue14] Victoria Mingote Bueno, *Definición de xml schemas para la prescripción de fórmulas magistrales en dermatología*, Master's thesis, Universidad de Zaragoza, Escuela de Ingeniería y Arquitectura, 2014, http://deposita.unizar.es/record/16637?ln=es.

[Cla99] Clark, *Xsl transformations (xslt)*, W3C recommendation, W3C, 1999, http://www.w3.org/TR/xslt.

[dav14] davidmoten, *xsd-forms*, 2014, https://github.com/davidmoten/xsd-forms.

[NS14] Nitobi and Adobe Systems, *Phonegap*, 2014, http://phonegap.com/.

[Rau10] Rein Raudjärv, *dynaform*, 2010, https://github.com/reinra/dynaform.

[sch06] *Information technology — document schema definition languages (dsdl) — part 3: Rule-based validation — schematron*, Tech. report, ISO/IEC, 2006, http://www.schematron.com/.

[Sof] Ilerian Software, *Xsdform*, http://www.ilerian.com/xsd-web-form-overview.

[xpa04] *Xml path language (xpath)*, W3C recommendation, W3C, 2004, http://www.w3.org/TR/xpath/.

[xsd04a] *Xml schema part 0: Primer*, W3C recommendation, W3C, 2004, http://www.w3.org/TR/xmlschema-0/.

[xsd04b] *Xml schema part 1: Structures*, W3C recommendation, W3C, 2004, http://www.w3.org/TR/xmlschema-1/.

[xsd04c] *Xml schema part 2: Datatypes*, W3C recommendation, W3C, 2004, http://www.w3.org/TR/xmlschema-2/.

Anexos

Apéndice A

Subconjuntos de XML Schema y Schematron soportados

La aplicación desarrollada no da soporte a toda la especificación de XML Schema y Schematron, sino que solamente soporta un subconjunto de sus elementos. Se ha intentado que los elementos a los que se dan soporte sean los más útiles y los más utilizados.

Subconjunto de elementos de XML Schema a los que se da soporte:

- *xs:schema*
- *xs:include*
- *xs:element*
- *xs:attribute*
- *xs:all*
- *xs:choice*
- *xs:sequence*
- *xs:complexType*
- *xs:simpleType*
- *xs:restriction* incluyendo todas las restricciones sobre tipos simples (*xs:minInclusive*, *xs:enumeration*, etc.).
- *xs:list*
- *xs:union*
- *xs:simpleContent* solo para extender un tipo simple añadiéndole atributos.

Subconjunto de elementos de Schematron a los que se da soporte:

- *sch:schema*
- *sch:pattern*
- *sch:rule*

- *sch:assert*
- *sch:report*
- *sch:value-of*
- *sch:name*

Apéndice B

Correspondencia entre elementos del formulario de la aplicación y elementos del XML Schema

En este anexo se explican para cada tipo de elemento del formulario y los elementos del Schema a los que está asociado. Junto con los ejemplos de XML Schema se adjuntarán también capturas de la aplicación para dar una idea más clara sobre dichas asociaciones y su modo de presentación al usuario. Hay que tener en cuenta que esto son solamente ejemplos, pero la aplicación se adapta dinámicamente al Schema proporcionado, y por ejemplo, soporta el resto de tipos simples, entre los que se incluyen los definidos por restricción, lista, unión, derivación de tipos primitivos o globales, etc.

La siguiente lista contiene los tipos de elementos del formulario (clase FormElement) y los elementos del Schema a los que están asociados acompañados de ejemplos:

Tipos simples

Están asociados al elemento del XML Schema *xs:simpleType*. Representan un tipo simple, es decir el texto de un nodo. La aplicación presenta al usuario tres visualizaciones diferentes que se adaptan a tres de los tipos de datos más utilizados:

1. **Enumeración o *ValueEnum*** Se utilizan cuando un elemento puede tener un valor de entre un conjunto de valores. Permiten al usuario seleccionar fácilmente un valor de entre los valores del conjunto válido. Ejemplo en figura B.1.

```
<xs:simpleType>
    <xs:restriction base="xs:int">
        <xs:minInclusive value="0" />
        <xs:maxInclusive value="120" />
    </xs:restriction>
</xs:simpleType>
```

