1 ÍNDICE

2 TABLA DE ILUSTRACIONES

3 INTRODUCCIÓN

En este capítulo se comenzará haciendo una pequeña reseña sobre el grupo TELSTAR, conjunto de empresas para las cuales este proyecto ha sido desarrollado, para continuar con las motivaciones y objetivos que fueron motivo de su origen.

3.1 La empresa

TELSTAR SA fue fundada en 1963 en Terrassa. En sus primeros años de actividad desarrolló productos y bienes de equipo en el campo de la tecnología de vacío. Al largo de los años ha ido creciendo, diversificando sus actividades y expandiendo las exportaciones. A día de hoy TELSTAR es un potente grupo industrial que está formado por una serie de empresas, cada una de las cuales tiene una actividad diferente a la del resto:

TELSTAR INDUSTRIAL: forma el núcleo duro de todo el grupo. Su principal actividad es el diseño, fabricación y comercialización de bienes equipos y componentes. La comercialización de sus productos se realiza a través de sus tres divisiones especializadas: división tecnología farmacéutica, división equipos de laboratorio, división vacío industrial.

TELSTAR PROJECTS: dedicada a la ingeniería de proyectos e instalaciones "llave en mano". Su especialidad son las salas blancas y los sistemas de agua pura.

TELSTAR INSTRUMAT: es una compañía distribuidora de equipos de alta tecnología, especialmente en campos diversos como la fotometría-radiometría, medidores de presión y caudal, criogenia, análisis y caracterización de superficies y alto vacío para sectores científicos e industriales.

IMA TELSTAR: dedicada al diseño, fabricación y comercialización de equipos de liofilización y soluciones de carga y descarga, secadores de vacío, congeladores de plasma sanguíneo y cámaras de simulación espacial.

SOCIEDAD DE VALIDACION Y SISTEMAS: compañía dedicada a la validación de los principales procesos, instalaciones y sistemas relacionados con la industria farmacéutica, química, alimentaria y veterinaria.

CELESTER TECHNOLOGIES: se dedica a la fabricación y comercialización de equipos de esterilización. Dispone de una amplia gama de autoclaves, generadores de vapor y hornos de esterilización para la industria farmacéutica.

TELSTAR HUADONG: empresa con sede en Shangai (China). Su principal actividad se centra en la fabricación de liofilizadores industriales y en su comercialización en el mercado asiático.

3.2 Motivación

Actualmente en TELSTAR SA el sistema de generación de ofertas se realiza de distintas formas dependiendo de la empresa que se trate. Una manera es a través de formularios creados en documentos Word, programados con macros escritas en Visual Basic. Otro modo es creando documentos a partir de ofertas similares, modificando su contenido de forma manual.

Las ofertas generadas a partir de formularios programados en plantillas de documento Word es un sistema utilizado desde hace varios años, ya que representó el primer intento realizado para automatizar esta tarea. Actualmente –y debido a la desactualización de dichas plantillas- este sistema está en desuso. Como consecuencia, las ofertas comienzan a crearse copiando y pegando partes de otros documentos con contenido similar.

Ante esta situación se hace necesario una mejora en el sistema de generación de ofertas de tal forma que este proceso sea más eficiente y –lo más importante- se eviten errores. Además, el sistema propuesto debe ser fácil de utilizar por las distintas partes implicadas en la confección de las propuestas de venta.

3.3 Objetivos

El objetivo planteado por los departamentos comerciales de las distintas empresas que forman parte del grupo TELSTAR SA son:

- **Generación eficiente de ofertas:** Implica la creación de documentos en un corto período de tiempo.

- **Autogestión:** Comporta el hecho de que el propio departamento comercial pueda ser capaz de gestionar de forma autónoma sus plantillas de ofertas: imagen corporativa, contenido y su distribución en el documento de forma fácil y sencilla.
- **Presentación:** El resultado final deberá ser un documento Word.
- **Gestión de la información**: deberá ser posible recuperar la información volcada en un documento.

Todos estos objetivos están planteados con una única intención: aumentar el volumen de ofertas entregadas, con el fin de incrementar las ventas.

4 DESCRIPCIÓN

En este capítulo se hace una revisión a las características del antiguo sistema de generación de ofertas, y los principales requerimientos del nuevo sistema. A continuación se comenta la planificación seguida para el desarrollo del proyecto así como la solución propuesta surgida a partir de los requerimientos.

4.1 Estado inicial

Como ya se ha mencionado, las ofertas son creadas utilizando plantillas Word que incluyen macros escritas en Visual Basic. Dado el carácter dinámico de las máquinas que se comercializan, dichas plantillas deben ser actualizadas a menudo para reflejar nuevas especificaciones técnicas. La tarea de actualización es llevada a cabo por una persona perteneciente al departamento comercial que debe poseer unos conocimientos mínimos sobre programación. Debido a la alta rotación de personal en dichos departamentos, el conocimiento de cómo gestionar los cambios en los documentos se ha ido "diluyendo" a lo largo del tiempo.

Todo esto lleva a que las plantillas Word utilizadas como base para crear ofertas estén completamente desactualizadas, lo que obliga a construir ofertas a partir de trozos de documentos antiguos enviados a otros clientes. Si a esta situación se une la celeridad exigida en los plazos de entrega de las ofertas, el resultado es un documento final descuidado, en donde la probabilidad de cometer errores es muy alta. Esto implica un peligro muy grande, si se tiene en cuenta el carácter contractual de un escrito de este tipo.

Al existir distintos perfiles comerciales –unas personas con más nivel técnico y conocimiento del producto; otras con un perfil más júnior – gran parte del tiempo de los comerciales más experimentados está dedicado a la revisión de las ofertas creadas por los menos expertos, lo que implica un coste elevado para la empresa.

Una oferta tipo está compuesta por unas 30 a 80 páginas dependiendo del producto ofertado y de los opcionales solicitados. En ella se incluye toda la información técnica y descriptiva relativa a los distintos componentes que forman parte de una máquina. La misma puede incluir imágenes y diagramas, principalmente para añadir más información y darle un mejor aspecto final.

Atendiendo al gran volumen de información asociada y a la complejidad de las máquinas ofertadas, el tiempo medio empleado para la creación de una propuesta de venta -y su posterior revisión- es elevado.

Las ofertas pueden ser generadas en distintos formatos e idiomas. A día de hoy, **por cada formato/idioma existe un documento con sus correspondientes macros**. Esto implica que el cambio en una de las plantillas deberá ser replicado manualmente en el resto: como se ve, esto es algo impráctico.

Una de las necesidades del departamento comercial es la de autogestionar la información contenida en los documentos debido a la variabilidad de las especificaciones técnicas de las máquinas comercializadas, por lo que no es realizable que otro departamento (por ejemplo el departamento de informática) intervenga en el proceso de gestión de cambios. Si así fuese, esto generaría un trabajo extra al departamento implicado. Esto crea la obligación de implementar una solución "*user-friendly*".

4.2 Requerimientos

Como resultado de una entrevista realizada con los distintos miembros del departamento comercial, se han extraído una serie de requerimientos de los cuales se presentan los más relevantes:

1. El formato del documento generado deberá ser en formato DOC de Microsoft Word.
2. Gestión por parte del departamento comercial del contenido de las ofertas.
3. Las ofertas podrán generarse en distintos idiomas, sin que ello signifique multiplicar esfuerzos de forma innecesaria.
4. La información a incluir en el documento podrá ser alojada en cualquier parte (encabezado, pie de página, tablas, etc.) y con cualquier formato/color de texto.
5. El contenido de la oferta será "dinámico" en función de la selección de elementos opcionales por parte del usuario, existiendo en el documento diferentes apartados que

hacen referencia a ellos. Por apartado se entiende cualquier elemento que permita ser insertado en un documento Word: imagen, párrafo, viñetas, dibujo, etc.

6. Cada elemento opcional tiene asociado en el documento más de una sección: si el mismo no es incluido, todas ellas deberán eliminarse sin dejar "huecos" o espacios en blanco.
7. Deberá existir una forma sencilla de:
   - Definir el contenido de los formularios que deberán completar los usuarios.
   - Gestionar su cambio.
8. La modificación de un formulario no deberá implicar la imposibilidad de recuperar la información de formularios creados con anterioridad.
9. La generación de ofertas podrá hacerse tanto online como offline.
10. La información introducida por el usuario deberá ser validada, no permitiendo la generación de ofertas si la información contenida no cumple alguna regla.

En el anexo A se encuentran las fichas de los principales requerimientos solicitados, así como otros requerimientos surgidos en reuniones posteriores.

4.3 Planificación

La planificación inicial para llevar a cabo este proyecto incluye –a grandes rasgos- las siguientes fases:

- Visión
  - Análisis del problema
  - Definición de los requerimientos
  - Estudio de viabilidad
- Desarrollo
  - Diseño
  - Codificación
- Implantación

La aplicación de las fases no se ha realizado en forma lineal, es decir, en bloques de arriba hacia abajo; sino que se ha seguido una metodología en espiral: al momento de tener una parte de código operativa era posible la refinación de algunos requerimientos o incluso la aparición de otros nuevos.

Una vez obtenidos los requerimientos se observó la necesidad de integrar información de forma eficiente en Microsoft Word. Este hecho motivó una investigación sobre las posibles vías de conseguirlo, y representó una fase previa al planteamiento de cualquier solución, ya que la misma vendría condicionada por esta circunstancia.

También se hizo evidente la necesidad de desarrollar distintos componentes de naturaleza heterogénea, lo que obligó a tener en cuenta una "sub-planificación" para ellos dentro de la planificación global del proyecto.

4.4 Solución planteada

La solución al problema planteado se libra en varios frentes y se basa principalmente en la integración de diversos estándares y la creación de una herramienta de generación de formularios. La principal idea es separar:

- **Presentación**: Documento entregado al cliente.
- **Datos**: Información volcada en el documento, recolectada a través de un formulario.

De esta forma se logra tener una única fuente de recolección de datos, y múltiples sitios donde podrán ser volcados.

De forma resumida el proceso propuesto para la generación de una oferta implica dos fases:

- **Diseño**: Creación del formulario y de una plantilla Word por parte de un usuario "diseñador".
- **Ejecución**: Utilización del formulario e integración de datos con plantilla Word definida en la primera fase por parte de usuarios convencionales.

A su vez, el proceso de diseño implica:

- Definición de la información contenida en el formulario, los datos que deberá introducir el usuario y la relación ente ellos: restricciones de visibilidad, de valores, de navegabilidad, etc. Como resultado final se obtendrá un formulario.
- Definición de "marcas" en un documento Word que cumplirá la función de plantilla. Las marcas indicarán la posición donde existe información que debe ser reemplazada y/o mantenida.

El proceso de definición del formulario será llevado a cabo en forma gráfica a través de la aplicación Smart Proposal.

Resumiendo de forma gráfica los pasos a seguir para la construcción de un formulario y generación de una oferta son los siguientes:

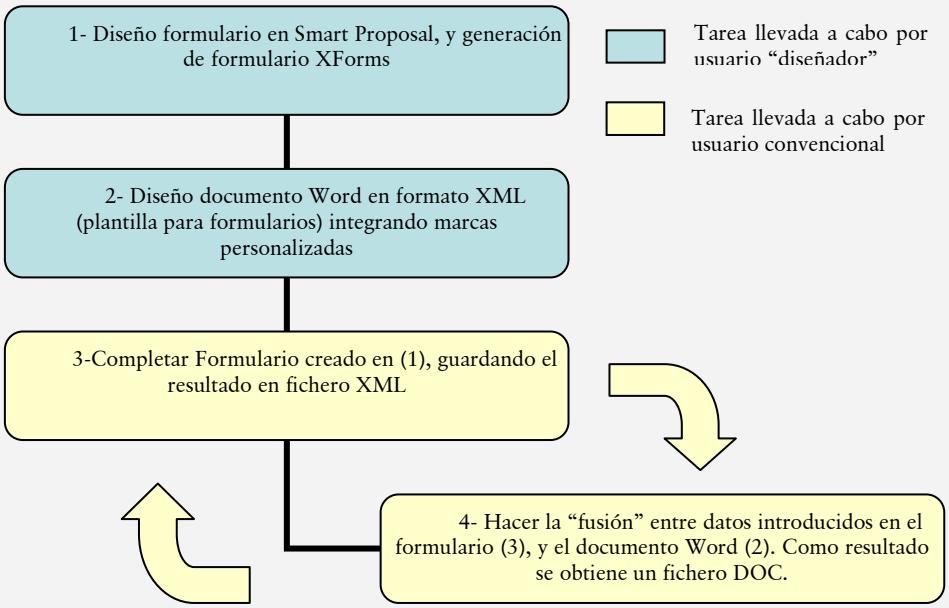


Figura 4-1 Proceso generación formulario y oferta

Las etapas 1 y 2 del diagrama de la Figura 4-1 deberán realizarse **una única vez**, para la definición y construcción del formulario por un usuario con un perfil de "diseñador" que debe tener conocimientos sobre las aplicaciones Smart Proposal y Smart Document.

Las etapas 3 y 4 se llevarán a cabo por usuarios convencionales (normalmente comerciales) cada vez que se quiera realizar una oferta:

- Se abrirá el formulario creado en el punto 1, y se guardará el fichero creado con el mismo conteniendo la información de la oferta.
- Se seleccionará un documento Word donde presentar la información recolectada, (plantilla creada en el punto 2) y se obtendrá un fichero DOC con la oferta resultante.

5 DISEÑO

En el diseño general de la solución propuesta intervienen tres módulos principales llamados Smart Proposal, Smart Document y Proposal Generator. Otro módulo del proceso es el formulario generado por la aplicación Smart Proposal, que aunque no disponga de un nombre particular, es parte fundamental del procedimiento de generación de ofertas y merece una mención particular. Cada una de estas aplicaciones están construidas sobre diferentes lenguajes y tecnologías, que serán citadas y –algunas de ellas- ampliadas a lo largo del capítulo.

5.1 Módulos

A continuación se describen de forma global las distintas aplicaciones/módulos utilizados para resolver el problema planteado. En secciones posteriores se profundizará en su funcionalidad e implementación.

5.1.1 Smart Proposal

Aplicación que permite definir de forma fácil y sencilla el contenido y estructura de los formularios con los que el usuario interactuará. Smart Proposal define dos tipos de elementos principales:

- **Entidades**: Agrupan un conjunto de datos relacionados entre si. Estos datos reciben el nombre de atributos.
- **Conectores**: Vinculan de forma condicional entidades.

5.1.2 Formulario

Representa el medio para recolectar información de la oferta. Es creado a través de un proceso de generación implementado en Smart Proposal para tal efecto. Este formulario reflejará la estructura de datos (entidades y atributos) y su relación (conectores) definidos por medio de dicha aplicación.

5.1.3 Proposal Generator

Proposal Generator es un control ActiveX que permite generar un fichero Microsoft Word a partir de un documento convenientemente formateado que funcione a modo de plantilla (creado con la extensión Smart Document) y un fichero creado a partir del formulario producido con Smart Proposal.

5.1.4 Smart Document

Smart Document es un complemento para Microsoft Word que permite introducir marcas en los documentos que servirán de plantilla para la generación de ofertas. Dichas marcas están relacionadas con las entidades y atributos definidos en la aplicación Smart Proposal.

Las marcas que podrán introducirse son de dos tipos:

- **Markers**: Corresponden a zonas del documento asociadas a entidades. Pueden incluir imágenes, tablas, etc.
- **ReplaceText**: Corresponden a pequeñas porciones de texto (generalmente palabras) que serán reemplazadas por valores de atributos.

5.2 Tecnologías utilizadas

Las tecnologías utilizadas son muy variadas, debido al carácter interdisciplinar que se ha adoptado en la solución del problema. A continuación se detallan para cada módulo los lenguajes/estándares utilizados, y en secciones posteriores se realiza una pequeña descripción de cada una de ellos.

Smart Proposal

- Lenguaje de programación C# para el desarrollo de la aplicación
- Framework ANTLR

Formulario

- XForms

- CSS
- JavaScript

Smart Document

- XML
- XSD

Proposal Generator

- Lenguaje de programación C# para el desarrollo del control ActiveX
- XML
- XSD
- XSLT

El objetivo es aprovechar las ventajas que la combinación de todos estos estándares ofrecen, realizando desarrollos allí donde algunos de ellos no llegue a cubrir todas las necesidades. Quizás el mayor desconocido de todos ellos sea –a día de hoy- el estándar XForms. Debido a ello -y a su uso casi en exclusiva para la construcción del formulario- se merece una mención especial.

5.2.1 ANTLR

ANTLR (ANother Tool for Language Recognition) es un analizador sintáctico y un generador que permite construir compiladores, traductores e intérpretes para distintos lenguajes específicos como pueden ser formatos de datos propietarios, protocolos de comunicación, lenguajes de procesamiento, etc. La definición de las reglas de reconocimiento se realiza con un formato similar al EBNF (Extended Backus-Naur Form).

ANTLR es más que un lenguaje para la definición de gramáticas, ya que las herramientas que proporciona permite implementar analizadores léxicos (lexers) y analizadores sintácticos (parsers) en gran variedad de lenguajes: Java, C++, C#, etc.

Con ANTLR es posible definir las reglas que el lexer debe utilizar para interpretar un conjunto de caracteres como tokens y las reglas que el parser debe utilizar para interpretar un conjunto de tokens. Por lo tanto ANTLR es sumamente útil a la hora de generar lexers y parsers que puedan interpretar código fuente de programas escritos en un lenguaje propio y traducirlos a otros lenguajes.

Por ejemplo, el siguiente parser acepta expresiones aritméticas entre enteros con operadores de suma, resta y multiplicación como 2+6*3 o expresiones con paréntesis como (2+6)*3 para forzar el orden de evaluación.

```
expr:   expr2 ((PLUS|MINUS) expr2)*    ;
expr2 :   expr3  (STAR expr3)*    ;
expr3:   INT     |   LPAREN expr RPAREN     ;
LPAREN: '(' ;
RPAREN: ')' ;
PLUS  : '+' ;
MINUS : '-' ;
STAR  : '*' ;
INT   : ('0'..'9')+ ;
WS    : ( ' '  | '\r' '\n'  | '\n'  | '\t'   ) ;
```

En el proyecto presentado se ha utilizado ANTLR para la generación de un analizador de expresiones donde aparecen operaciones aritméticas tradicionales, donde intervienen constantes e identificadores. Además se da soporte a los tipos de datos entero, flotante y string.

5.2.2 Hojas de estilo CSS

Las hojas de estilo en cascada (CSS o Cascade Style Sheets) suponen un gran avance en la creación de sitios web. Mejoran las posibilidades de diseño y presentación de documentos, facilitando su mantenimiento, ya se trate de un único archivo o de grandes sitios con gran cantidad de páginas.

La filosofía de las CSS responde a la idea de separar forma y presentación. Las páginas tendrán únicamente etiquetas con información acerca de la estructura y contenido del documento (párrafos, cabeceras, listas, etc.) sin ninguna información sobre la apariencia del documento. Esta apariencia, la forma en que ese documento debe ser visualizado, se dicta por una serie de reglas que se definen de forma separada, de tal manera que se pueda cambiar la manera en que el documento es visualizado sin tocar ni una sola línea de contenido ni cambiar las etiquetas, simplemente introduciendo unos pocos cambios en la definición de estilo.

En el formulario XForms generado la mayor parte de la información relacionada con la presentación (estilo y distribución del texto y controles) es almacenada en una hoja de estilos, teniendo en cuenta las particularidades que obliga a utilizar este estándar. Todo ello se verá con más detalle en secciones posteriores.

5.2.3 XML

XML (eXtensible Markup Language o Lenguaje extensible de marcas) es un conjunto de reglas utilizadas para definir etiquetas semánticas con el fin de organizar un documento. Además XML es un metalenguaje que permite diseñar lenguajes de etiquetas propios, a diferencia de otros lenguajes de etiquetas tradicionales como HTML. XML describe la información en un formato independiente de la plataforma que permite su distribución.

XML como lenguaje es la combinación de una serie de entidades (elementos, atributos, referencias de entidad, comentarios e instrucciones de procesamiento) siguiendo las directrices de un conjunto de reglas sintácticas. Las reglas sintácticas más importantes son las siguientes:

- Todos los documentos XML deben tener un elemento raíz.
- Todos los elementos XML deben tener una etiqueta de cierre.
- Los nombres para las etiquetas distinguen entre mayúsculas y minúsculas.
- Todos los elementos XML deben estar anidados correctamente.
- Los atributos deben estar incluidos en la etiqueta de apertura y sus valores deben ser escritos entre comillas.

En el proyecto se hace un uso intensivo de las tecnologías XML, ya que la información es recogida en un fichero XML, a través de un formulario escrito en XML, e integrada en un documento Word en formato XML. Sobre estos temas se hablará con mayor detalle en secciones posteriores.

5.2.4 XSLT

Cuando se habla de XSLT, intervienen tres especificaciones: XSL (también llamada XSL-FO), XSLT y XPath. Originariamente estos tres lenguajes fueron parte de uno solo, XSL o "Extensible Stylesheet Language", pero finalmente fueron separados. XPath es utilizado por XSLT, que a su vez es usado por XSL (Figura 5-1)

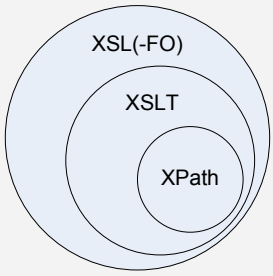


Figura 5-1 Tecnologías dependientes XSLT

XSL es un lenguaje utilizado para escribir hojas de estilo para documentos XML. Está integrado por dos módulos:

- Un vocabulario de formateo XML
- Un lenguaje de transformación XML

XSLT es el componente de transformación de XSL. El otro componente –el vocabulario de formateo- es normalmente llamado XSL Formatting Objects, o XSL-FO. En teoría puede funcionar de forma independiente como un vocabulario de formateo, pero normalmente es

utilizado como parte de XSL. El uso más común para XSL-FO es transformar documentos XML en documentos con un formato apropiado para la impresión, como PDF.

XSLT (Extensible Stylesheet Language Transformations) es un lenguaje para transformar documentos XML en otros documentos XML o hacia otros formatos como HTML o texto puro. Una hoja de estilos XSLT es un programa que define de forma declarativa la transformación desde un "árbol origen" (entrada) hacia un "árbol resultado" (salida). Las hojas de estilo XSLT están representadas en formato XML, por lo tanto, significa que hay tres documentos XML básicos o "árboles" involucrados en cualquier transformación XSLT tal como puede verse en la Figura 5-2.

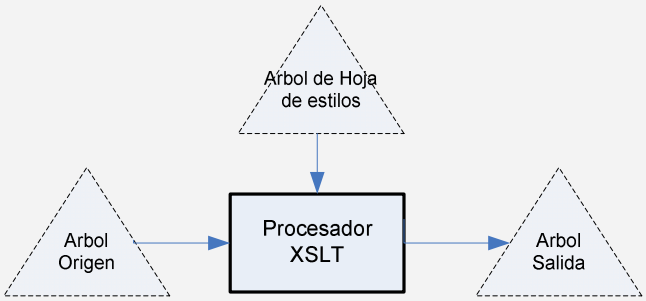


Figura 5-2 Procesador XSLT

Las entradas del procesador XSLT son el documento XML origen (el árbol entrada) y el programa XSLT (el árbol de hoja de estilos). La salida de la transformación es un nuevo documento XML (el árbol resultante)

En este proyecto, la integración de datos en el documento Word se realiza a través de una transformación XSLT a partir de los datos introducidos por el usuario en el formulario XForms.

5.2.5 XPath

XPath es la abreviatura de "XML Path Language". Es un lenguaje de expresiones para hacer referencia a partes de documentos XML. XPath es parte esencial de XSLT y es usado para seleccionar "nodos" del árbol de entrada que luego serán procesados.

El modelo de datos XPath es fundamental para XSLT. Matemáticamente hablando, define el dominio o rango de las funciones XSLT. En otras palabras, define los nodos sobre los que se aplicará una transformación. XPath define que es un árbol, los siete tipos de nodos que pueden aparecer en un árbol (nodo raíz, elemento, atributo, comentario, instrucción de procesamiento, espacio de nombre y de texto) y como están relacionados.

A continuación se detallan algunos ejemplos de expresiones XPath:

```
/article/heading
```

Selecciona el elemento `heading` hijo del elemento raíz `article`.

```
/article/para[position( )=1]
```

Selecciona el primer elemento `para` hijo del elemento raíz `article`. Expresión equivalente a `/article/para[1]`

```
/article/para[position( )=last( )]
```

Selecciona el último elemento `para` hijo del elemento raíz `article`. Expresión equivalente a `/article/para[last( )]`

```
self::node( )
```

Selecciona el nodo de contexto (nodo actual). Equivalente a la expresión `"."`.

```
./order
```

Selecciona el nodo `order` hijo del nodo actual. Expresión equivalente a `order`.

```
order/attribute::price
```

Selecciona el atributo `price` de cada nodo `order` que sea hijo del nodo actual. Expresión equivalente a `order/@price`.

```
order[@price > 30]/shipTo
```

Selecciona el elemento `shipTo` hijo de cada nodo `order` (que a su vez sea hijo del nodo actual) cuyo atributo `price` tenga un valor mayor a `30`.

Estos ejemplos sólo ilustran algunas operaciones y funciones soportadas por XPath. Además de las presentadas, pueden utilizarse expresiones donde intervengan operaciones aritméticas (`+`, `-`, `*`, `div`, `mod`) o lógicas (`and`, `or`).

Las expresiones XPath aparecen como valores de varios atributos en XSLT. Por ejemplo, el atributo `select` que forma parte de muchas instrucciones XSLT contiene una expresión XPath:

```
<xsl:variable name="bName" select="./@attName" />
```

5.2.6 XSD

XSD (XML Schema Description) es un vocabulario XML que permite describir otros vocabularios XML de tal manera que los programas puedan comprobar cuando un documento dado cumple con las reglas establecidas en el esquema.

A continuación se presenta un pequeño ejemplo donde se muestra un documento XML y su correspondiente esquema donde se describe su estructura.

```
<?xml version="1.0" encoding="us-ascii"?>
<authors>
    <person id="lear">
        <name>Edward Lear</name>
        <nationality>British</nationality>
    </person>
    <person id="asimov">
        <name>Isaac Asimov</name>
        <nationality>American</nationality>
    </person>
    <person id="whois"/>
</authors>
```

Este documento contiene un elemento `authors`, que a su vez contiene múltiples elementos `person`. Cada elemento `person` contiene un atributo `id` y puede contener de forma opcional un elemento `name` y un elemento `nationality`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
     <xs:element name="authors">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="person"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence minOccurs="0">
        <xs:element ref="name"/>
        <xs:element ref="nationality"/>
```

```
      </xs:sequence>
      <xs:attribute ref="id" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="name" type="xs:string"/>

  <xs:element name="nationality" type="xs:string"/>

  <xs:attribute name="id" type="xs:string"/>
</xs:schema>
```

Este esquema comienza definiendo el elemento `authors`, que es la raíz del documento, y su contenido. Ya que el elemento `authors` contiene más elementos que no son solo texto, es definido como un tipo `xs:complexType`. Este tipo contiene una secuencia de elementos `person`. La definición del tipo `person` contiene a su vez un `xs:complexType` que contiene un `xs:sequence`, donde se especifican los elementos `name` y `nationallity` (cada uno de ellos contiene una cadena). El `xs:complexType` para el elemento `person` contiene además la definición para el atributo `id`.

Los atributos `maxOccurs`, `minOccurs` y `use` permiten definir –respectivamente- el número de repeticiones máximas, mínimas y si el atributo o elemento es obligatorio.

5.2.7 XForms

La World Wide Web Consortium (W3C) desarrolló el estándar XForms para la recolección y presentación de datos en formularios. Como indica el documento de recomendación de W3C, XForms intenta ser *"la próxima generación de formularios para la Web"*. XForms incluye un gran número de ventajas respecto a la tecnología de formularios HTML existentes. Tal como la recomendación indica, *"separando los formularios tradicionales en tres partes –modelo XForms, datos de instancia, e interfaz de usuario- se reducen el número de consultas al servidor, teniendo además (…) escasa necesidad de programar scripts."*

Estas ventajas se detallan a continuación:

- XForms mejora la experiencia de usuario, ya que ha sido diseñado para permitir validación de datos en lado cliente, como por ejemplo, comprobación de los tipos de campos que están siendo rellenados y validación de los campos obligatorios. Esto reduce la necesidad de enviar y recibir continuamente datos a y desde el servidor, y de la utilización de soluciones basadas en scripts. De esta forma, el usuario recibe una retroalimentación inmediata sobre los campos que está completando.
- Es XML, y puede enviar XML. XForms está integrado con XML: está basado en XML, los datos recogidos en los formularios son XML, puede cargar documentos XML externos como datos iniciales, y puede enviar los resultados en formato XML. Por lo tanto, es posible su integración con servicios Web y otras aplicaciones que consuman datos en este formato.
- Combina tecnologías XML existentes. XForms utiliza varias tecnologías XML existentes, tales como XPath para direccionamiento y cálculo de valores, y XML Schema (XSD) para la definición de tipos de datos.
- Es independiente de los dispositivos. El mismo formulario puede ser enviado sin cambios a un navegador tradicional, una PDA, un teléfono móvil...
- Es más fácil crear formularios complejos y flexibles sin tener que recurrir a la programación de scripts, ya que XForms utiliza etiquetas para definir propiedades de valores y para construir relaciones entre éstos.

Actualmente existen dos versiones del estándar:

- **XForms 1.0**: Primera versión publicada en el año 2003. Posteriormente, y debido a la limitación/carencia de determinadas funcionalidades detectadas por los diseñadores de formularios, se realizó una revisión que recibió el nombre de XForms 1.0 Second Edition. Actualmente se encuentra en estado borrador la tercera revisión de XForms 1.0 (Third Edition)

- **XForms 1.1**: A día de hoy XForms 1.1 tiene el estado de recomendación. Propuesto en 2004, XForms 1.1 refina el procesamiento XML introducido por la versión 1.0 añadiendo nuevas características: manejadores de eventos, funciones, mejoras en la interfaz de usuario, nuevos tipos de datos, posibilidad de acceder a información de contexto de los eventos, etc.

XForms divide los formularios tradicionales en tres partes

- Modelo XForms
- Interfaz de usuario
- Datos de instancia

Para comenzar, se define el llamado modelo del formulario que contiene una o varias instancias de datos. La instancia de datos está representada por una estructura XML con la información que se desea recoger/enviar, además del modo de hacerlo: URL del servidor, método utilizado (GET/POST), entre otros.

Además deben definirse enlaces (binds) para los distintos elementos de la instancia donde se indican ciertas propiedades. Por ejemplo, si es un dato de tipo obligatorio, relevante, de solo lectura, calculado a partir de otros valores, así como las restricciones que deben existir en cuanto a su contenido. Puede especificarse además el tipo de información que contiene: número entero, flotante, cadena, información binaria, etc.

Luego se definen los controles del formulario. Existen diferentes clases, incluyendo por supuesto los controles más comunes: botones, cuadros y áreas de texto, listas desplegables, botones de radio, cuadros de selección, controles de tipo calendario y rango. Cada uno de ellos está asociado a uno de los enlaces definidos en la primera sección. De esta forma el contenido de un valor enlazado es representado por un determinado control en el formulario.

XForms permite aumentar las funcionalidades e integración de datos a través del uso de scripts escritos en JavaScript. Por ejemplo, es posible obtener los datos del nodo `openFilePath` de la instancia `sp-internalVars` que pertenece al modelo `sp-main_model` de la siguiente forma:

```
var vModel = document.getElementById("sp-main_model");
var vInstance = theModel.getInstanceDocument("sp-internalVars");
var vNode = theInstance.selectSingleNode("//openFilePath");
alert("Valor para openFilePath:" + theNode.text);
```

El siguiente ejemplo ayudará a clarificar la mayoría de conceptos:

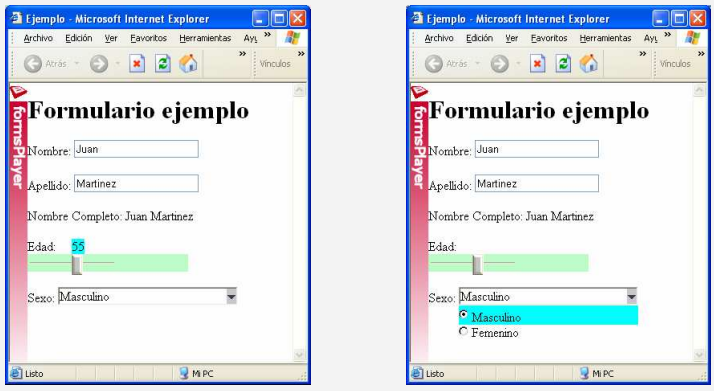


Figura 5-3 Formulario ejemplo en IE6 utilizando formsPlayer

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:xf="http://www.w3.org/2002/xforms">
  <head>
    <object width="0" height="0" id="formsPlayer"
            classid="CLSID:4D0ABA11-C5F0-4478-991A-375C4B648F58"
            codebase="http://skimstone.x-
```

```
port.net/files/releases/formsPlayer1.4.3.1028.cab#Version=1,4,3,1028">
      <b>formsPlayer no se encuentra instalado.</b>
    </object>
    <?import namespace="xf" implementation="#formsPlayer"?>

    <title>Ejemplo</title>
    <xf:model id="sp-main_model" xmlns:xf="http://www.w3.org/2002/xforms">
      <xf:instance id="sp-data" xmlns="">
        <root   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <edad xsi:type="xsd:decimal">20</edad>
          <nombre>Juan</nombre>
          <apellido>Martinez</apellido>
                  <nombrecompleto></nombrecompleto>
          <sexo>Masculino</sexo>
        </root>
      </xf:instance>

      <xf:bind id="sp-bind-edad" nodeset="instance('sp-data')/edad" />
      <xf:bind id="sp-bind-nombre" nodeset="instance('sp-data')/nombre" />
      <xf:bind id="sp-bind-apellido" nodeset="instance('sp-data')/apellido" />
      <xf:bind id="sp-bind-sexo" nodeset="instance('sp-data')/sexo" />
      <xf:bind  id="sp-bind-nombrecompleto" nodeset="instance('sp-data')/nombrecompleto"
                readonly="true" calculate="concat(instance('sp-data')/nombre, ' ' ,
instance('sp-data')/apellido)"/>
    </xf:model>
  </head>

  <body>
        <h1>Formulario ejemplo</h1>

        <xf:input bind="sp-bind-nombre">
                <xf:label>Nombre: </xf:label>
        </xf:input>

        <br/><br/>

        <xf:input bind="sp-bind-apellido">
                <xf:label>Apellido: </xf:label>
        </xf:input>

        <br/><br/>

        <xf:output bind="sp-bind-nombrecompleto">
                <xf:label>Nombre Completo: </xf:label>
        </xf:output>

        <br/><br/>

        <xf:range bind="sp-bind-edad" start="1" end="99" step="1" >
            <xf:label>Edad: </xf:label>
        </xf:range>

        <br/><br/>

        <xf:select1 bind="sp-bind-sexo" appearance="minimal">
                <xf:label>Sexo: </xf:label>
                <xf:item>
                        <xf:label>Masculino</xf:label>
                        <xf:value>Masculino</xf:value>
                </xf:item>
                <xf:item>
```

```
                        <xf:label>Femenino</xf:label>
                        <xf:value>Femenino</xf:value>
                </xf:item>
        </xf:select1>

  </body>
</html>
```

En este ejemplo puede observarse la definición de la instancia de datos (`sp-data`) dentro de un modelo (`sp-main_model`) donde se enlazan sus distintas partes utilizando elementos `bind`. Mediante atributos es posible asignar propiedades sobre su valor. Por ejemplo, el elemento `nombrecompleto` de la instancia `sp-data` es de solo lectura y calculado a partir de la concatenación de otros dos elementos.

En el cuerpo del formulario se encuentra la definición de los controles, los cuales están asociados a los datos de la instancia por medio de los elementos `bind`. Las propiedades definidas en estos elementos afectarán la forma en que los controles son representados en el formulario. Por ejemplo, un valor de instancia que no sea relevante hará que su control asociado permanezca oculto.

En este pequeño ejemplo puede observarse el uso de cuatro controles diferentes: controles de entrada y salida de texto (`input` y `ouput` respectivamente), selección (`select1`) y de rango (`range`). Todos ellos contienen un elemento `label` que define la etiqueta asociada al control. El resultado de su ejecución puede verse en la Figura 5-3.

A continuación se realiza un repaso por las distintas implementaciones del estándar que existen en la actualidad, presentándose soluciones para entornos de servidor y de cliente.

5.2.7.1 XForms cliente

A día de hoy ninguno de los navegadores más populares (Internet Explorer y Mozilla Firefox) ofrecen soporte de forma nativa para XForms. En su lugar, deben instalarse extensiones o controles ActiveX en los clientes (o exploradores web) para suplir esta carencia.

Para Internet Explorer existe un plug-in desarrollado por fomsPlayer (1) que implementa muchas de las características del borrador del estándar en su versión 1.1. Se ofrecen bajo dos tipos de licencia: comunity y professional. La primera de ellas es gratuita y no da derecho a soporte técnico, además de estar limitado su uso a un determinado periodo de tiempo. Pasado dicho periodo, en los formularios se hace visible un logo de la empresa creadora junto cada control que rompe el diseño de la página. Desde formsPlayer justifican este comportamiento argumentando que de esta forma se aseguran que los usuarios utilizan siempre la versión más reciente de su software.

La versión profesional es igual que la versión gratuita, pero sin restricción en cuanto a tiempo de uso, con lo que no se está obligado a actualizar el software de forma constante.

Para Mozilla Firefox existe una extensión gratuita (2) desarrollada por un grupo de trabajo de IBM (empresa que brinda pleno apoyo a este estándar) que a día de hoy solo soporta de forma parcial la versión 1.0 Second Edition de XForms, y algunas pocas características de la versión 1.1.

Existe una tercera solución - independiente del navegador- consistente en una implementación 100% JavaScript del estándar en su versión 1.0. FormFaces es un conjunto de librerías JavaScript que utiliza técnicas AJAX para convertir/transformar un formulario XForms en un formulario HTML puro de forma transparente. De esta forma no se necesita instalar ningún plugin en el cliente.

5.2.7.2 XForms servidor

En el lado servidor la alternativa más completa –y que dispone de una versión gratuita- es la que brinda la empresa Orbeon a través de su producto Orbeon Forms. Este producto debe instalarse sobre un servidor de aplicaciones que se ejecute sobre Java e implemente el API Servlet 2.3 (como por ejemplo Tomcat o JBoss) En su sitio web (3) pueden verse demos de formularios que demuestra lo potente que puede llegar a ser esta tecnología (Figura 5-4)

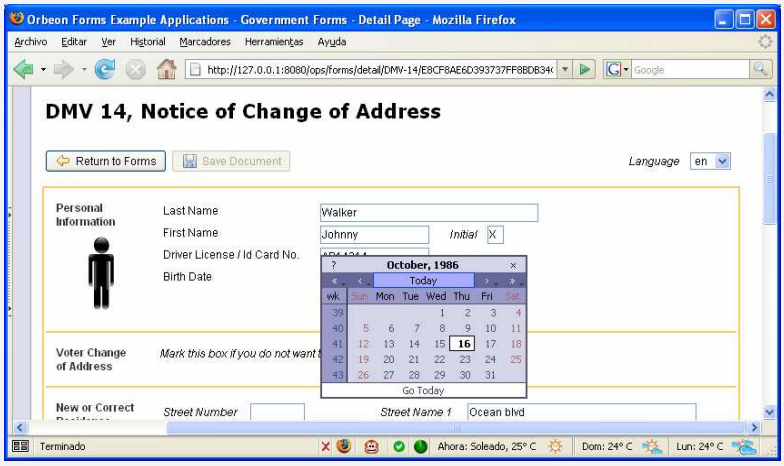


Figura 5-4 Formulario XForms ejemplo utilizando Orbeon

El problema fundamental en todas las implementaciones (cliente o servidor) es que -de una forma u otra y dependiendo de la complejidad del formulario- incluyen elementos no estándares que dificultan la migración entre ellas. Además, no es raro observar distintos comportamientos sobre un mismo formulario según la solución escogida.

Al inicio del proyecto se utilizó la implementación XForms de Mozilla, pero al observar las carencias y limitaciones que presentaba, se decidió pasar a la implementación de formsPlayer (la solución más completa en el lado cliente)

Dado que uno de los requisitos del proyecto es la posibilidad de generar ofertas en modo offline, la solución ofrecida por Orbeon fue descartada. Sin embargo, se planteó la posibilidad de instalar un servidor de aplicaciones en cada cliente si las ventajas ofrecidas lo justificaban.

6 SMART PROPOSAL

Smart Proposal es la aplicación desarrollada en C# cuya finalidad es la de diseñar y generar formularios XForms. Su interfaz (Figura 6-1) está dividida en tres secciones principales:

- **Librería de objetos**: En ella están representados los distintos objetos que forman parte del documento.
- **Área de dibujo**: En la parte central se encuentra el área de edición del documento, donde se podrán añadir, eliminar, mover y editar objetos.
- **Propiedades**: Se encuentra situada en la parte derecha y permite editar las propiedades del objeto que se encuentre seleccionado en el área de dibujo.