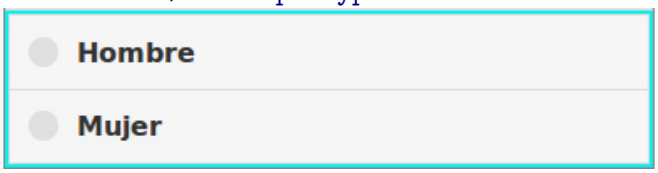


Figura B.1: Ejemplo de parte de un XML Schema y vista del *ValueEnum* que produce.

2. **Sliders o *ValueInput* con slider** Se utilizan cuando el tipo de dato es numérico y cuenta con un máximo y un mínimo. Permiten al usuario introducir con facilidad un valor numérico dentro del rango. También le permite escribir un valor libremente para obtener una precisión mayor si lo desea. Ejemplo en figura B.2.

```
<xs:simpleType>
    <xs:restriction base="xs:int">
        <xs:minInclusive value="0" />
        <xs:maxInclusive value="120" />
    </xs:restriction>
</xs:simpleType>
```

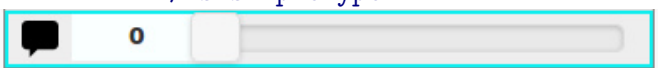


Figura B.2: Ejemplo de parte de un XML Schema y vista del *ValueInput-slider* que produce.

3. **Input o *ValueInput*** Se utilizan cuando un elemento de tipo simple no se encuentra dentro de ninguno de los dos grupos anteriores. Permiten al usuario escribir un valor libre. Ejemplo en figura B.3.

```
<xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:enumeration value="Hombre" />
        <xs:enumeration value="Mujer" />
    </xs:restriction>
</xs:simpleType>
```

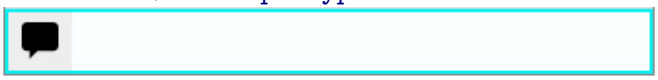


Figura B.3: Ejemplo de parte de un XML Schema y vista del *ValueInput* que produce.

Elemento XML o *XMLElement*

Esta asociado al elemento del Schema *xs:element*. Representa un nodo de tipo elemento del documento XML. Puede ser de tipo simple si solo tiene un nodo de texto