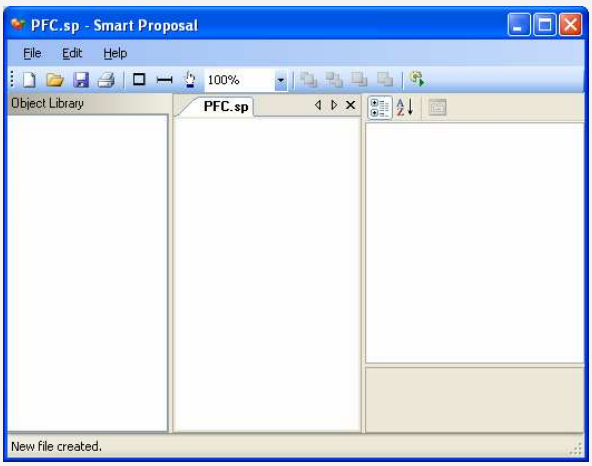


Figura 6-1 Smart Proposal

A partir de ahora –y para evitar confusiones- llamaremos "diseñador" al usuario de esta aplicación y simplemente "usuario" a la persona que utiliza los formularios generados por ella (habitualmente un comercial)

La aplicación Smart Proposal está pensada para ser utilizada por personas sin conocimientos de programación. Su objetivo es la de definir una estructura y un flujo de datos que será plasmado en un formulario que podrá ser abierto desde un navegador web.

¿Pero a que se hace referencia con "flujo de datos"? Se hace alusión al hecho de poder ocultar y/o hacer visibles determinadas partes del formulario en función de la entrada del usuario, según una serie de condiciones predefinidas.

Toda la funcionalidad de la aplicación está basada en dos tipos de objetos: Entidades y conectores.

6.1 Entidades

Tal como puede verse en la Figura 6-2, cada entidad tiene asociada un conjunto de propiedades:

- **Nombre**: Nombre de la entidad. El nombre de cada entidad/conector debe ser único.
- **Fuente y color de fuente**: Define el tipo de fuente y el color que se utiliza para presentar el nombre de la entidad y sus atributos en pantalla.
- **Color inicial y color final**: Especifica los colores del degradado que se utilizan para representar la entidad en el área de dibujo.
- **Ancho de línea**: Fija el valor de ancho de línea que se utiliza para representar la entidad.
- **Opcional**: Determina si el usuario puede eliminar la entidad en el formulario generado.

Una entidad incluye un conjunto de atributos que representan los campos que forman parte del formulario. La misión de la entidad es agrupar datos relacionados entre si por alguna propiedad de acuerdo a algún criterio definido por del diseñador.

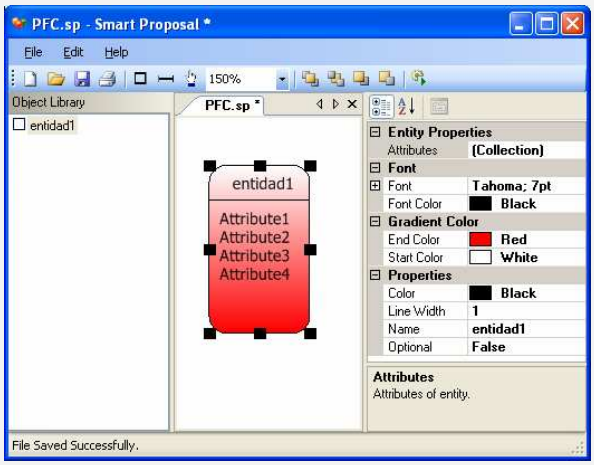


Figura 6-2 Entidad con cuatro atributos

6.1.1 Atributos

Los atributos tienen asociados unas propiedades que afectan el aspecto y la forma en la que se comportarán en el formulario una vez generado.

- **Clase**: Indica si el atributo contiene un valor simple (cuadro de texto) o múltiple valores (lista desplegable)
- **Tipo**: Define los distintos tipos de valores aceptado por el campo: Número entero, número flotante o cadena.
- **Etiqueta**: Texto que aparece junto al cuadro de texto o lista desplegable que representa al atributo.
- **Validación**: Conjunto de reglas que deben cumplirse para considerar el valor del atributo correcto.
- **Mensaje de error**: Texto informativo que aparece en caso de introducir un valor incorrecto o que no cumple alguna regla de validación.
- **Sin valor**: Indica si el atributo debe contener algún valor para ser válido.
- **Solo lectura**: Define si el valor del atributo podrá ser modificado por el usuario.
- **Visible**: Determina si el atributo es visible en el formulario.
- **Valores**: Permite definir valores (por defecto o calculados) para el atributo.

A través del editor de atributos (Figura 6-3) es posible modificar todas las propiedades anteriormente descritas.

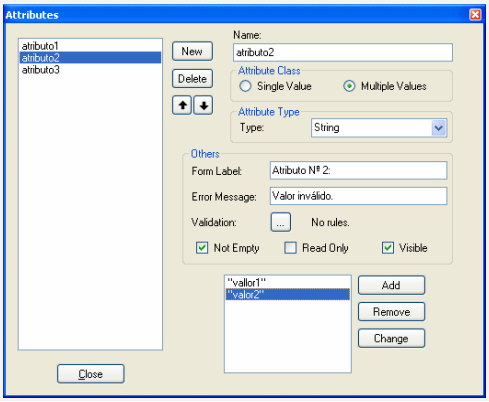


Figura 6-3 Atributos de una entidad

Las reglas de validación para los atributos son definidos a través del editor de reglas (Figura 6-4)

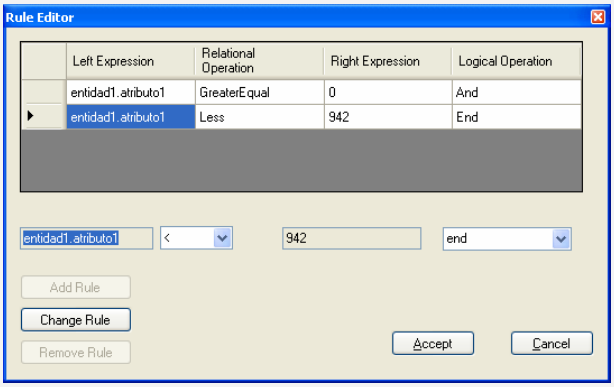


Figura 6-4 Editando las reglas de validación para un atributo

Cualquier atributo que forme parte de una entidad podrá ser seleccionado para formar parte de una expresión a través del selector de atributos (Figura 6-5 y Figura 6-6)

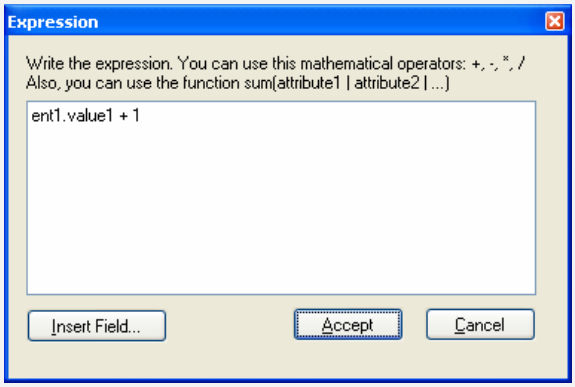


Figura 6-5 Editor de expresiones

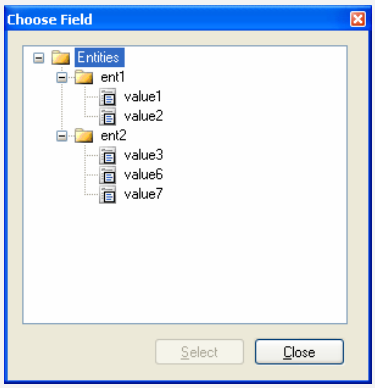


Figura 6-6 Selector de atributos

6.2 Conectores

Las entidades pueden estar vinculadas entre si por medio de conectores. Todo conector tiene una entidad origen, y una entidad destino: la entidad destino no es visible en el formulario generado si las reglas asociadas al conector no se satisfacen. En el momento que se cumplen, la entidad destino es visible; y por lo tanto sus atributos asociados también lo serán.

Los conectores tienen una serie de propiedades que solo afectan la forma en la que son visualizados en la aplicación Smart Proposal, ya que éstos no tienen una representación en el formulario final:

- **Nombre**: Nombre del objeto. El nombre de cada conector/entidad debe ser único.
- **Fuente y color de fuente**: Define el tipo de fuente y el color que se utiliza para presentar el nombre del conector y sus reglas en pantalla.
- **Color**: Color del conector.
- **Ancho de línea**: Fija el valor de ancho de línea que se utiliza para representar el conector.

En la Figura 6-7 puede observarse un conector que vincula a dos entidades, además de sus propiedades asociadas.

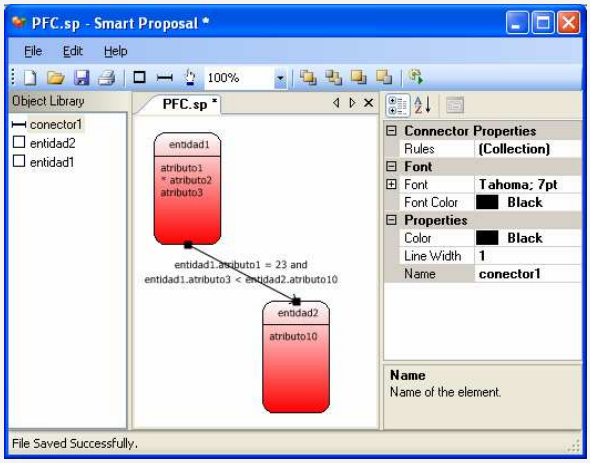


Figura 6-7 Conector definido entre entidad1 y entidad2

La propiedad más relevante es la que permite definir las reglas que condicionan la aparición de la entidad destino del conector en el formulario. Dichas reglas son definidas en función del valor de los atributos de las entidades a través del editor de reglas (Figura 6-8)

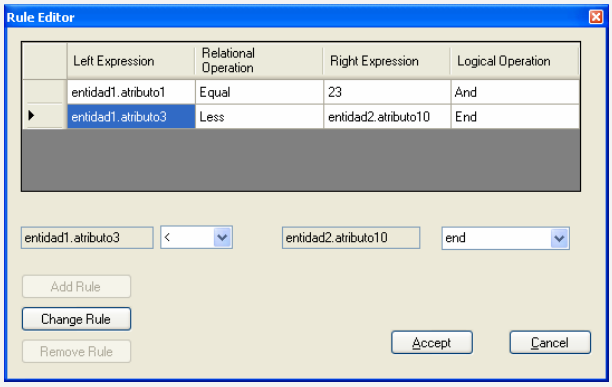


Figura 6-8 Editando las reglas de un conector

6.3 Clases

A nivel de clases, Smart Proposal se encuentra dividido en cuatro espacios de nombres:

- **SP:** Incluye la definición de controles y formularios.
- **SP.Common:** Aquí se definen las clases que realizan operaciones comunes a todo el proyecto.
- **SP.Comp**: Este espacio de nombres incluye las clases del compilador que son utilizadas para analizar las expresiones de las reglas y los valores de los atributos.
- **SP.ObjectModel**: Incluye la definición de las clases de objeto y colecciones utilizados para mantener la interfaz gráfica de la aplicación y la estructura de los documentos. Estos últimos se encuentran en el subespacio `SP.ObjectModel.Draw`

Estos cuatro espacios de nombre y sus componentes son descritos en las secciones 6.4, 6.5, 6.6 y 6.7 respectivamente.

6.4 Espacio de nombres SP

Incluye la definición de todos los controles y formularios de la interfaz gráfica de la aplicación, además de incluir la clase que es punto de entrada en el programa.

Todos los formularios definidos derivan de la clase `Form`, mientras que los controles derivan de la clase `UserControl`. Los diagramas de clases que se presentan en la Figura 6-9 y Figura 6-10 incluyen los métodos más relevantes y ocultan las variables internas para hacerlos más comprensibles y legibles.

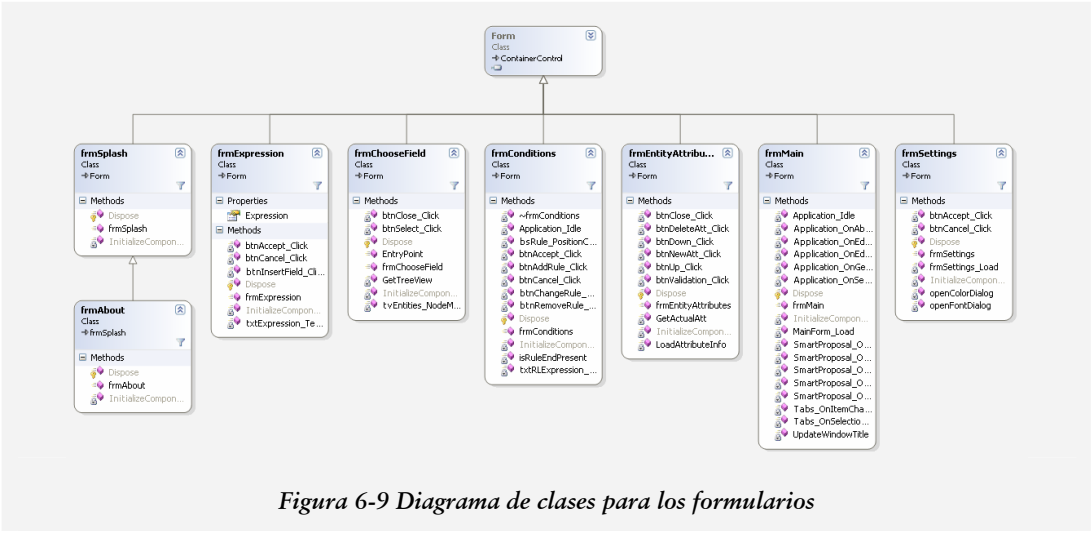


Figura 6-9 Diagrama de clases para los formularios

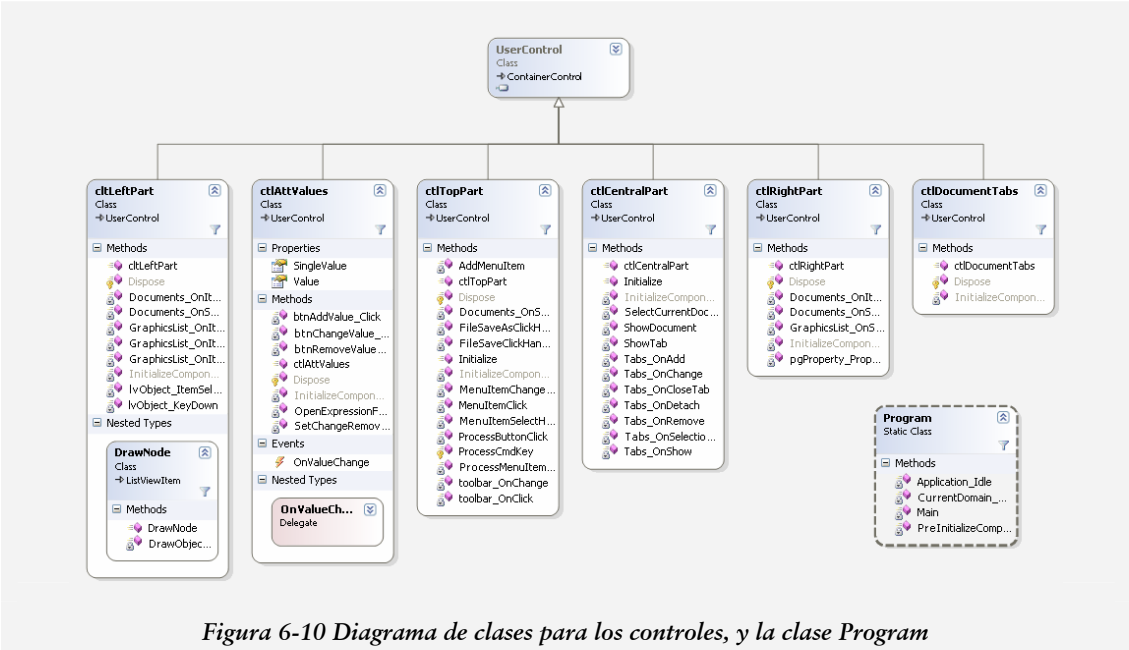


Figura 6-10 Diagrama de clases para los controles, y la clase Program

6.4.1 frmSplash

Formulario que solo contiene imágenes y textos estáticos. Su misión es la de estar visible durante la carga inicial del programa.

6.4.2 frmAbout

Formulario que hereda todo su contenido del formulario `frmSplash`. Este formulario será visible cuando el usuario haga clic sobre la opción de menú **Help — About…**

6.4.3 frmExpression

Editor de expresiones (Figura 6-5). A través de este formulario se realizará la entrada de todas las expresiones que formen parte de los documentos. El editor de expresiones permite realizar construcciones matemáticas de relativa complejidad donde pueden intervenir atributos de cualquier entidad. Los operadores matemáticos permitidos son la suma, la resta, la multiplicación y la división. Además, se ofrece soporte para la función XPath `sum` que permite sumar varios elementos. La principal diferencia ente `sum` y realizar sumas concatenadas es que `sum` trata un valor no numérico como 0.

NOTA: Este comportamiento de la función `sum` *puede variar entre las distintas implementaciones de XForms-XPath. Por ejemplo la extensión para Mozilla Firefox presenta el comportamiento descrito, situación que no se produce con algunas versiones del plug-in para Internet Explorer.*

La propiedad `Expression` del formulario permite obtener o asignar la expresión que se está editando.

El botón **Insert Field…** permite insertar un atributo del documento en cualquier parte de la expresión, con la ayuda del formulario `frmChooseField`.

6.4.4 frmChooseField

Formulario que presenta en una estructura en forma de árbol todas las entidades y atributos del documento, permitiendo la selección de un atributo (Figura 6-6). El orden en el que aparecen las entidades corresponde al orden en que se encuentran en la librería de objetos.

6.4.5 frmConditions

Este formulario (Figura 6-8) es utilizado para crear, editar o eliminar las expresiones condicionales utilizadas en los conectores y en las reglas de validación de los atributos.

El diseñador puede encadenar reglas por medio de los operadores lógicos `AND` y `OR`. No es posible crear subconjuntos de reglas utilizando paréntesis. Por ejemplo la siguiente expresión no puede construirse: `X OR (Y AND Z)`, solo es posible definir una expresión como `X OR Y AND Z`. El orden de evaluación se realiza de derecha a izquierda, por lo que `X OR Y AND Z` equivale a `((X OR Y) AND Z)`

Los operadores relacionales soportados son = (igual), `!=` (distinto), > (mayor), < (menor), >= (mayor igual), <= (menor igual)

Para indicar el final de una expresión condicional debe incluirse el operador `END` en una de las reglas.

6.4.6 frmEntityAttributes

Formulario utilizado para la edición de los atributos de las entidades (Figura 6-3). A través del mismo el diseñador podrá insertar, eliminar o modificar los atributos. Se trata por lo tanto de uno de los puntos clave de la aplicación. Para la edición de los valores de los atributos se incluye un control de tipo `ctlAttValues`. `frmEntityAttributes` se suscribe el evento `OnValueChange` de `cttAttValues` para recibir notificaciones de los cambios realizados.

El orden de los atributos marcará el orden en el que aparecerán en el formulario final generado.

6.4.7 frmMain

Principal formulario de la aplicación (Figura 6-1). Incluye instancias de los controles `ctlCentralPart`, `ctlTopPart` y la barra de estado donde se informa del resultado de las operaciones.

6.4.8 frmSettings

Formulario en el que se definen las opciones globales de la aplicación (Figura 6-11). A través de él se indicará los colores, tipos de fuente y grosor de líneas utilizadas para representar las entidades y conectores, así como para especificar si se desea ver los atributos de las entidades o las reglas definidas en los conectores.

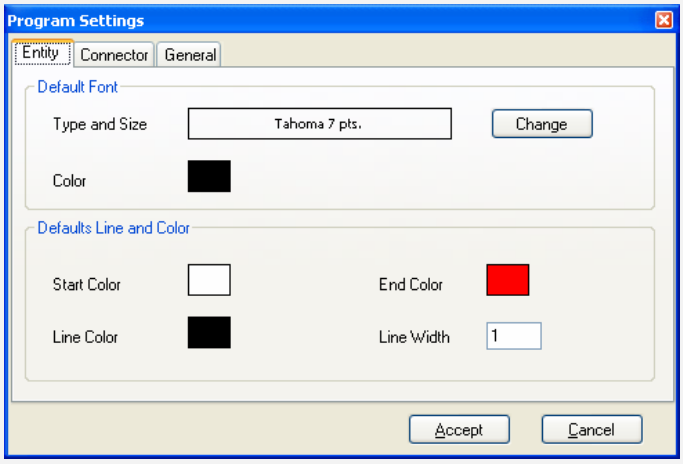


Figura 6-11 Configuración de la aplicación

6.4.9 ctlLeftPart

Este control recibe el nombre de librería de objetos y presenta en una lista todos los objetos (entidades y conectores) del documento que se encuentra activo en la aplicación. El orden en el que aparecen las entidades define el orden en que aparecerán en el formulario final, además de determinar el orden en el que serán representadas en el área de dibujo.

Las acciones que obligan a actualizar el contenido de este control son las siguientes:

- Se añade o elimina un objeto en el documento, por lo que debe añadirse o eliminarse un ítem en la librería de objetos.
- Se cambia el nombre de una entidad o conector.
- Se selecciona un objeto o conjunto de objetos en el área de dibujo.
- El documento activo cambia.

Para detectar todas estas actividades por parte del diseñador sobre los elementos de la aplicación este control se suscribe a los eventos `OnItemCollectionAdd` y `OnSelectionChanged` de la clase `Document`, y a los eventos `OnItemChange`, `OnItemCollectionAdd`, `OnItemCollectionRemove` de la clase `DrawElementCollection`.

Para representar los objetos dentro de la lista se crea una clase interna llamada `DrawNode`, que suscribe al evento `OnSelect` de cada `DrawElement` que forma parte de la `DrawElementCollection`. Así, se reciben notificaciones de los objetos seleccionados por el diseñador en el área de dibujo, seleccionando los nodos correspondientes de la lista.

6.4.10 ctlAttValues

Este control es el encargado del mantenimiento de los valores de los atributos. Para comunicar que un valor ha sido modificado (editado, añadido o eliminado) se realiza una llamada a `OnValueChange`. De esta forma todos los objetos que se suscriban a este evento recibirán notificación y podrán proceder de forma conveniente.

Para añadir, modificar o editar un valor se crea una instancia de la clase `frmExpression`, a la que se pasa el valor actual del atributo en caso de una modificación.

6.4.11 ctlTopPart

`ctlTopPart` es el control que contiene la barra de herramientas y la barra de menú. Este control se suscribe al evento `OnSelectionChanged` de `DocumentCollection` para detectar cuando cambia el documento seleccionado en la ventana central con el objetivo de actualizar la lista desplegable que almacena el valor de zoom.

6.4.12 ctlCentralPart

Contiene instancias de otros tres controles: `ctlLeftPart`, `ctlDocumentTabs` y `ctlRightPart`. Es el encargado de gestionar las pestañas que forman parte del control `ctlDocumentTabs`. Para ello recibe información de los eventos `OnItemCollectionAdd`, `OnItemChange`, `OnItemClose`, `OnItemDetach`, `OnItemCollectionRemove`, `OnSelectionChanged` y `OnItemShow` de la clase `TabCollection`; y de los eventos `OnItemShow` y `OnItemCollectionRemove` de la clase `DocumentCollection`.

De esta forma es posible:

- Crear una nueva pestaña cuando un documento es creado.
- Eliminar una pestaña cuando un documento es cerrado.
- Actualizar el título de la pestaña cuando las modificaciones de un documento no han sido guardadas (presentando un asterisco junto a su nombre)
- Hacer visible la pestaña a la que pertenece el documento activo.

6.4.13 ctlRightPart

Contiene un control de tipo `PropertyGrid` para permitir la visualización y edición de las propiedades del objeto actualmente seleccionado.

Este control deberá actualizar su contenido cuando cambie el documento activo o la selección de objetos en el área de dibujo. Por ello, se suscribe a los eventos `OnItemCollectionAdd` y `OnSelectionChanged` de la clase `DocumentCollection`; y al evento `OnSelectionChanged` de la clase `DrawElementCollection`.

Este control no soporta la edición de propiedades sobre una selección de múltiple objetos: sólo podrán editarse las propiedades de objetos individuales.

6.4.14 ctlDocumentTabs

Este control posee un contenedor de pestañas donde se alojan los distintos documentos de la aplicación. Toda la funcionalidad es mantenida desde el control `ctlCentralPart`.

6.4.15 Program

Esta clase estática es el punto de entrada de la aplicación. El método `Main` se encarga de hacer visible el formulario `frmSplash` mientras se realizan las tareas de inicialización. Una vez la aplicación está lista para ejecutarse (notificación recibida al suscribirse al método `Idle` del objeto `Application` de `WindowsForms`) se carga el formulario `frmMain` y se cierra el formulario `frmSplash`.

6.4.16 Settings

Clase del sistema que permite gestionar de forma sencilla las preferencias y configuraciones de la aplicación utilizadas en `frmSettings`.

6.5 Espacio de nombres SP.Common

Incluye la definición de las clases y datos de tipos enumerados comunes al resto de espacios de nombres. (Figura 6-12 y Figura 6-13)

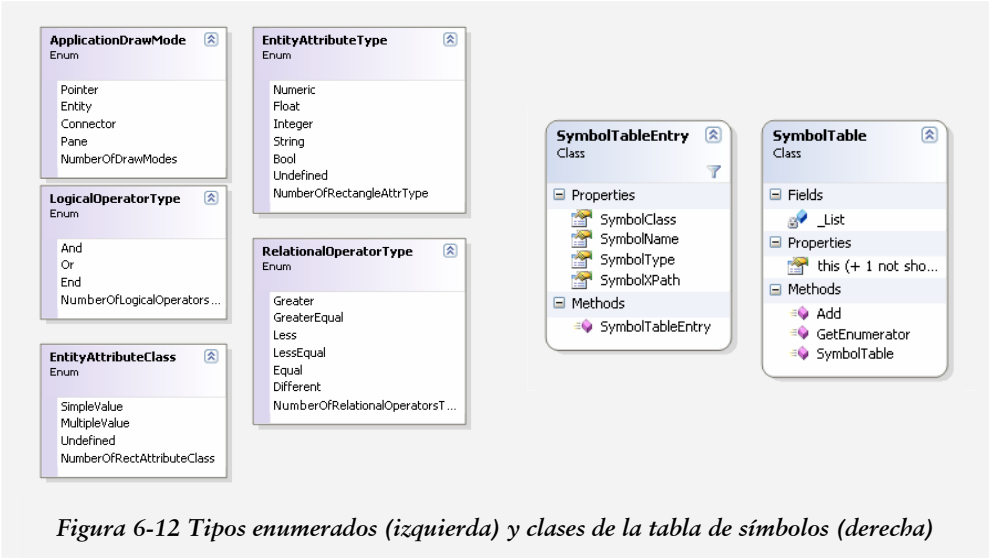


Figura 6-12 Tipos enumerados (izquierda) y clases de la tabla de símbolos (derecha)

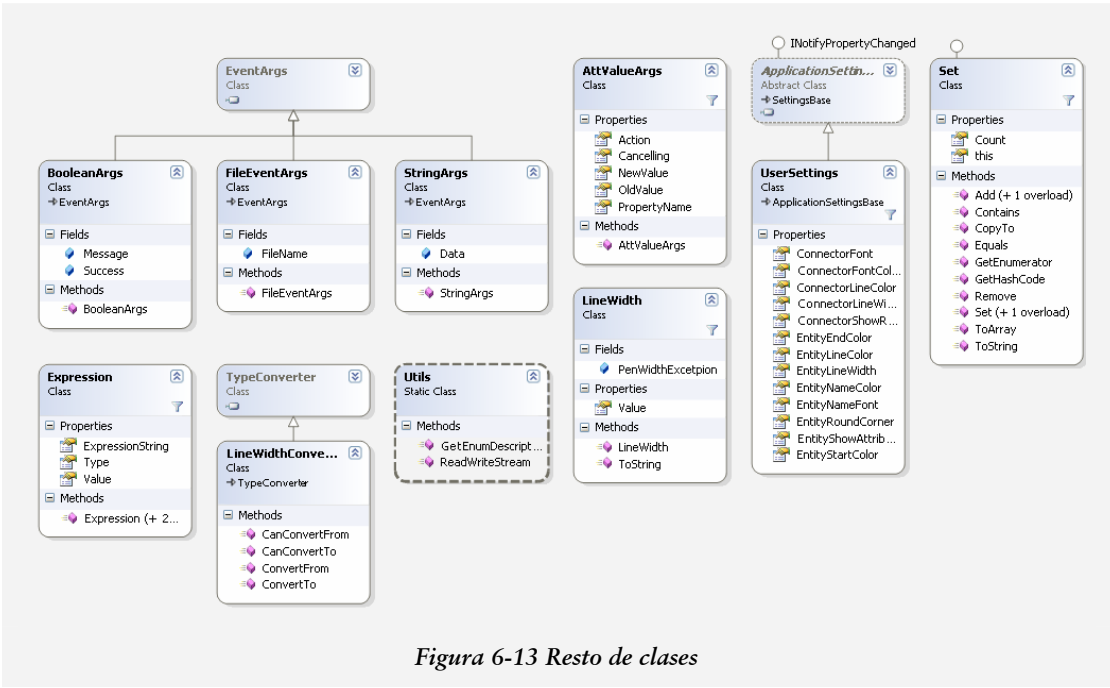


Figura 6-13 Resto de clases

6.5.1 ApplicationDrawMode

Enumera los distintos modos de dibujo en que se puede encontrar la aplicación:

- **Entity**: cuando se encuentra pulsado el botón dibujar entidad de la barra de herramientas.
- **Connector**: cuando se encuentra pulsado el botón dibujar conector de la barra de herramientas.
- **Pane**: cuando se encuentra pulsado el botón de desplazamiento de la barra de herramientas.
- **Pointer**: cuando no se encuentra pulsado ningún botón de dibujo en la barra de herramientas.

6.5.2 EntityAttributeType, EntityAttributeClass

Enumeran los posibles tipos de datos y clases de los atributos de las entidades.

6.5.3 LogicalOperatorType, RelationalOperatorType

Enumeran los distintos operadores lógicos y relacionales que pueden utilizarse en las expresiones condicionales.

6.5.4 SymbolTable, SymbolTableEntry

Estas clases permiten materializar una tabla de símbolos con sus correspondientes entradas. La tabla de símbolos es utilizada de forma intensiva por el compilador quien es el encargado de analizar la validez sintáctica y semántica de las expresiones que el diseñador utiliza en el documento. Además, interviene en la generación del formulario final, tal como se verá más adelante.

`SymbolTableEntry` define una entrada de la tabla de símbolos. Un símbolo está caracterizado por un nombre, un tipo (entero, cadena,..), una clase (valor simple o múltiple) y una expresión XPath. Este último valor permite modificar expresiones de forma tal que el nombre de un atributo pueda ser reemplazado por su expresión XPath correspondiente; facilitando la generación del formulario.

6.5.5 EventArgs, BooleanArgs, FileEventArgs, StringArgs, AttValueArgs

Estas clases son utilizadas para pasar como parámetros objetos en las llamadas a eventos.

Especial mención recibe la clase `AttValueArgs`, utilizada como parámetro en los eventos que implican el cambio de valor de ciertas propiedades, como por ejemplo `OnItemChange` de la clase `DrawElementCollection`.

Los objetos de la clase `AttValueArgs` incluyen cinco propiedades:

- `Action`: cadena que describe la acción realizada.
- `Cancelling`: valor booleano que permite cancelar la acción que se está llevando a cabo.
- `PropertyName`: cadena con el nombre de la propiedad cuyo valor es cambiado.
- `NewValue`: nuevo valor de la propiedad afectada.
- `OldValue`: antiguo valor de la propiedad afectada.

6.5.6 Set

Esta clase implementa el concepto de conjunto y sus operaciones más comunes. Como todo conjunto, esta colección de valores no incluye elementos repetidos. Además implementa las operaciones básicas: añadir y eliminar elementos, determinar si un elemento es parte del conjunto, conversión a array, etc.

Esta clase es utilizada en la generación del formulario: En un objeto de tipo `Set` se almacenan las dependencias entre atributos como resultado de su inclusión en expresiones de cálculo.

6.5.7 Expression

Esta clase representa una expresión a través de sus tres propiedades:

- `ExpressionString`: Cadena con el desarrollo de la expresión. Por ejemplo `17 + ent1.att1`.
- `Type`: Tipo de la expresión. Deberá ser un valor de la enumeración `EntityAttributeType`.
- `Value`: Valor de la expresión. Si `ent1.att1` tiene un valor de `5`, el valor de `17 + ent1.att1` será de `22`.

Objetos de esta clase son utilizados para representar expresiones en las reglas condicionales y en los valores de los atributos.

6.5.8 LineWidth

Clase que representa el valor de ancho de línea con el que se representan las entidades y conectores en el área de dibujo. Debido a que el valor de esta propiedad debe ser almacenado en disco al momento de guardar un documento, es necesario que sea serializable.

6.5.9 LineWidthConverter

Esta clase permite la conversión de objetos `LineWidth` a otros tipos de clases: `int` y `string`. Deriva directamente de la clase de sistema `TypeConverter`, que al no ser serializable obliga a implementar la conversión en una clase separada y no dentro de la propia clase `LineWidth`.

6.5.10 Utils

Clase estática que implementa dos métodos:

- `ReadWriteStream`: copia el contenido proveniente de un stream de lectura a un stream de escritura (ambos pasados como parámetros). Este método se utiliza al momento de generar el formulario para crear los ficheros auxiliares y que se encuentran almacenados como recursos embebidos en el ensamblado, como por ejemplo la hoja de estilo CSS.
- `GetEnumDescription`: Devuelve la descripción asociada a un tipo enumerado. La descripción es meta-información vinculada al elemento enumerado, y definida de la siguiente forma:

```
public enum RelationalOperatorType
{
    [Description(">")]        Greater=0,
    [Description(">=")]       GreaterEqual,
    [Description("<")]        Less,
    [Description("<=")]       LessEqual,
    [Description("=")]        Equal,
    [Description("!=")]       Different,
    NumberOfRelationalOperatorsType
}
```

6.5.11 UserSettings

Clase que deriva de `ApplicationSettings` (que a su vez deriva de la clase `ApplicationBase`) utilizada para almacenar las opciones de la aplicación que afectan la forma en que se representarán las entidades, atributos y conectores en el área de dibujo.

6.6 Espacio de nombres SP.Comp

En este espacio de nombres se encuentran definidos todas las clases y elementos necesarios para el análisis sintáctico, semántico y para la reescritura de expresiones. (Figura 6-14)

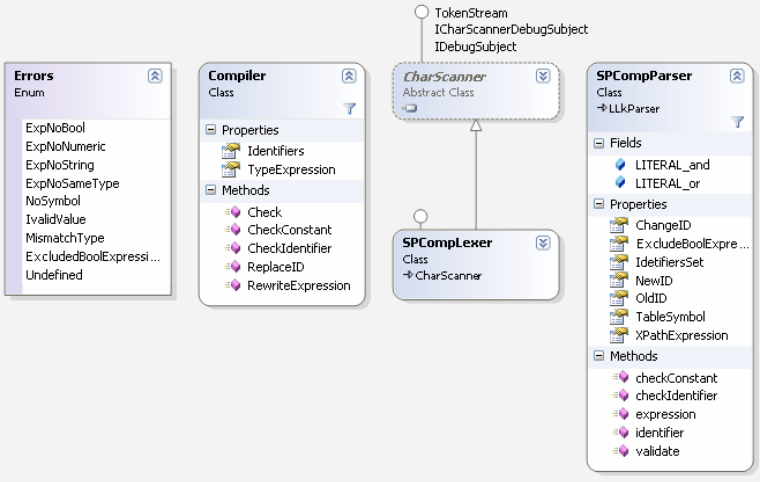


Figura 6-14 Clases definidas en SP.Comp

Estas acciones son llevadas a cabo con la ayuda del framework ANTLR que permite de forma rápida y sencilla definir e integrar en la aplicación un analizador de expresiones que además brinda soporte para la reescritura de expresiones.

¿Pero para qué es necesaria la reescritura? Para reemplazar la aparición de identificadores del tipo `entidad1.atributo2` por su correspondiente expresión XPath: por ejemplo `/document/entidad1/attributes[2]/atributo2`.

Esta operación no podría realizarse de forma fácil sin la ayuda de ANTLR. Una primera aproximación podría ser reemplazar cada ocurrencia de texto por su correspondiente expresión XPath. Esta solución no es viable ya que dentro de una expresión la ubicación de una cadena de texto puede jugar distintos papeles según su contexto.

Por ejemplo, en la expresión

```
entidad1.atributo2 = “entidad1.atributo3 + 45”
```

el valor del atributo2 es la cadena `entidad1.atributo3 + 45`, mientras que en la expresión

```
entidad1.atributo2 = entidad1.atributo3 + 45
```

el valor de `atributo2` corresponde al resultado de evaluar el valor de `atributo3` y sumarle `45`. Si hubiésemos reemplazado literalmente todos los valores de la cadena `entidad1.atributo3` el resultado hubiese sido erróneo para la expresión

```
entidad1.atributo2 = “entidad1.atributo3 + 45” => se convierte a
entidad1.atributo2 = “/document/entidad1/attributes[2]/atributo3 + 45”
```

El compilador acepta expresiones de acuerdo a un conjunto de expresiones regulares representadas mediante los diagramas sintácticos presentados en la Figura 6-15.

Diagrama expression

Diagrama exp1

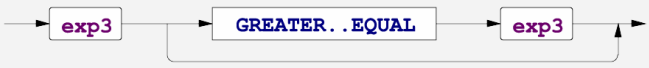


Diagrama exp2

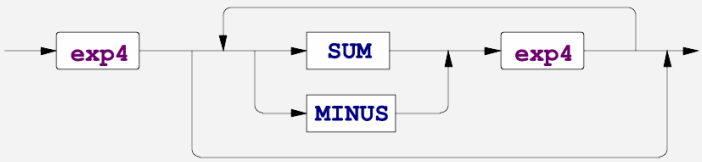


Diagrama exp3

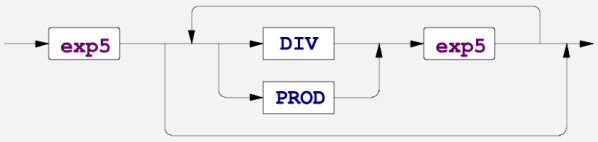


Diagrama exp4

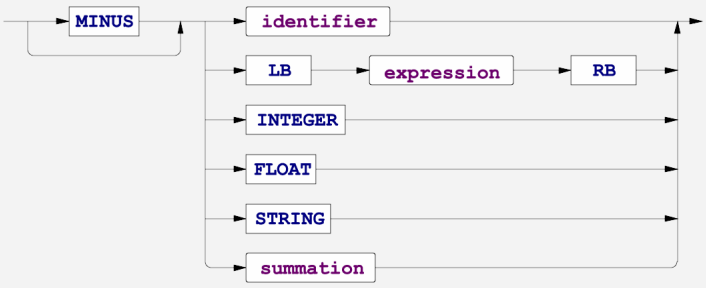


Diagrama exp5

Diagrama summation

Diagrama checkConstant

Diagrama identifier

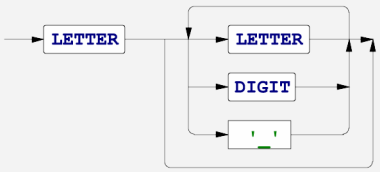


Diagrama IDENT

Figura 6-15 Diagramas sintácticos

6.6.1 Errors

Define las distintas categorías de errores que pueden presentarse al momento de verificar las expresiones.

6.6.2 Compiler

Clase que define los métodos y propiedades utilizados para comprobar la validez de las expresiones.

- `Check`: verifica la validez de la expresión pasada como parámetro, a través de la llamada a la regla `expression`.
- `CheckConstant`: comprueba si la expresión pasada como parámetro corresponde a una constante numérica o de tipo cadena.

- `CheckIdentifier`: verifica si la cadena pasada como parámetro es válida para ser utilizada como identificador.
- `ReplaceID`: reemplaza todas las ocurrencias de un identificador por otro en la expresión pasada como parámetro.
- `RewriteExpression`: reemplaza todas las ocurrencias de un identificador de la forma "`entidad.atributo`" o "`entidad`" por su expresión XPath correspondiente.

6.6.3 SPCompParser y SPCompLexer

Clases generadas por el framework ANTLR de forma automática a partir de las expresiones definidas en el fichero fuente de la gramática.