o complejo si tiene contenido. Por ejemplo, en el ejemplo de la figura B.4 Nombre, Edad y Sexo son elementos XML simples, puesto que simplemente tienen un valor, mientras que Persona es un elemento de tipo complejo, puesto que está compuesto por otros elementos (Nombre, Edad y Sexo).

```
<xs:element name="persona">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Nombre" type="xs:string" />
            <xs:element name="Edad">
                <xs:simpleType>
                    <xs:restriction base="xs:int">
                        <xs:minInclusive value="0" />
                        <xs:maxInclusive value="120" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="Sexo">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Hombre" />
                        <xs:enumeration value="Mujer" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

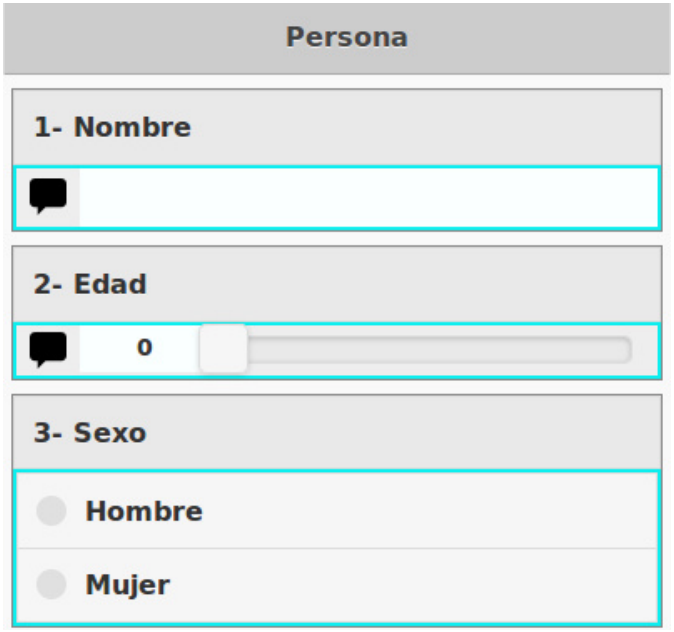


Figura B.4: Ejemplo de parte de un XML Schema y vista del *XMLElement* que produce.

Secuencia o *Sequence*

Está asociado al elemento del Schema *xs:sequence*. Representa un elemento que especifica que sus hijos deben aparecer en el orden especificado en la secuencia. El ejemplo anterior de la figura B.4 contiene una secuencia. El orden definido por la secuencia debe reproducirse en el documento XML para considerarse válido, es decir, dentro del elemento XML Persona, debe aparecer primero el elemento Nombre, después el elemento Edad y por último el elemento Sexo.

Elemento sin orden o *All*

Está asociado al elemento del Schema *xs:all*. Representa un elemento que especifica que sus hijos pueden aparecer en cualquier orden. En el siguiente ejemplo, el documento XML sería válido independientemente del orden de Nombre y Edad en Persona. Por ello, cuando se utiliza este elemento, la aplicación permite al usuario reordenar los elementos presentes a su antojo tal y como se muestra en la figura B.5.

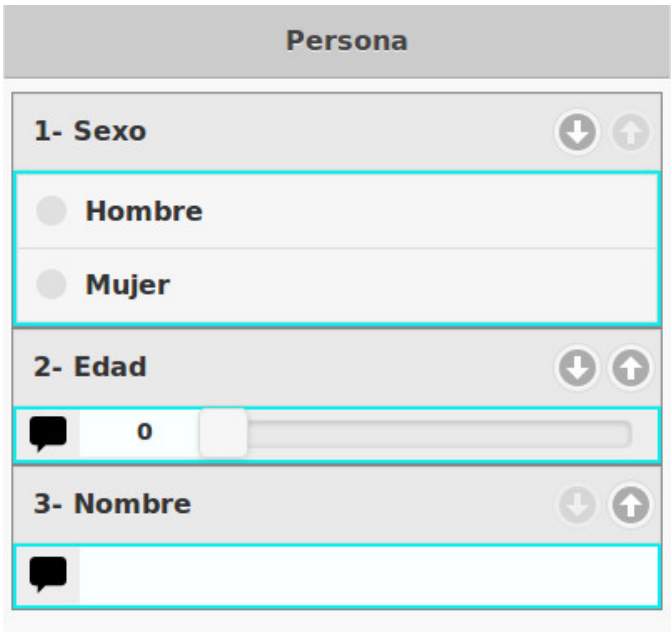


Figura B.5: Ejemplo de un *All* producido por un XML Schema idéntico al de la figura B.4 utilizando *xs:all* en lugar de *xs:sequence* y tras mover algunos elementos.

Selección o *Choice*

Está asociado al elemento del Schema *xs:choice*. Representa que uno y solo uno de sus elementos debe aparecer en el documento XML válido. Para representar este elemento solamente se permite que se seleccione uno de sus hijos, y por tanto que solo uno de sus hijos esté en el documento XML. Obsérvese la diferencia de que en un Choice hay que elegir uno entre varios elementos arbitrariamente complejos, mientras que en un ValueEnum hay que elegir un valor de texto entre un conjunto de varios valores. La figura B.6 muestra un ejemplo de este tipo de elemento.

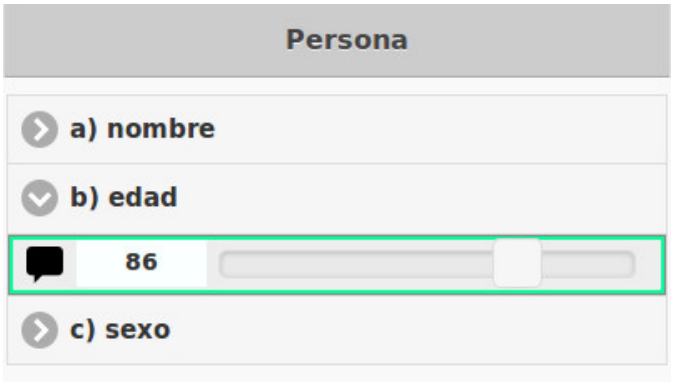