6.7 Espacio de nombres SP.ObjectModel

La idea principal es diseñar la aplicación con una arquitectura flexible siguiendo el patrón de diseño Modelo-Vista-Controlador (MVC) presentado en la Figura 6-16. Siguiendo este modelo se consigue un código claro, ya que se desacopla la lógica de la aplicación de la lógica de la interfaz de usuario.

Básicamente, los principios del patrón de diseño MVC son:

- Existe un modelo que contiene los datos.
- Existe al menos un controlador que hace los cambios sobre el modelo.
- Existe al menos una vista que muestra la salida por la interfaz gráfica. El controlador utiliza la vista para mostrar la salida del modelo.
- Controladores y vistas observan el modelo a la espera de cambios. Cuando un cambio es detectado, la vista los refleja en la interfaz de usuario y el controlador actualiza la vista y/o el modelo.
- La vista recibe la interacción del usuario y refleja los cambios en el modelo a través del controlador.

Resumiendo:

- **Modelo**: Representa los datos y la lógica de negocio que determina como esos datos son actualizados.
- **Vista**: Representa el contenido del modelo y determina como debe ser enseñado. De la vista es la responsabilidad de mantener la consistencia entre la presentación y el contenido del modelo cuando es actualizado. Esto se puede conseguir siguiendo un modelo push donde el modelo envía notificaciones a la vista cada vez que se produce un cambio, o un modelo pull donde la vista es la encargada de solicitar el estado del modelo cada vez que necesita representar los datos.
- **Controlador**: Es el encargado de transformar las interacciones del usuario sobre la vista en acciones sobre el modelo. Estas acciones incluyen la llamada a procesos o a cambios en el estado del modelo. Dependiendo de las acciones del usuario y del resultado del modelo, el controlador responde seleccionando la vista apropiada.

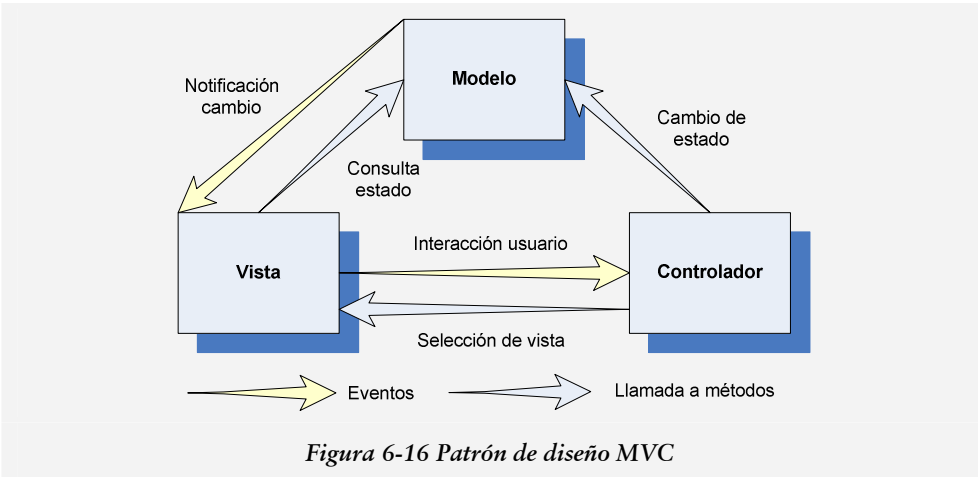


Figura 6-16 Patrón de diseño MVC

La arquitectura seguida para la construcción de Smart Proposal es similar a la arquitectura utilizada por el modelo de programación de las aplicaciones Office de Microsoft, donde un conjunto de clases reposa sobre una clase base llamada `Application`. (Figura 6-17)

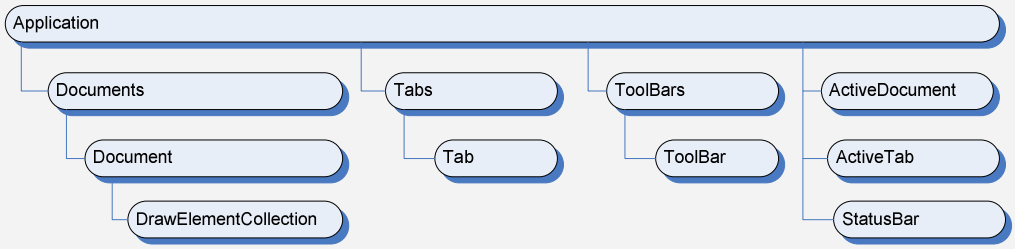


Figura 6-17 Modelo de objetos

Desde `Application` se puede acceder por medio de código a cualquier documento, pestaña, barra de herramientas, mostrar mensajes en la barra de estado, etc. Por ejemplo, para terminar la aplicación se debe llamar al método `Application.Quit()`, mientras que para hacer visible la primera pestaña se llamaría al método `Application.Tabs[0].Show()`. Si se desea cambiar el nombre del documento actual, solo es necesario cambiar el valor de la propiedad `Name` del objeto `ActiveDocument`. Ante cualquiera de estos cambios, la interfaz de usuario responderá y se actualizará automáticamente.

Por ejemplo, en Smart Proposal en el momento que se cierra un documento se desencadenan una serie de eventos: se actualiza la barra de tareas y el título de la ventana, se cierra la pestaña del documento, ciertos botones de la barra de menú se deshabilitan… Todos estos detalles no se han tenido en cuenta al momento de desarrollar la aplicación, sino que se explotan las ventajas del modelo MVC. La clave es no pensar en todo momento en la interfaz de usuario, ya que la responsabilidad de implementar ciertas características es distribuida. Esto quiere decir –por ejemplo- que el módulo del área de dibujo no tiene que saber de la existencia del módulo de la librería de objetos y viceversa. Todo lo que necesitan saber es que existe un modelo que incluye objetos que enviarán notificaciones cuando algo les ocurra.

En SmartProposal, cuando una nueva entidad es añadida al documento ocurre la siguiente secuencia de acciones:

- El área de dibujo recibe la notificación de que una nueva entidad ha sido añadida a la colección de objetos del documento, por lo que crea una representación para ella sobre el área de edición.
- La librería de objetos recibe la notificación de que una nueva entidad ha sido añadida a la colección de objetos del documento, por lo que crea un nuevo nodo en la lista de objetos.

Diferentes módulos de la aplicación se suscriben a diferentes colecciones y objetos expuestos por el objeto `Application` y escuchan las notificaciones relevantes para ellos. Cuando una notificación es recibida, se ejecutan las acciones asociadas a ella.

La clave en la implementación de este modelo se encuentra en las clases presentadas en la Figura 6-18, y de las cuales derivan la gran mayoría de clases de la aplicación.

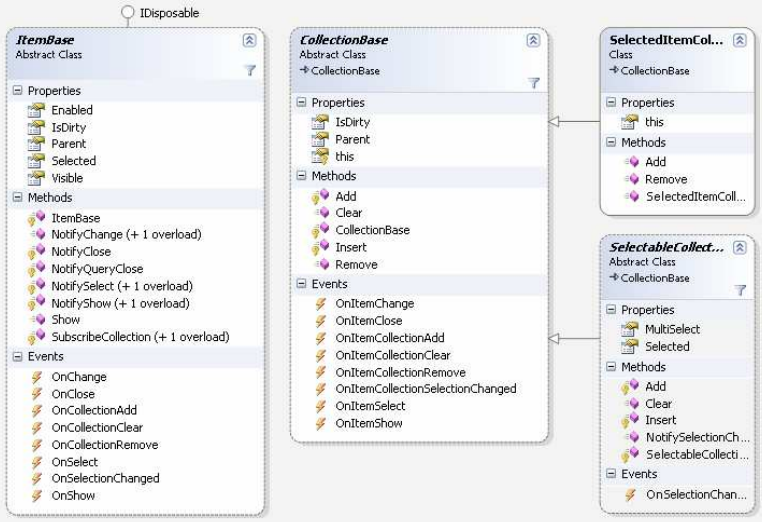


Figura 6-18 Clases base

6.7.1 Modelo de objeto base

El modelo de objeto base se construye a partir de tres clases: `ItemBase`, `CollectionBase` y `SelectableCollectionBase`. Estas tres clases implementan la característica más importante del modelo: el modelo de objeto observable. Al derivar una clase desde cualquiera de ellas, se pueden observar los cambios que se producen en sus objetos. Por ejemplo, la clase `DocumentCollection` hereda de la clase `CollectionBase`. `CollectionBase` contiene todo el código necesario para disparar eventos cuando un elemento es añadido, eliminado, etc. Además, la clase colección escucha los cambios producidos en todos los ítems que contiene. Como resultado, ella puede capturar eventos lanzados por elementos hijos y relanzarlos a sus observadores.

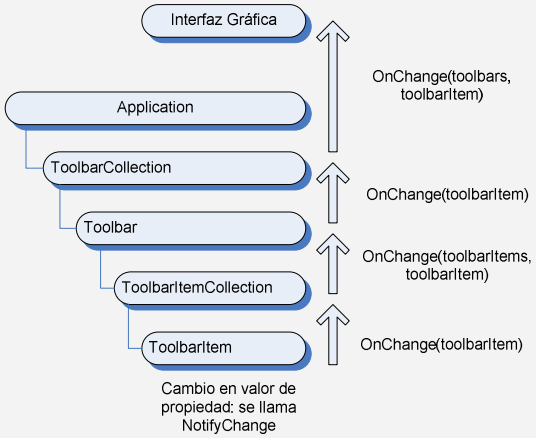


Figura 6-19 Eventos desde elementos hijos a elementos padres

La Figura 6-19 muestra como los eventos se disparan desde los objetos hijos a sus padres. De esta forma si un objeto se suscribe para escuchar los eventos de una colección, recibirá notificaciones desde cualquier elemento que la misma contenga sin importar el nivel de jerarquía en el que se encuentre.

6.7.1.1 ItemBase

Sobre esta clase se construyen otras como `Document`, `DrawElement`, etc. De ella derivan las clases de cuyos objetos se desea recibir notificación al producirse algún cambio.

Contiene un método clave llamado `SubscribeCollection`, que para una colección se suscribe a todos sus eventos. Este método es necesario para todas aquellas clases como `Toolbar` que

contiene su propia colección `ToolbarItemCollection`. Para poder relanzar eventos hacia arriba en la jerarquía de clases es necesario capturar todos los eventos generados desde las colecciones "hijas".

6.7.1.2 CollectionBase

La función de esta clase es la de agrupar objetos que derivan de `ItemBase` y proporcionar soporte para relanzar eventos de los objetos contenidos. Ejemplos de este tipo de colecciones son `DocumentCollection` y `ToolbarCollection`.

Cuando un objeto es añadido a la colección, ésta se suscribe a todos los eventos expuestos por `ItemBase`. Como resultado, donde sea que un evento sea disparado por un objeto contenido en la colección, primero se recibe la notificación y luego se dispara el evento a través de sus propios métodos. Por ejemplo si se recibe el evento `OnShow` desde un ítem, la colección dispara el evento `OnItemShow(item)`. Esto proporciona una característica realmente útil: poder suscribirse a una colección en lugar a los objetos individuales, y recibir notificaciones de los eventos de todos ellos.

6.7.1.3 SelectableCollectionBase

Esta clase extiende `CollectionBase` y expone la propiedad `Selected` y la colección `SelectedItems`. `ItemBase` contiene la propiedad `Selected`. Cuando su valor es `true`, si el ítem forma parte de una `SelectableCollection`, la propiedad `Selected` de la colección toma el valor del ítem, además de ser añadido a la colección `SelectedItems` de la clase `Collection`.

Existen dos modos: selección simple y selección múltiple. En el primer modo como máximo puede ser seleccionado un solo ítem. Tan pronto como la colección recibe un evento de cambio de estado para la propiedad `Selected` de algún ítem, asigna a la propiedad `Selected` del ítem anteriormente seleccionado el valor `false`, y establece como selección actual el nuevo objeto.

El modo de selección múltiple se comporta de forma similar, y permite la selección de múltiples objetos para una colección.

6.7.2 Modelo de objeto de la aplicación

Tal como se ha comentado, Smart Proposal tiene un modelo de objetos particular, cuya raíz es el objeto `Application` que pertenece al espacio de nombres `SP.ObjectModel`. La mayoría de clases de la aplicación (Figura 6-20, Figura 6-21 y Figura 6-22) derivan de las tres clases base comentadas en el apartado anterior: `ItemBase`, `CollectionBase` y `SelectableCollectionBase`.

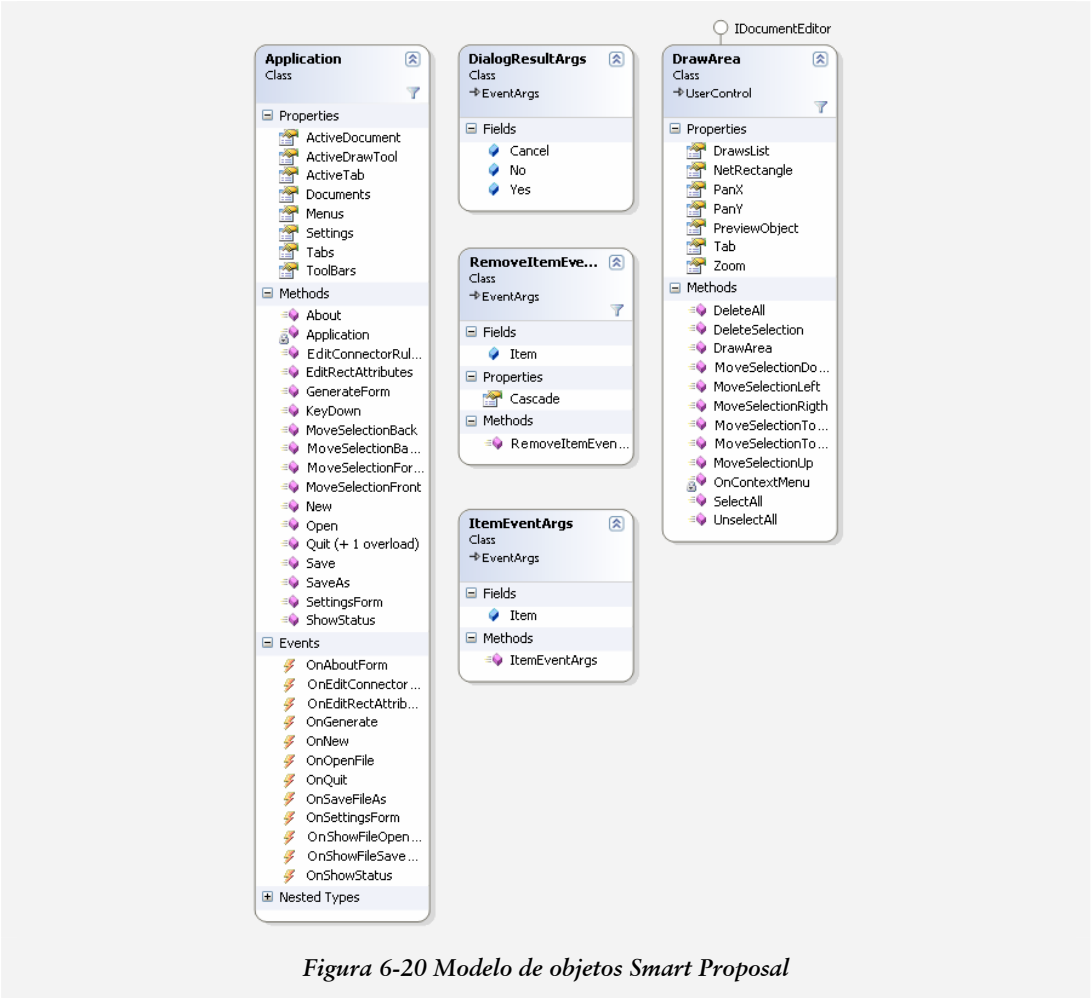


Figura 6-20 Modelo de objetos Smart Proposal

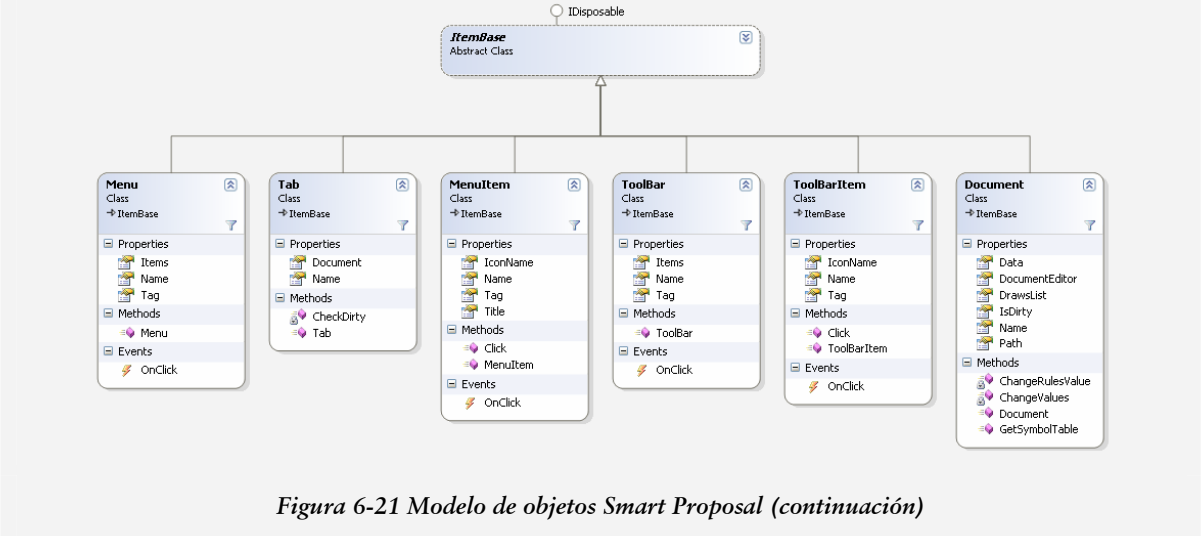


Figura 6-21 Modelo de objetos Smart Proposal (continuación)

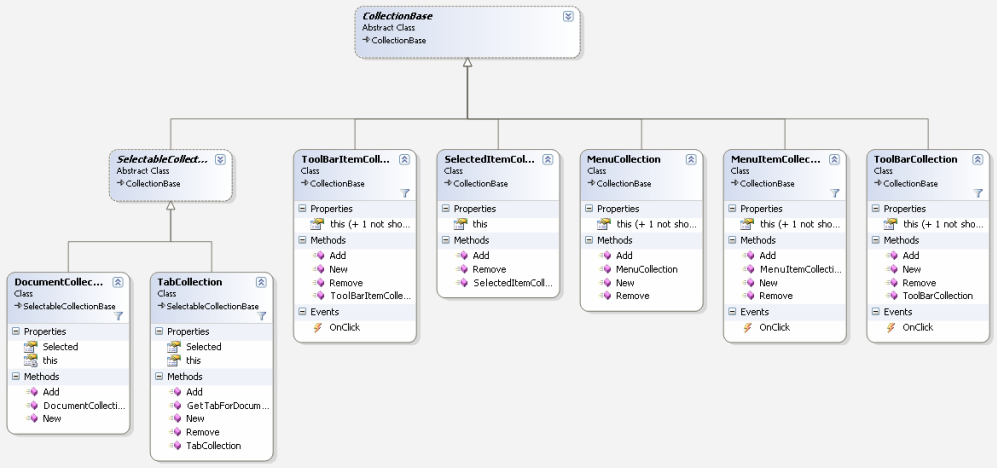


Figura 6-22 Modelo de objetos Smart Proposal (continuación)

6.7.2.1 Application

Esta clase es la contenedora del resto de objetos que forma parte de la aplicación. Sus propiedades son:

- `ActiveDocument`: Contiene la instancia del documento activo.
- `ActiveDrawTool`: Indica la herramienta de dibujo activa.
- `ActiveTab`: Contiene la instancia de la pestaña activa.
- `Documents`: Esta colección almacena las instancias de todos los documentos abiertos en la aplicación.
- `Menus`: Colección que contiene todos los ítems de la barra de menú.
- `Settings`: Objeto de clase `UserSettings` que recoge la configuración general de la aplicación.
- `Tabs`: Objeto de tipo colección que contiene todas las pestañas abiertas en la aplicación.
- `ToolBars`: El objeto `ToolBars` almacena las instancias de las barras de herramientas de la aplicación. Smart Proposal solo cuenta con una barra de herramientas, por lo que esta colección solo almacena un objeto.

Los métodos de esta clase permiten efectuar operaciones como salvar, abrir y crear documentos (`Save`, `SaveAs`, `Open`, `New`), cerrar la aplicación (`Quit`), entre otros. Cada uno de estos métodos realiza la llamada de sus eventos correspondientes (`OnSaveFileAs`, `OnQuit`…) para notificar de la acción realizada a los objetos que así lo han requerido. La mayoría de veces la única acción que realizan los métodos es la llamada a los eventos, siendo los objetos que se suscriben a ellos los que implementan la acción. Por ejemplo, el método invocado para generar el formulario XForms en la clase `Application` es el siguiente:

```
public static void GenerateForm()
{
    if (OnGenerate!= null)
        OnGenerate();
}
```

El encargado de implementar esta funcionalidad recibirá la notificación de que se ha pulsado el botón correspondiente en la barra de herramientas y procederá a la generación del formulario. Esto es beneficioso ya que permite desacoplar las funciones de los distintos elementos de la aplicación, conociendo solo aquellos detalles de implementación que son necesarios para cada clase.

6.7.2.2 DialogResultArgs

Los objetos de esta clase son utilizados como parámetros en la llamada de eventos para informar de la acción escogida por el usuario cuando interviene un cuadro de diálogo.

6.7.2.3 ItemEventArgs

De forma similar a los objetos de la clase `DialogResultArgs`, los objetos de la clase `ItemEventArgs` son utilizados como parámetros en los eventos. La propiedad `Item` contiene la instancia del objeto que interviene en la llamada (por ejemplo un documento, una opción de menú, un atributo, etc.)

6.7.2.4 RemoveItemEventArgs

Los objetos de esta clase son utilizados al momento de eliminar un objeto de una colección. La propiedad `Cascade` tiene sentido cuando se utiliza para las entidades: indica si los objetos de las colecciones que contiene (conectores de entrada y salida) también deben eliminarse en caso que la entidad es eliminada.

6.7.2.5 DrawArea

Esta clase deriva de `UserControl` y representa el área de dibujo para un documento. Con ella el usuario interactuará para añadir, eliminar o modificar entidades y conectores. Además podrá modificar la disposición de los objetos y la forma de visualizarlos (zoom, pan, etc.) según sus preferencias.

Todo documento tiene asociada una colección de dibujos llamada `DrawsList`. El área de dibujo asociada al documento realizará todas las modificaciones sobre dicha colección, de la cual contiene una propiedad con el mismo nombre que hace referencia a ella.

La propiedad `NetRectangle` se utiliza para dibujar un rectángulo de selección discontinuo que aparece al hacer clic en el área de dibujo cuando no está activa ninguna herramienta de dibujo.

Las propiedades `PanX`, `PanY` y `Zoom` se utilizan para modificar que parte del documento es visible en el área de dibujo.

`PreviewObject` se utiliza al momento de dibujar nuevos elementos: si la acción realizada es correcta, el `PreviewObject` se convierte en elemento de la colección de objetos del documento. En caso que la acción no sea una acción permitida (como por ejemplo conectar dos entidades con más de un conector en el mismo sentido) el objeto dibujado no formará parte de la colección `DrawsList`.

Los métodos expuestos por esta clase permiten manipular los distintos objetos de la colección: eliminarlos, seleccionarlos, cambiar el orden en que se dibujan en pantalla, etc.

6.7.2.6 Menu y Toolbar

Los objetos de la clases `Menu` representan las distintas opciones de menú (File, Edit…), mientras que los objetos de la clase `Toolbar` representan las barras de herramientas de la aplicación.

Para la clase `Menu` la propiedad `Items` contiene los elementos de cada opción de menú (en el caso de File: Open, Save…; en el caso de Edit: Select All, Delete…), mientras que para la clase `Toolbar` contiene los objetos que representan los botones contenidos en la barra de herramientas. Para ambas clases, la propiedad `Name` almacena el nombre de cada opción de menú/botón.

6.7.2.7 MenuItem y ToolBarItem

Los elementos almacenados en la propiedad `Items` de la clase `Menu` son de tipo `MenuItem`. Esta clase incluye las propiedades básicas que puede incluir un ítem de menú: su nombre, el icono asociado y el texto que será visible.

De forma similar, los elementos almacenados en la propiedad `Items` de la clase `Toolbar` son de tipo `ToolbarItem`. En este caso, las propiedades más relevantes son el nombre del botón y su imagen asociada.

6.7.2.8 Tab

Los objetos de esta clase representan las distintas pestañas de la aplicación. Cada una de ellas alojará un documento, al que se hace referencia mediante la propiedad `Document`. El método `CheckDirty` verifica si el documento debe ser guardado a la hora de cerrar la pestaña.

6.7.2.9 Document

Esta clase almacena toda la información relacionada con un documento de la aplicación. La propiedad `DrawsList` es la colección de entidades y conectores del documento.

`DocumentEditor` es una instancia del editor (de tipo `DrawArea`) utilizado para editar el documento.

`IsDirty` es la propiedad que indica si el documento ha sido modificado desde la última vez que ha sido guardado. Cualquier modificación realizada sobre una entidad o un conector establecerá el valor de `IsDirty` a `true`.

`Name` almacena el nombre del documento, que al momento de ser creado es Untitled.

La propiedad `Path` indica la ruta del documento.

Los métodos `ChangeRulesValue` y `ChangeValues` son llamados cuando un objeto (conector o entidad) es cambiado de nombre, por lo que todas las apariciones del mismo deberá ser cambiado en las reglas y valores de los atributos.

El método `GetSymbolTable` devuelve la tabla de símbolos asociada al documento. Dicha tabla es utilizada al momento de generar el formulario, y cuando se realizan validaciones de expresiones introducidas por el usuario.

6.7.3 Modelo de objeto de datos

En este apartado se engloban todas las clases cuyos objetos el usuario manipula en el proceso de construcción de un formulario. (Figura 6-23 y Figura 6-24)

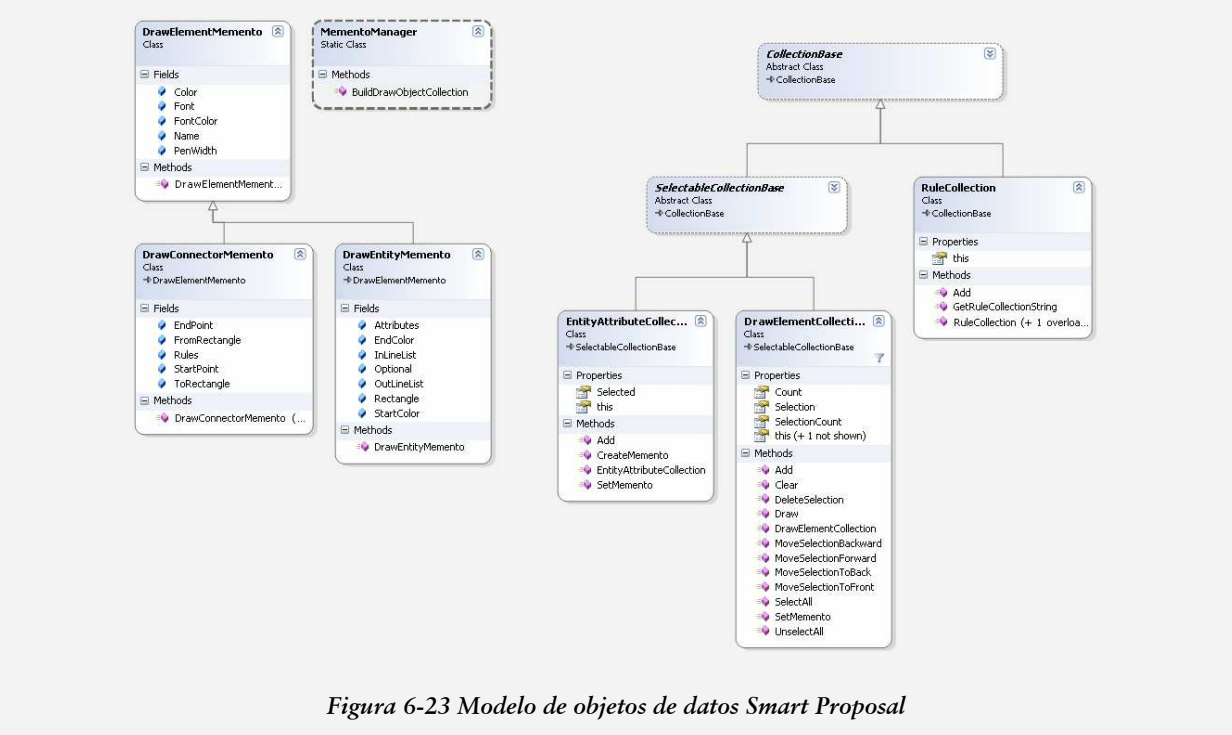


Figura 6-23 Modelo de objetos de datos Smart Proposal

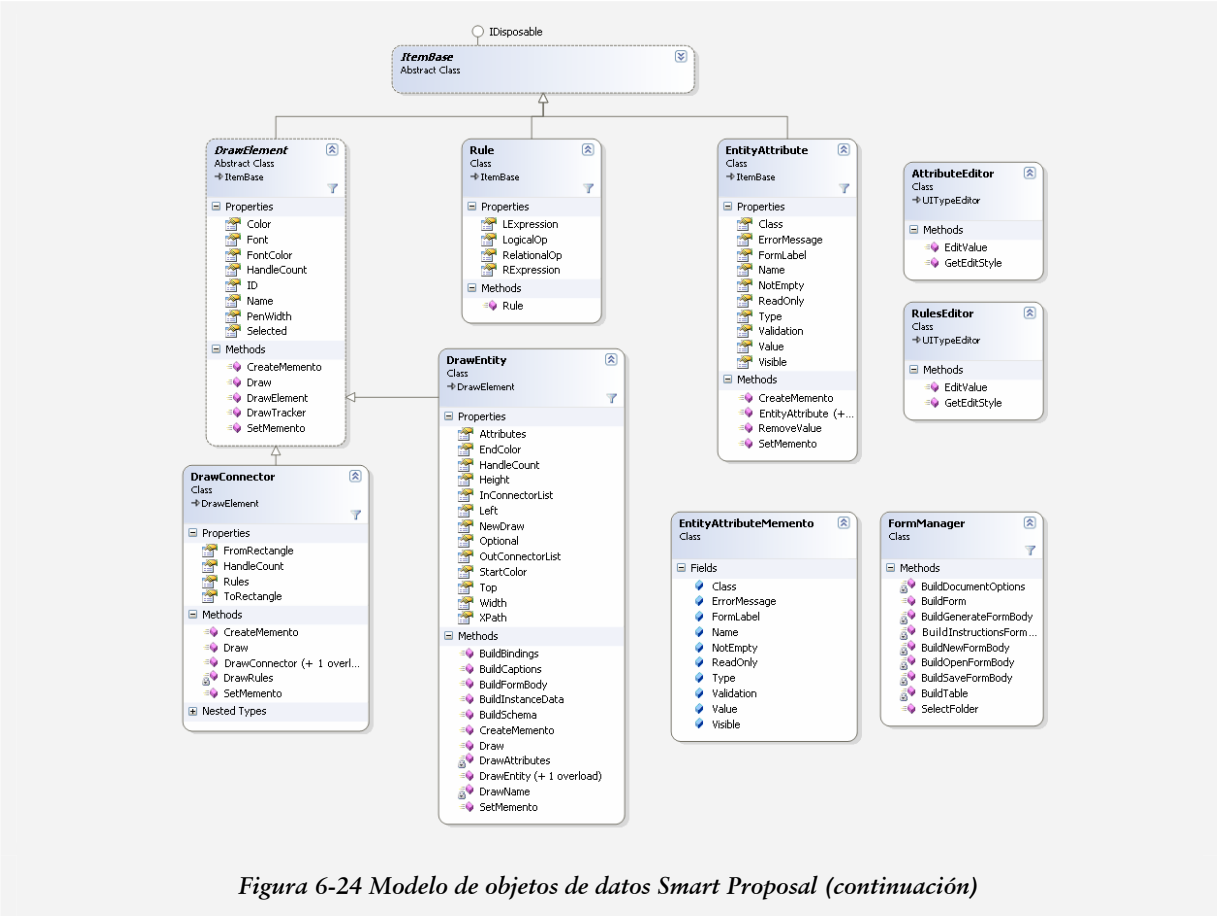


Figura 6-24 Modelo de objetos de datos Smart Proposal (continuación)

Para la implementación de la funcionalidad de manejo de ficheros (abrir/guardar) fue necesaria la intervención del patrón de diseño Memento, el cual se explica a continuación.

6.7.3.1 El patrón de diseño Memento

Cuando es necesario persistir un objeto, aparece el siguiente problema: algunos o todos los estados internos del objeto son privados, por lo que no hay forma de obtener estos valores sin violar la propiedad de encapsulación del paradigma de programación orientado a objetos.

Pero la técnica de serialización de .NET permite persistir objetos en ficheros o streams implementando la interfaz `ISeralizable`. Sin embargo, existen objetos o clases que no pueden implementarla o no pueden ser marcados con el atributo `<Serializable>`. Por ejemplo, si se desea persistir un objeto que extiende la clase `System.Windows.Forms.Control` se generará una excepción ya que dicha clase no implementa la interfaz `ISerializable`.

Además existen otros dos problemas a resolver. El primero se refiere a la eficiencia. Es el caso que se presenta si existe un objeto "grande" que deba ser serializado, pero solo una pequeña parte de sus estados internos deben ser almacenados. Serializando el objeto completo requerirá mas espacio del necesario, además de que llevaría más tiempo la operación de serializado y recuperación.

El segundo problema está relacionado con la seguridad, cuando se desea proteger algún estado interno del objeto. En otras palabras, cuando es necesario persistir algunos estados, pero otros no.

Los problemas anteriores hacen pensar que la técnica de serialización de .NET no sea adecuada para guardar y recuperar objetos. Pero no es del todo cierto. Aún se puede utilizar dicha técnica con la ayuda del patrón de diseño Memento.

La idea de este patrón de diseño es sencilla. En este patrón el objeto cuyos estados se desean persistir recibe el nombre de creador. Para aplicar el patrón se debe crear otro objeto, llamado memento. El objeto memento es un objeto externo que almacenará el estado del creador. Por lo tanto, si se desea almacenar el valor de las variables `a`, `b` y `c` del creador, el objeto memento tendrá las mismas variables `a`, `b` y `c`. ¿Pero qué sucede si alguna de estas variables son privadas, y

por lo tanto no son accesibles desde fuera del creador? La solución radica en que el creador tiene dos métodos: uno llamado `CreateMemento` y otro llamado `SetMemento`.

El método `CreateMemento` crea un objeto memento, copia todos los estados que deben ser serializados en dicho objeto, y lo retorna. Por lo tanto, para persistir al creador, se realiza la llamada al método `CreateMemento` y se obtiene un objeto memento que será el objeto serializado.

Al deserializar un objeto memento, la llamada al método `SetMemento` del objeto creador permite copiar los estados anteriormente almacenados.

6.7.3.2 DrawElementMemento, DrawConnectorMemento, DrawEntityMemento, EntityAttributeMemento

Estas cuatro clases permiten la creación del objeto memento para los conectores y entidades. Aprovechando que ambos tipos de dibujos tienen propiedades comunes, sus respectivos objetos memento derivan de la clase `DrawElementMemento`. La construcción de esta clase es muy sencilla, y su único objetivo es la de obtener un objeto que almacene solo los estados internos de interés. Por ejemplo, el código de la clase `DrawEntityMemento` es el siguiente:

```
[Serializable]
public class DrawEntityMemento: DrawElementMemento
{
    public Color StartColor;
    public Color EndColor;
    public ArrayList Attributes;
    public ArrayList InLineList;
    public ArrayList OutLineList;
    public Rectangle Rectangle;
    public bool Optional;

    public DrawEntityMemento(DrawElementMemento memento)
        : base(memento)
    {
    }
}
```

Para estas tres clases, sus objetos creadores (según lo visto en el apartado anterior) son los objetos `DrawElement`, `DrawEntity` y `DrawConnector`.

Los objetos de la clase `EntityAttributeMemento` son objetos memento de la clase creadora `EntityAttribute`.

6.7.3.3 MementoManager

El único método que contiene esta clase es `BuildDrawObjectCollection`. Su objetivo es –una vez recuperado de disco un documento- relacionar los distintos objetos de las colecciones recién creadas de forma correcta completando para ello las colecciones `InConnectorList` y `OutConnectorList` de las entidades, y estableciendo el valor de las propiedades `FromRectangle` y `ToRectangle` de los conectores.

6.7.3.4 AttributeEditor, RulesEditor

La clase `AttributeEditor` permite la edición de los atributos de los objetos `DrawEntity`. Gracias a ella, cuando se desea editar los atributos de una entidad desde el panel de propiedades se abre una instancia del formulario `frmEntityAttributes`.

La clase `RulesEditor` permite la edición de las reglas asociadas a los conectores y a los atributos de las entidades a través del formulario `frmConditions`.

6.7.3.5 RuleCollection

Esta colección agrupa elementos de clase `Rule`. El método `GetRuleCollectionString` retorna la representación string de la colección como la concatenación de todas las reglas.

6.7.3.6 EntityAttributeCollection

Colección de atributos para una entidad. Solo permite la selección simple de objetos. El método `Add` permite añadir nuevos elementos a la colección, mientras que la propiedad `Selected` retorna el objeto atributo actualmente seleccionado.

6.7.3.7 DrawElementCollection

Esta colección agrupa todos los elementos (entidades y conectores) del documento. Permite la selección múltiple de objetos. La propiedad `Selection` retorna la colección de todos los elementos actualmente seleccionados. Sobre esta selección operan el resto de métodos: `UnselectAll`, `MoveSelectionToFront`, `MoveSelectionToBack`, etc. Los objetos son almacenados internamente en una lista, por lo que el orden en que aparecen en el área de dibujo puede ser alterado con los métodos anteriores, brindando en la interfaz de usuario de la aplicación botones y opciones de menú para hacerlo.

6.7.3.8 DrawElement

Clase base con las propiedades y métodos comunes a entidades (`DrawEntity`) y conectores (`DrawConnector`). Todos sus métodos son virtuales, por lo que admiten ser redefinidos en las clases derivadas.

6.7.3.9 DrawEntity

Esta clase permite representar las entidades del documento. Sus propiedades más relevantes son:

- `Attributes`: Colección con los atributos asociados a la entidad.
- `InConnectorList` y `OutConnectorList`: Colecciones que almacenan referencias a los objetos de tipo conector que tienen como origen o destino a la entidad, respectivamente.
- `XPath`: Retorna la expresión XPath asociada a esta entidad.

Los métodos `Draw`, `DrawAttributes` y `DrawName` son utilizados para representar la entidad sobre el área de dibujo, mientras que los métodos `BuildInstanceData`, `BuildFormBody`, `BuildCaptions`, `BuildSchema`… son utilizados en la construcción del formulario XForms. Estos últimos aceptan como parámetro un stream de salida donde son volcados los resultados de su ejecución.

6.7.3.10 DrawConnector

Los objetos de la clase `DrawConnector` representan a los conectores del documento. Sus propiedades más relevantes son:

- `FromRectangle`, `ToRectangle`: permiten obtener/establecer respectivamente las entidades origen y destino asociadas al conector.
- `Rules`: Colección con las reglas asociadas al conector.
- Los métodos `Draw` y `DrawRules` permiten representar el conector sobre el área de dibujo.

6.7.3.11 Rule

Una regla puede formar parte de un conector o a una entidad a través de las reglas de validación de sus atributos.

Una regla está compuesta por dos expresiones (`LExpression` y `RExpression`): una ubicada a la izquierda y otra a la derecha; un operador relacional que las vincula, y un operador lógico que la relaciona con otras reglas.

Por ejemplo, en la regla

```
a + 4 > b AND
```

`a + 4` es el valor de `LEXpression`, `b` es el valor de `RExpression`, `RelationalOperatorType.Greater` (`>`) es el valor de `RelationalOp`, y `LogicalOperatorType.And` (`AND`) es el valor de `LogicalOp`.

6.7.3.12 EntityAttribute

Los atributos de las entidades son representados por objetos de esta clase. Un atributo está caracterizado por:

- `Class`: propiedad de tipo `EntityAttributeClass` que determina si el atributo podrá contener valores simples o múltiples.
- `Type`: tipo de datos del atributo: entero, flotante, o cadena.
- `Validation`: Conjunto de reglas que deben cumplirse para que el valor del atributo sea considerado válido. Dichas reglas pueden establecerse en función de cualquier elemento del documento.
- `Value`: Conjunto de valores/valor predefinido para el atributo.
- `Visible`: Indica si el atributo será visible en el formulario generado.
- `FormLabel`: Etiqueta que aparecerá en el formulario generado junto al control que representa al atributo.
- `ErrorMessage`: Mensaje de error que será visible solo si el valor del atributo no es válido. Esto ocurrirá si en un atributo de tipo numérico se introduce una cadena –por ejemplo- o si no se cumple las reglas de validación asociadas al mismo.