Figura B.6: Ejemplo de un *Choice* producido por un XML Schema idéntico al de la figura B.4 utilizando *xs:choice* en lugar de *xs:sequence* y tras seleccionar su segundo elemento y darle valor.

Atributo o *Attribute*

Está asociado a un elemento del Schema *xs:attribute*. Representa un atributo de un elemento del documento XML. Su valor corresponde a un tipo simple similar a los explicados anteriormente. Su vista es similar a la de un elemento simple puesto que su valor se corresponde con un tipo simple.

Lista o *List*

Representa otro elemento del formulario (correspondiente a *xs:element*, *xs:sequence*, *xs:choice*, *xs:all* o *xs:attribute*) que o bien es opcional, o bien puede repetirse varias veces. Por defecto el valor de la cardinalidad de los elementos (atributos *maxOccurs* y *minOccurs*) es 1-1. Aunque la implementación de la lista es la misma existen visualizaciones de dos tipos para facilitar las cosas al usuario:

1. **Opcional** El elemento puede aparecer 0 o 1 veces. En la figura B.7 se da un ejemplo en el que el elemento Edad es opcional, este sería el equivalente si el elemento Edad tuviera el atributo *minOccurs=0* en el Schema de la figura B.4.

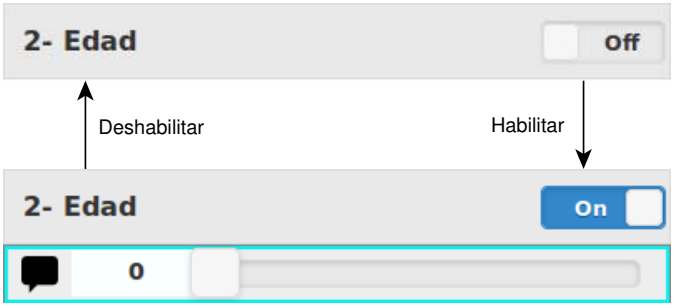


Figura B.7: Ejemplo de la vista de un List opcional. Se muestra que el usuario puede interaccionar con el elemento y activarlo o desactivarlo cuando quiera.

2. **Común** El elemento puede aparecer otro número de veces (diferente de 0-1 y 1-1). En la figura B.8 se da un ejemplo en el que el elemento nombre es repetible, este sería el equivalente si el elemento Nombre tuviera los atributos *minOccurs=2* y *maxOccurs=5* en el Schema de la figura B.4.

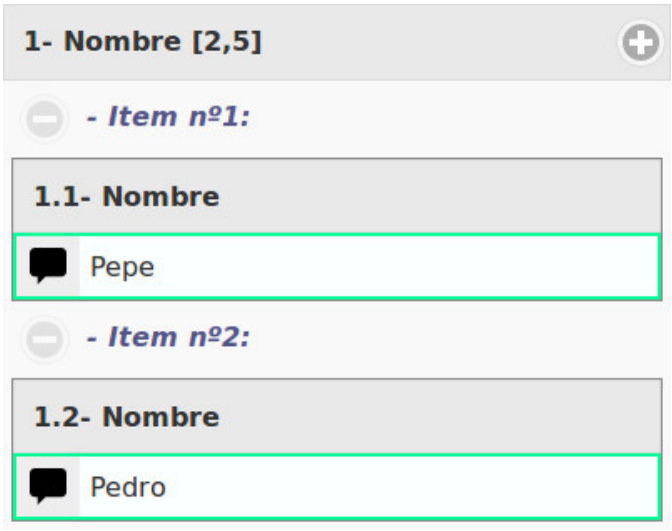


Figura B.8: Ejemplo de la vista de un List común. El usuario puede interaccionar con el elemento y añadir o quitar elementos de la lista dentro de los límites de cardinalidad. En este ejemplo están deshabilitados los botones de borrar elemento porque la lista debe de tener como mínimo dos elementos.

Apéndice C

Tutorial y aplicación

Este anexo presenta enlaces a un vídeo con un tutorial de uso de la aplicación y a la propia aplicación para ser usada en android. El vídeo está orientado al uso de la aplicación en el ámbito de la dermatología, aunque puede ser de utilidad para cualquier tipo de usuarios. Dicho vídeo se encuentra disponible en la dirección `https://www.youtube.com/watch?v=D6UEvVZsJPw`. La aplicación para android puede obtenerse de la siguiente dirección: `https://www.dropbox.com/s/il2yu1ee2o6f563/formulatron.apk?dl=0`.

Apéndice D

Créditos a librerías y contenidos de terceros

Este anexo presenta una lista de las librerías y contenidos de terceros utilizadas en la realización de este trabajo. Desde la aplicación desarrollada puede accederse a una lista de créditos similar utilizando la opción créditos del menú, que lleva a una página que contiene información de las librerías, de los autores, de las licencias de cada elemento, etc.