Otras propiedades que definen el comportamiento de los atributos en el formulario una vez generado son: `NotEmpty` y `ReadOnly`.

6.7.3.13 FormManager

Esta clase es la encargada de construir el formulario XForms y copiar todos los ficheros de recursos necesarios: hoja de estilo CSS, imágenes, etc.

Cuenta con el método `BuildForm` que acepta como parámetros la colección de dibujos del documento, su nombre y la ruta donde se encuentra almacenado. A partir de esta información se pregunta al usuario por la ubicación de los ficheros generados y se procede a la creación de los mismos. Si se detecta algún tipo de error, se informa al usuario y se cancela el proceso. Un error podría ser el provocado por insertar un atributo en una expresión y luego eliminar dicho atributo de la entidad. Al intentar generar el formulario se validarán todas las expresiones, y al no existir el atributo eliminado que forma parte de la expresión, se producirá el error.

`BuildForm` genera el formulario en varias "partes" a través de las llamadas a los métodos definidos para tal fin en los objetos de clase `DrawEntity`. Finalmente, `BuildForm` integra las secciones generadas en un único documento XForms.

El resto de métodos (`BuildDocumentOptions`, `BuildGenerateFormBody`, `BuildInstructionsFormBody`, `BuildNewFormBody`, etc.) son utilizados para la creación de las distintas secciones que aparecen en el formulario.

7 FORMULARIO

El formulario es obtenido como resultado del proceso de generación implementado en la aplicación Smart Proposal y es el medio por el cual se recoge la información de las ofertas. Por lo tanto, es parte fundamental del sistema de creación de ofertas y representa la interfaz con la que los usuarios (comerciales) habitualmente trabajarán. Su implementación se ha realizado utilizando XForms, y sus distintas particularidades serán examinadas en este capítulo.

7.1 Visión general

Tal como se ha comentado, el formulario es obtenido por medio de la aplicación Smart Proposal a través de la clase `FormManager`. En él, las entidades se transforman en opciones de menú mientras que sus atributos son campos de texto que pueden ser completados por el usuario.

En la parte superior del panel izquierdo hay una barra de herramientas con diversas opciones que permitirán:

- Abrir, guardar y crear un nuevo formulario.
- Generar un documento Word.
- Visualizar un texto de ayuda con instrucciones para completar el formulario.

Los datos de un formulario no podrán ser guardados si no se cumplen todas las reglas de validación definidas en tiempo de diseño. Los campos que contienen errores son marcados en color rojo, enseñándose además el mensaje de alerta definido por el diseñador.

La forma en que se definen y relacionan las entidades y conectores determinan su comportamiento final en el formulario. Si un atributo es marcado como `NotEmpty` y no contiene un valor al momento de rellenar el formulario, el mismo será considerado como inválido y el mensaje de error asociado se hará visible.

Si un atributo es marcado como `ReadOnly` el usuario no podrá modificar su contenido, pero sí podrá ser modificado a través de las reglas definidas durante el diseño del formulario.

Si un atributo es marcado como no visible, el mismo no será mostrado en el formulario. Pero atención! Todas las reglas de validación asociadas al atributo deberán cumplirse para poder guardar el formulario. Al tratarse de un atributo invisible, los mensajes de error de validación asociados también lo serán.

Tal como se ha comentado al inicio del apartado, las entidades se convierten en opciones de menú en el panel lateral izquierdo. Dichas opciones serán visibles si alguno de los conectores que llegan a su correspondiente entidad (ambos definidos en Smart Proposal) cumple todas las condiciones asociadas a él. De otra forma, la opción no será accesible. En la Figura 7-1 se establece que el valor de `atributo1` debe ser mayor a `10` y menor a `20` como condición asociada a la existencia de la `entidad2`. En la Figura 7-2 puede verse el resultado de esta restricción sobre el formulario.

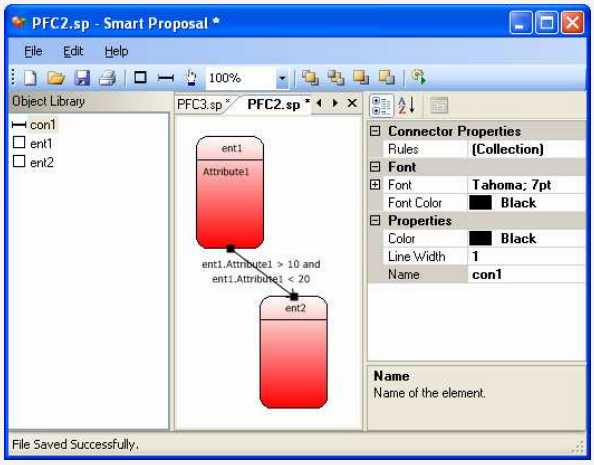


Figura 7-1 Ejemplo de entidad condicionada

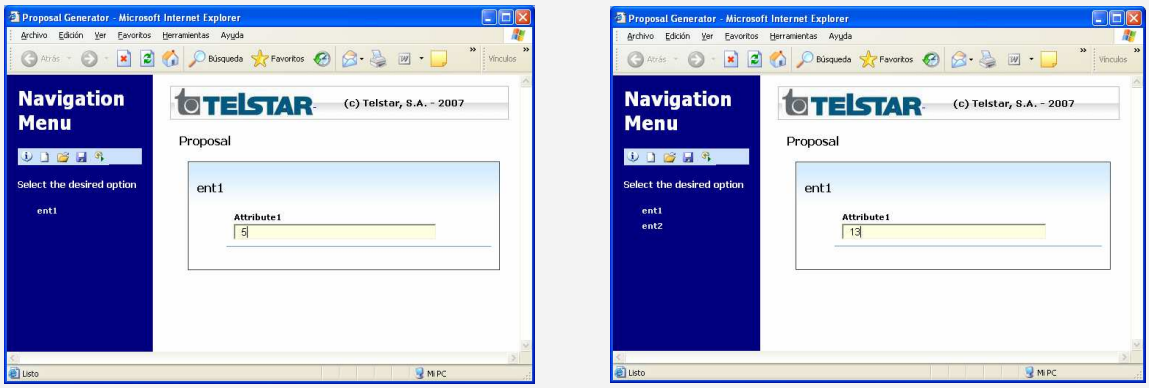


Figura 7-2 Visibilidad de la entidad ent2

Smart Proposal permite la definición de entidades opcionales, es decir, entidades que el usuario puede eliminar y volver a añadir si así lo decide. Añadir o eliminar una entidad afecta a la visibilidad de las entidades que están relacionadas con ella a través de conectores; o a otras entidades que hacen referencia a algún valor de sus atributos. El ejemplo de la Figura 7-3 -tomando como base el de la Figura 7-1- define como opcional la entidad `ent1`:

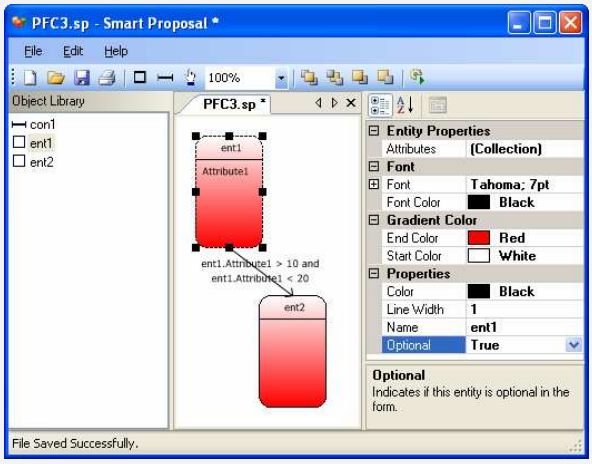


Figura 7-3 Entidad Opcional

En el formulario generado (Figura 7-4) aparece un botón cuyo comportamiento cambia dependiendo si la entidad opcional está o no presente, permitiendo añadir o quitar la entidad. Como la aparición de `ent2` está condicionada a la existencia de la entidad `ent1` (además del valor de uno de sus atributos) el ítem `ent2` desaparece de las opciones disponibles.

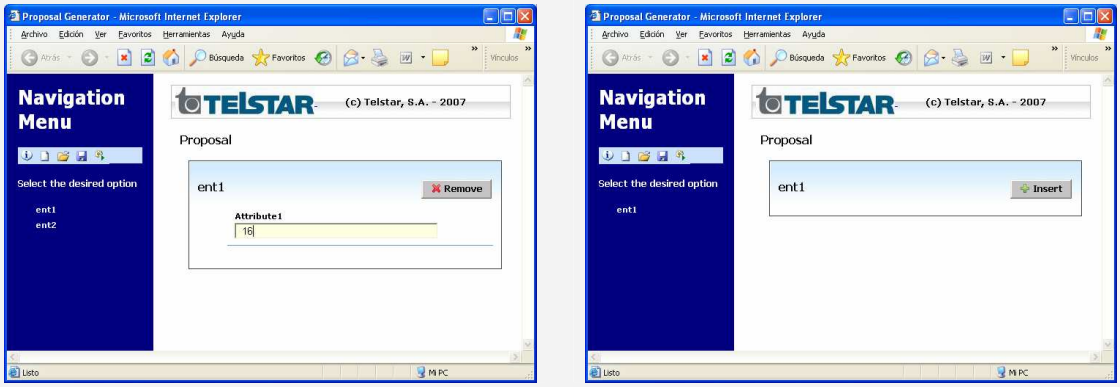


Figura 7-4 Botón Remove-Insert para una entidad opcional

Mientras se diseña el formulario, es posible establecer valores "por defecto" para los atributos. Cuando el mismo es cargado en el navegador web, estos valores son asignados. Lo mismo pasa si una entidad es eliminada y luego añadida, o si una entidad se hace visible al cumplirse alguna de las condiciones de sus conectores asociados.

Al mismo tiempo, pueden crearse dependencias entre valores de atributos. Por ejemplo, al establecer que el valor de `att1` es igual a `att2 + 4`, en el momento que cambia el valor de `att2` debe actualizarse el valor de `att1`. XForms proporciona por defecto –a través del atributo `calculate` de los elementos `bind`- una funcionalidad similar, pero con una característica: un atributo calculado mediante el atributo XForms `calculate` no puede ser modificado.

Este comportamiento no es el deseado para todos los campos de texto, ya que los usuarios deberían poder modificar los valores propuestos por alguno de los cálculos. Es por ello que al cambiar un valor se disparan una serie de eventos cuya misión es la de modificar los valores de los atributos del formulario.

7.2 Implementación

Para la traducción o transformación de las entidades y atributos definidos en la aplicación Smart Proposal a un fichero XML se han utilizado los siguientes criterios:

- El contenedor de la oferta es el elemento raíz `proposal`.
- Dentro del elemento `proposal`, existirán elementos de la forma `nombreEntidad` por cada entidad definida
- Cada entidad contendrá la estructura de atributos duplicado dentro de un elemento de la forma `sp-atts_nombreEntidad`
- Los atributos están representados por elementos de la forma `nombreEntidad_nombreAtributo`

Por ejemplo, la estructura del documento XML para el documento de la Figura 7-3 es el siguiente:

```
<proposal>

  <ent1>
    <sp-atts_ent1>
        <ent1_Attribute1 />
    </sp-atts_ent1>
    <sp-atts_ent1>
        <ent1_Attribute1 />
    </sp-atts_ent1>
  </ent1>
```

```
  <ent2>
    <sp-atts_ent2 />
    <sp-atts_ent2 />
  </ent2>

</proposal>
```

XForms proporciona instrucciones para eliminar/insertar nodos en el documento en respuesta a las acciones del usuario, además de brindar la posibilidad de cargar-recibir/almacenar-enviar los datos editados a un fichero en disco o un servicio web. XForms permite editar de manera sencilla el contenido de documentos XML.

El motivo por el cual se almacena un duplicado de la estructura de las entidades es debido a una limitación de implementación: el estándar XForms especifica que la inserción en un nodeset (conjunto de nodos, como por ejemplo `<ent1>`) debe realizarse **copiando** los valores del último de sus elementos (siguiendo con el ejemplo anterior, `<sp-atts_ent1>`). Es aquí donde se presentan el problema: ¿Qué pasa si no existe ningún elemento dentro del nodeset? El resultado es que no se realiza ninguna inserción, ya que no existen valores "patrones" o "modelos" para copiar y crear el nuevo elemento.

Para solucionar este inconveniente, siempre existirá un elemento dentro de los elementos entidad que funcione a modo de patrón, pero que nunca es visible para el usuario. En la versión 1.1 es posible insertar un elemento utilizando un nuevo modificador en la instrucción `insert`, el cual especifica la ubicación de un patrón modelo. En otras palabras, permite tener un documento XML cuya función es la de ser una plantilla para otros documentos.

Los datos que el usuario manipula en el formulario son almacenados en la segunda repetición de los datos de la entidad, y son los únicos que se tienen en cuenta para todos los procesos.

7.2.1 Etiquetas

Todas las etiquetas del formulario están definidas en un fichero auxiliar llamado `captions.xml`. La información definida en él se estructura en parejas de tipo nombre-valor:

```
<captions>
  <caption>
    <name>sp-title</name>
    <label>Proposal</label>
  </caption>
    <name>sp-instructions</name>
    <label>Instructions</label>
  </caption>
  <caption>
    <name>entidad1</name>
    <label>entidad1</label>
  </caption>
  <caption>
    <name>entidad1_atributo1</name>
    <label>Valor 1:</label>
  </caption>
  ...
</captions>
```

En el formulario XForms existe un modelo específico de datos que contiene la instancia de datos de las etiquetas, y al cual se hace referencia a lo largo del documento.

```
<xf:model id="sp-caption_model">
  <xf:instance id="sp-captions" src=".\src\PFC_captions.xml" xmlns="" />
```

```
  <xf:bind id="sp-bind-caption-title" nodeset="caption[name= 'sp-title']/label" />
  <xf:bind id="sp-bind-caption-save" nodeset="caption[name= 'sp-save']/label" />
  <xf:bind id="sp-bind-caption-open" nodeset="caption[name= 'sp-open']/label" />
  <xf:bind id="sp-bind-caption-new" nodeset="caption[name= 'sp-new']/label" />
  <xf:bind id="sp-bind-caption-generate" nodeset="caption[name= 'sp-generate']/label" />
  <xf:instance id="sp-instructions" src=".\src\PFC_instructions.xml" xmlns="" />
  <xf:bind id="sp-bind-instructions-tittle" nodeset="instance('sp-instructions')/tittle"
/>
  <xf:bind id="sp-bind-instructions-line" nodeset="instance('sp-instructions')/line" />
  <xf:bind id="sp-bind-caption-entidad1" nodeset="caption[name= 'entidad1']/label" />
  <xf:bind id="sp-bind-caption-entidad1_atributo1" nodeset="caption[name=
'entidad1_atributo1']/label" />
  <xf:bind id="sp-bind-caption-entidad1_atributo2" nodeset="caption[name=
'entidad1_atributo2']/label" />
   ...
</xf:model>
```

La idea al separar las etiquetas del fichero XForms era la de implementar un editor gráfico para que el diseñador pudiese definir de manera simple la distribución, el aspecto y el texto que acompaña a los controles. Este hecho afectaría directamente al contenido del fichero `captions.xml` y el de la hoja de estilos CSS del formulario.

Mientras dicho editor no sea desarrollado, el aspecto del documento XForms debe ser tratado editando directamente la hoja de estilos, mientras que las etiquetas de los controles debe ser definido a través de la herramienta Smart Proposal.

El contenido de las instrucciones brindadas al usuario al pulsar el correspondiente botón de la barra de herramientas puede ser modificado por el diseñador editando el fichero `instructions.xml`. Añadiendo, eliminando o cambiando el valor de los elementos `line` se logra el efecto deseado.

```
<instructions>
  <tittle>Instructions</tittle>
  <line>All the required values of the form are marked with an asterisk in the lavel.
Please, fill all of them or you will not be able to save the form.</line>
  <line>If a field contains an invalid value, an error message will be displayed.</line>
  <line>You can open a filled form pressing the "Open" button in the toolbar.</line>
</instructions>
```

7.2.2 Hoja de estilos

La hoja de estilos tiene toda la responsabilidad a la hora de definir la distribución de los controles y su aspecto en los formularios HTML tradicionales. En los formularios XForms la hoja de estilos sigue jugando el mismo papel.

A la hora de hacer referencia a un tipo de control cada implementación del estándar brinda sus propias “pseudos clases”. Por ejemplo, la extensión de Mozilla Firefox utiliza la siguiente sintaxis para referirse al contenido de un cuadro de texto, o para hacer referencia a un control de entrada que sea de solo lectura:

```
.xf-value
{
        width: 70%;
}
```

```
xf|input:-moz-read-only .xf-value
{
        background-color: cyan;
}
```

Mientras que en formsPlayer la sintaxis es la siguiente:

```
.pe-value
{
        width: 70%;
}
```

```
 .pc-read-only .input-value
 {
         background-color: cyan;
 }
```

En el formulario, la aparición del mensaje de error asociado a un control depende de si el mismo contiene un valor inválido. Este aspecto también se controla desde la hoja de estilos de la siguiente forma (se presenta la sintaxis para formsPlayer y para Firefox)

```
.pc-valid xf\:alert                    *:valid   > xforms|alert
{                                      {
        display: none;                         display: none;
}                                      }

.pc-invalid  xf\:alert                 *:invalid  > xforms|alert
{                                      {
        display: inline;                       display: inline;
}                                      }
```

7.2.3 Presentación opciones

La presentación de opciones en la pantalla se lleva a cabo mediante el uso de los elementos `switch-case` y `toggle` de XForms. Gracias a ellos, el contenido asociado a las distintas opciones del formulario aparece en su parte central de forma excluyente. Es decir, no puede ser visible en pantalla y de forma simultánea el contenido de dos o más opciones.

El uso de estos elementos en el formulario es el siguiente:

- Se definen los controles que serán visibles en el formulario dentro de una estructura `switch-case`

```
<xf:switch class="form">
     <xf:case id="sp-case-instructions" class="internal-box">
         ...
     </xf:case>

     <xf:case id="sp-case-save" class="internal-box">
         ...
     </xf:case>

     <xf:case id="sp-case-entidad1" class="internal-box">
         <xf:output bind="sp-bind-caption-entidad1" model="sp-caption_model" />
          ...
     </xf:case>
</xf:switch>
```

- El código asociado al evento clic (`DOMActivate`) de los botones (es decir, de las opciones) establece que parte del formulario será visible utilizando la acción `toggle`

```
       <xf:trigger appearance="minimal" bind="sp-bind-data-instructions">
       <xf:action ev:event="DOMActivate">
         <xf:toggle case="sp-case-instructions" />
       </xf:action>
         ...
     </xf:trigger>

    <xf:trigger appearance="minimal" bind="entidad1_aux">
         <xf:label bind="sp-bind-caption-entidad1" model="sp-caption_model" />
         <xf:action ev:event="DOMActivate">
           <xf:toggle case="sp-case-entidad1" />
         </xf:action>
   </xf:trigger>
```

- El atributo `appearance` determina el aspecto inicial del botón. Un valor de `minimal` indica que el botón debe enseñarse como un link

7.2.4 Abrir y guardar formularios

XForms permite definir como y bajo que condiciones es posible abrir o guardar un fichero. Por ejemplo, puede establecerse que un formulario no pueda ser guardado hasta que todos sus elementos sean válidos, o que el fichero sea almacenado en disco o procesado por un servicio web.

Para ello el formulario XForms utiliza el elemento `submission`:

```
<xf:submission id="sp-save-local-file" method="put" indent="true" replace="none">
    <xf:resource value="instance('sp-internalVars')/saveFilePath" />
</xf:submission>

<xf:submission id="sp-open-local-file" method="get" replace="instance" instance="sp-
data" validate="false">
    <xf:resource value="instance('sp-internalVars')/openFilePath" />
</xf:submission>
```

Para guardar un formulario (elemento `submission` `sp-save-local-file`) se especifica el tipo de acción mediante el atributo `method`, y si la salida será tabulada mediante el atributo `ident` (aspecto puramente estético, que no afecta el funcionamiento del resto de módulos). El atributo `replace` indica el comportamiento del formulario una vez que se han enviado los datos. Por defecto el resultado devuelto por el servidor sustituye al documento entero, pero en este caso, y dado que el fichero es almacenado en disco, el valor de `none` indica que se deja el documento que contiene el formulario tal como está, sin ser sustituido. La ruta donde será almacenado el fichero se obtiene del elemento `saveFilePath` de la instancia de variables internas del documento.