Librerías y contenidos de terceros utilizadas:

Phonegap
Framework usado para crear las aplicaciones móviles multiplataforma a partir de código HTML5, JavaScript y CSS3.
Dirección: `http://phonegap.com/`.
Se ha utilizado los siguientes plugins de Phonegap:

- *org.apache.cordova.file 1.3.2 "File"*: para la API de ficheros.
- *org.apache.cordova.file-transfer 0.4.8 "File Transfer"*: para descargar el PDF.
- *org.apache.cordova.inappbrowser 0.5.4 ÏnAppBrowser"*: para abrir enlaces externos en el navegador.

jQuery
Biblioteca JavaScript (DOM, eventos, animaciones, etc).
Dirección: `http://jquery.com`.

jQuery Mobile
Biblioteca JavaScript para la interfaz gráfica móvil (diseño interfaz, transiciones, iconos, etc.).
Dirección `http://jquerymobile.com`.

jQuery xpath
Plugin que implementa XPath 2.0 para jquery. - Modificado/extendido para evaluar una expresión varias veces parseándola solo una. Utilizado para el XPath del Schematron.
Dirección `https://github.com/ilinsky/jquery-xpath`.

Schemas y Schematrones predefinidos (de Victoria Mingote Bueno)
Los Schemas y Schematrones predefinidos forman parte del Trabajo de Fin de Grado de Victoria Mingote Bueno en la Universidad de Zaragoza. Se les han añadido

pequeñas modificaciones y existe consentimiento por parte de dicha autora para el uso de dichos ficheros en esta aplicación.
Dirección `http://deposita.unizar.es/record/16637?ln=es`.

Mustache.js
Sistema de plantillas JavaScript. Modificado para añadir soporte a variables @index y @indexLetter.
Dirección `https://github.com/janl/mustache.js`.

vkBeautify
Embellecedor (identación, etc.) de código (XML).
Dirección `http://www.eslinstructor.net/vkbeautify`.

highlight.js
Embellecedor (coloreado, resaltado de sintaxis, etc.) de código (XML).
Dirección `https://highlightjs.org`.

FileSaver.js
Implementación de función javascript *saveAs()* para guardar un fichero especificada en el HTML5 W3C.
Dirección `https://github.com/eligrey/FileSaver.js`.

AJAXSLT
Implementación de transformaciones XSL-T.
Dirección `http://goog-ajaxslt.sourceforge.net/`.

Otros iconos
Icono carpeta usado en el selector de ficheros obtenido de `http://findicons.com/icon/64167/folder?id=64346`.
Icono fichero utilizado en el selector de ficheros obtenido de `https://www.iconfinder.com/icons/283040/browser_document_explorer_file_finder_folder_format_pdf_psd_rar_text_windows_explorer_zip_icon#size=128`.
Iconos de ficheros y carpetas usados en firefoxOS obtenidos de `http://dojotoolkit.org`.

dompdf
Para transformar HTML a PDF se utiliza la librería dompdf en el servicio externo creado.
Dirección `https://github.com/dompdf/dompdf`.