Cuado se desea abrir un formulario hay que utilizar también el elemento `submission`. En este caso su atributo `id` tiene el valor `sp-open-local-file`. Como el objetivo es abrir un fichero debe especificarse el método `get` en el atributo `method` y que instancia de datos debe ser reemplazada (atributos `replace` e `instance`). Adicionalmente se indica si debe ser validado el contenido del fichero abierto. La ruta del fichero es obtenida a través del valor de la variable interna `openFilePath`.

Ambos elementos `submission` son activados por medio del elemento `send` como respuesta a la pulsación de sus correspondientes botones.

```
<xf:trigger>
   <xf:label>
     <img src="./src/sp-opendoc.gif" />
     Open File
   </xf:label>
   <xf:action ev:event="DOMActivate">
     <xf:send submission="sp-open-local-file"
/>
   </xf:action>
</xf:trigger>
```

```
<xf:trigger>
   <xf:label>
     <img src="./src/sp-Save.gif" />
     Save File
   </xf:label>
   <xf:action ev:event="DOMActivate">
     <xf:send submission="sp-save-local-file"
/>
   </xf:action>
</xf:trigger>
```

¿Qué sucede cuando ocurren errores durante la carga o la apertura de un formulario? Se le comunica al usuario mediante un mensaje. Las implementaciones XForms no proporcionan a día de hoy métodos que permitan obtener información de contexto sobre que condición de error se ha presentado y en que elemento, por lo que la información brindada al usuario es un tanto genérica.

Para indicar en todo momento el resultado de la operación se utilizan dos variables internas que almacenan el texto del mensaje visualizado por el usuario.

```
<xf:bind id="sp-bind-data-save-text" nodeset="instance('sp-internalVars')/save-text" />
```

```
<xf:bind id="sp-bind-data-open-text" nodeset="instance('sp-internalVars')/open-text" />
```

Estas variables son asociadas a dos controles de tipo output para presentar su contenido.

```
<xf:output bind="sp-bind-data-save-text" />
<xf:output bind="sp-bind-data-open-text" />
```

Sus valores son establecidos capturando el evento `xforms-submit-error` que es disparado si ocurre un error, y `xforms-submit-done` si la operación se realiza sin errores.

```
<xf:setvalue ev:event="xforms-submit-error" ev:observer="sp-open-local-file" bind="sp-bind-data-open-text">Error opening the form. Please, check the path and be sure you have read rights over the file.</xf:setvalue>
<xf:setvalue ev:event="xforms-submit-done" ev:observer="sp-open-local-file" bind="sp-bind-data-open-text">File opened without errors.</xf:setvalue>
<xf:setvalue ev:event="xforms-submit-error" ev:observer="sp-save-local-file" bind="sp-bind-data-save-text">Error submitting form. Please, fill all required fields (marked with a red line), and correct all fields with errors (marked in red colour).</xf:setvalue>
<xf:setvalue ev:event="xforms-submit-done" ev:observer="sp-save-local-file" bind="sp-bind-data-save-text">Submission done without errors!</xf:setvalue>
```

7.2.5 Atributos de valores múltiples

Los atributos de valores múltiples en el formulario son representados por cuadros de selección desplegables. Para implementarlos es necesario definir para todos sus valores la instancia del documento a la cuál están asociados. Por ejemplo, si la `entidad1` contiene un atributo llamado `atributo2` de valores múltiples con dos posibles opciones, su construcción en el formulario seguirá la siguiente secuencia:

1. En la instancia de datos `sp-ddw_temporal` definida dentro del modelo principal se crea un elemento `entidad1_atributo2` que contiene dos elementos llamados `value`.

```
<xf:instance id="sp-ddw_temporal" xmlns="" xmlns:xf="http://www.w3.org/2002/xforms">
    <root>
        <entidad1_atributo2>
            <value />
            <value />
        </entidad1_atributo2>
          ...
    </root>
</xf:instance>
```

2. A continuación se enlazan con instrucciones `bind` cada uno de los elementos `value` para el `atributo2`, con el objetivo de establecer sus valores. Además se crea un enlace con el elemento que almacenará el valor de la selección realizada por el usuario.

```
<xf:bind id="sp-bind-mv-entidad1_atributo21" nodeset="instance('sp-ddw_temporal')/entidad1_atributo2/value[position()=1]" calculate="cadena1" />
<xf:bind id="sp-bind-mv-entidad1_atributo22" nodeset="instance('sp-ddw_temporal')/entidad1_atributo2/value[position()=2]" calculate="cadena2" />

<xf:bind id="sp-bind-data-entidad1_atributo2" nodeset="instance('sp-data')/entidad1/sp-atts_entidad1[position()=2]/entidad1_atributo2" required="true()" />
```

3. Para terminar es necesario crear el control y definir su contenido. Para conseguir el primer objetivo se utiliza la instrucción `select1`, mientras que para conseguir el segundo se utiliza la instrucción `itemset`. Con `itemset` debe especificarse el valor de la opción y el texto asociado que será visualizado. En el caso de los formularios Smart Proposal, ambos valores coinciden.

```
<xf:select1 bind="sp-bind-data-entidad1_atributo2">
    <xf:label>Atributo 2:</xf:label>
    <xf:itemset nodeset="instance('sp-ddw_temporal')/entidad1_atributo2/value">
        <xf:label ref="text()" />
        <xf:value ref="text()" />
    </xf:itemset>
</xf:select1>
```

8 PROPOSAL GENERATOR

Una vez un formulario es completado y sus datos guardados en un fichero, el siguiente paso es generar la oferta que debe entregarse al cliente. Esta tarea la lleva a cabo este control ActiveX (Figura 8-1) insertado en el formulario, al cual se accede a través de su barra de herramientas haciendo clic en el botón "Generate".

Para crear el documento final, Proposal Generator integra la información recolectada por el formulario en un documento Word especialmente marcado. Los detalles de implementación de este control son tratados en el presente capítulo.

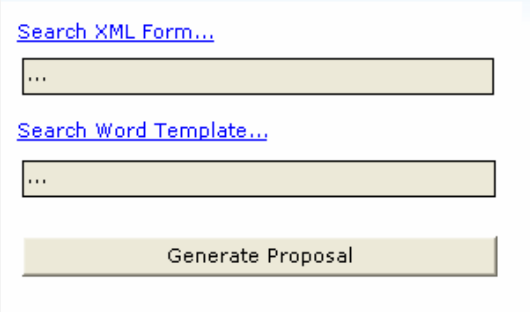


Figura 8-1 Interfaz Proposal Generator

8.1 Clases

Este control ActiveX se encuentra definido en el espacio de nombres `MergeCustomControl`, y su diagrama de clases -donde se enseñan los métodos y propiedades más importantes- está representado en la Figura 8-2.

8.2 Espacio de nombres MergeCustomControl

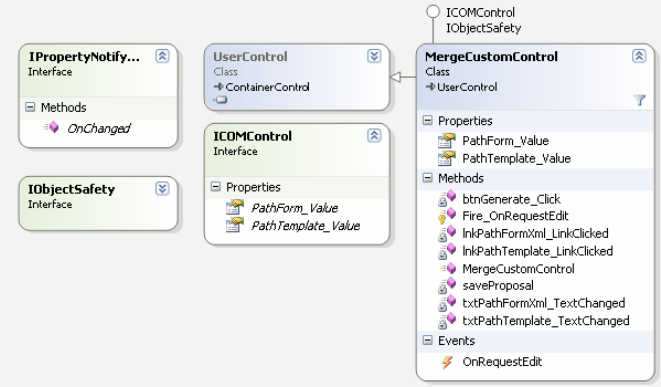


Figura 8-2 Diagrama de clases MergeCustomControl

La operación realizada por el control -de forma resumida- consiste en aplicar una transformación XSLT sobre el documento XML entrada para generar un nuevo documento XML de salida. Este último al ser convertido a formato DOC -utilizando para ello la API de Microsoft Word- representa la oferta final que podrá ser entregada al cliente. El proceso detallado de toda esta operación es descrito en el siguiente apartado.

NOTA: Antes de continuar hay que recordar que un diseñador debe "marcar" previamente un documento Word en aquellas posiciones donde deba reemplazarse texto (etiquetas `replaceText`) o deba aparecer/eliminarse una sección (etiquetas `marker`) utilizando la extensión Smart Document. Este documento debe ser accesible por el usuario ya que es el fichero que funcionará como plantilla para combinar la información del formulario.

8.2.1 btnGenerate_Click

La funcionalidad del control es gestionada por este método, que se ejecuta como resultado a la pulsación del botón "Generate Proposal".

Desde `btnGenerate_Click` se realizan las siguientes acciones:

- Se crea una carpeta temporal donde se copia:
  - El formulario XForms -desde la ubicación especificada por el usuario en el control-, con el nombre "Proposal1.xml"
  - El fichero de transformación XSLT, con el nombre "Transform.xslt".
  - Del documento-plantilla indicado por el usuario se extrae la ruta del fichero XSD utilizado para introducir las marcas. Dicho archivo es copiado a la carpeta temporal con el nombre "schema.xsd".
- Se aplica la transformación XSTL a partir de las reglas definidas en el fichero Transform.xslt, utilizando como entrada el documento-plantilla. La salida es generada en la carpeta temporal y recibe el nombre "Output.xml".
- El fichero Output.xml es convertido a formato DOC.

El contenido y funcionalidad del fichero de transformación XSLT es explicado de forma detallada en el apartado 8.3.

8.2.2 saveProposal

Una vez aplicada la transformación sobre el fichero de entrada, el resultado es un fichero WordML llamado Output.xml. Este fichero es un documento Word "en toda regla" que puede ser abierto y editado de forma normal. Pero a pesar de ello, uno de los requisitos del proyecto es que el formato del fichero generado sea DOC, por lo que queda realizar la conversión.

Para ello se utiliza la API de Word: se abre el documento XML generado, y se guarda en formato DOC:

```
Word._Document wDoc = WordApp.Documents.Open(ref oNameOpen, ...)
object oFileFormat = Word.WdSaveFormat.wdFormatDocument
wDoc.SaveAs(ref oNameSave, ref oFileFormat, ...)
wDoc.Close(...)
WordApp.Quit(...)
```

La ubicación donde el fichero es guardado corresponde a la indicada por el usuario por medio de un cuadro de diálogo.

8.2.3 txtPathFormXml_TextChanged, txtPathTemplate_TextChanged, lnkPathFormXml_LinkClicked, lnkPathTemplate_LinkClicked

Los dos primeros métodos son ejecutados como respuesta al cambio en el texto almacenado por los cuadros de texto. Este cambio puede ser generado desde el mismo control ActiveX (si el usuario pulsa sobre alguna de las etiquetas para especificar la ruta del fichero) o desde el exterior. Esta última opción es posible ya que MergeCustomControl se trata de un control ActiveX que implementa las interfaces necesarias para ser incrustado –por ejemplo- en una página HTML y hacer visible/editable los valores de ambos cuadros de texto:

```
<object id="sp-generator" classid="clsid:1FEE489F-A555-4408-8FBF-3F69F8C57A43"
standby="Loading..." width="297" height="261">
                  <param name="PathForm_Value" Value="…" />
                  <param name="PathTemplate_Value" Value="…" />
</object>
```

Los métodos `lnkPathFormXml_LinkClicked`, `lnkPathTemplate_LinkClicked` son ejecutados cuando se pulsa sobre alguna de las etiquetas del control, y permiten seleccionar la ubicación de los ficheros de formulario y plantilla respectivamente.

8.3 Transform.xslt

El fichero de transformación representa la columna vertebral de todo el proceso, ya que en él se definen las reglas (también llamadas `templates`) aplicadas sobre el documento de entrada (en este caso en particular, un documento Word en formato XML). La forma en la que son definidas es a través de un lenguaje declarativo: definiendo el conjunto de datos con los que se trabajarán, y especificando en su cuerpo como serán transformados. Para más información del formato XML de Word y su integración en Office, consultar el capítulo 14.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
  xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:w10="urn:schemas-microsoft-com:office:word"
  xmlns:sl="http://schemas.microsoft.com/schemaLibrary/2003/core"
  xmlns:aml="http://schemas.microsoft.com/aml/2001/core"
  xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:configNS="http://www.telstar.eu/configurador"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  >

  <xsl:output method="xml" indent="yes" />

  <!-- Definimos los caracteres para millares y decimales -->
  <xsl:decimal-format name="spain" decimal-separator="," grouping-separator="."/>

  <xsl:variable name="configFile" select="document('Proposal1.xml')"/>
  <xsl:variable name="xsdFile" select="document('schema.xsd')"/>

  <!--Regla Identidad por defecto: copiar la entrada en la salida -->
  <xsl:template match="@*|node( )">
    <xsl:copy>
      <xsl:apply-templates select="@*|node( )"/>
    </xsl:copy>
  </xsl:template>

  <!-- Excepciones a la regla Identidad -->

  <xsl:template match="sl:schemaLibrary">
  </xsl:template>

  <xsl:template match="o:CustomDocumentProperties">
  </xsl:template>

  <!-- Copiamos en la salida el contenido de las partes del documento (marker's)
definidos en el fichero de configuracion -->
  <xsl:template match="configNS:marker">
    <xsl:variable name="bName" select="./@markerName"/>
    <xsl:variable name="longName" select="substring-before($bName, '|')"/>
    <xsl:variable name="shortName" select="substring-after($bName, '|')"/>


    <xsl:variable name="configNode" select="$configFile//*[local-name() = concat('sp-
```

```
atts_',$longName)]"/>
    <xsl:if test="count($configNode) = 2">
      <xsl:apply-templates/>
    </xsl:if>


  </xsl:template>

  <!--  Para el nodo del espacio de nombres del configurador, eliminarlo del documento o
reemplazar
        los valores contenidos en él   -->
  <xsl:template match="configNS:document">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="configNS:replaceText">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="configNS:replaceText//w:t">
    <xsl:variable name="attReplaceText"
select="ancestor::configNS:replaceText/@replaceTextName"/>
    <xsl:variable name="longName" select="substring-before($attReplaceText, '|')"/>
    <xsl:variable name="shortName" select="substring-after($attReplaceText, '|')"/>
    <xsl:variable name="allConfigNode" select="$configFile//*[local-name() =
$longName]"/>
    <xsl:variable name="configNode" select="$allConfigNode[position() = 2]"/>


    <xsl:text disable-output-escaping="yes">&lt;w:t&gt;</xsl:text>

    <!--  Buscar el tipo de este atributo en el fichero de esquema (XSD) para
          formatear correctamente en caso de números FLOAT / INTEGER
    -->
    <xsl:variable name="typeReplaceText" select="$xsdFile//xsd:element[@name =
$longName]/@type"/>

    <xsl:choose>
      <xsl:when test="string-length(normalize-space($configNode/text())) = 0">
        <xsl:value-of select="$configNode/text()"/>
      </xsl:when>
      <xsl:when test="contains($typeReplaceText, 'float') = true()">
        <xsl:value-of select="format-number($configNode/text(),
'###.###.###.###.###,00', 'spain')"/>
      </xsl:when>
      <xsl:when test="contains($typeReplaceText, 'integer') = true()">
        <xsl:value-of select="format-number($configNode/text(), '###.###.###.###.###',
'spain')"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$configNode/text()"/>
      </xsl:otherwise>
    </xsl:choose>

    <xsl:text disable-output-escaping="yes">&lt;/w:t&gt;</xsl:text>
  </xsl:template>
```

Al comienzo se definen los distintos espacios de nombres que puede contener un documento WordML, además del espacio de nombres utilizado para identificar las marcas introducidas en el documento: `configNS="http://www.telstar.eu/configurador"`

Luego de definir el tipo de salida y los separadores decimales y de millares que se utilizarán para representar los números, se abre el fichero que almacena los datos introducidos por el

usuario en el formulario (Proposal1.xml), y el fichero con la estructura de entidades/atributos (schema.xsd)

```
<xsl:variable name="configFile" select="document('Proposal1.xml')"/>
<xsl:variable name="xsdFile" select="document('schema.xsd')"/>
```

A continuación se define la llamada "regla identidad":

```
<xsl:template match="@*|node( )">
  <xsl:copy>
    <xsl:apply-templates select="@*|node( )"/>
  </xsl:copy>
</xsl:template>
```

Esta regla establece que a cualquier atributo (indicado por la cadena "@*") o nodo (es decir, instrucción de procesamiento, comentario, elemento…) del documento de entrada le será aplicada una regla, y el resultado de su aplicación formará parte del fichero de salida.

Pero según se ha comentado, el lenguaje de marcas WordML define cientos de elementos distintos para obtener toda la funcionalidad de un documento Word tradicional sobre un fichero XML. ¿Significa eso que hay que definir tantas reglas como elementos existan? La respuesta es no, ya que solo se definen reglas para aquellos elementos que se desean procesar. El resto de elementos –al no tener una regla definida- son copiados a la salida sin ningún cambio a través de la regla identidad.

Las primeras dos excepciones a la regla identidad son:

```
<xsl:template match="sl:schemaLibrary">
</xsl:template>

<xsl:template match="o:CustomDocumentProperties">
</xsl:template>
```

El objetivo de las reglas vacías es la de no copiar en la salida el contenido de las marcas `sl:schemaLibrary` ni `o:CustomDocumentProperties` ya que en ellas se encuentra alojada información sobre la personalización Smart Document que no es necesaria para abrir el documento generado.

Con la siguiente regla –y gracias al uso de la instrucción `xsl:apply-templates`- se procesan los elementos contenidos por la etiqueta `configNS:document` de forma recursiva. Recordar que esta marca contiene el resto de elementos del espacio de nombres `configNS: marker` y `replaceText`.

```
<xsl:template match="configNS:document">
  <xsl:apply-templates/>
</xsl:template>
```

Solo queda definir las reglas para los elementos `configNS:marker` y `configNS:replaceText`

```
<xsl:template match="configNS:marker">
  <xsl:variable name="bName" select="./@markerName"/>
  <xsl:variable name="longName" select="substring-before($bName, '|')"/>
  <xsl:variable name="shortName" select="substring-after($bName, '|')"/>
```

```
    <xsl:variable name="configNode" select="$configFile//*[local-name() = concat('sp-
atts_',$longName)]"/>
    <xsl:if test="count($configNode) = 2">
      <xsl:apply-templates/>
    </xsl:if>
  </xsl:template>
```

La regla anterior será aplicada a todos los nodos del espacio de nombres `configNS` llamados `marker`. En ella se obtiene el contenido del atributo `markerName`, el cual es dividido en dos subcadenas por medio de las instrucciones `substring-before` y `substring-after`. A continuación se seleccionan en la variable `configNode` todos los nodos del fichero Proposal1.xml cuyo nombre sea igual a la expresión "`sp-atts_nombreentidad`". Si existen dos nodos con dicho nombre, se siguen aplicando las reglas de transformación al contenido del nodo procesado (`apply-templates`). En caso contrario, significa que la entidad en la oferta no fue incluida por lo que el contenido asociado a ella en el documento Word debe ser eliminado, es decir, no procesado por la transformación.

La siguiente regla es similar a la aplicada sobre los elementos `configNS:document`, por lo que no se entrará en detalle.

```
  <xsl:template match="configNS:replaceText">
    <xsl:apply-templates/>
  </xsl:template>
```

Finalmente, la regla más extensa en la correspondiente a los elementos `w:t` que estén contenidos en un nodo `configNS:replaceText`:

```
<xsl:template match="configNS:replaceText//w:t">
    <xsl:variable name="attReplaceText"
select="ancestor::configNS:replaceText/@replaceTextName"/>
    <xsl:variable name="longName" select="substring-before($attReplaceText, '|')"/>
    <xsl:variable name="shortName" select="substring-after($attReplaceText, '|')"/>
    <xsl:variable name="allConfigNode" select="$configFile//*[local-name() =
$longName]"/>
    <xsl:variable name="configNode" select="$allConfigNode[position() = 2]"/>

    <xsl:text disable-output-escaping="yes">&lt;w:t&gt;</xsl:text>

    <!--  Buscar el tipo de este atributo en el fichero de esquema (XSD) para
          formatear correctamente en caso de números FLOAT / INTEGER
    -->
    <xsl:variable name="typeReplaceText" select="$xsdFile//xsd:element[@name =
$longName]/@type"/>

    <xsl:choose>
      <xsl:when test="string-length(normalize-space($configNode/text())) = 0">
        <xsl:value-of select="$configNode/text()"/>
      </xsl:when>
      <xsl:when test="contains($typeReplaceText, 'float') = true()">
        <xsl:value-of select="format-number($configNode/text(),
'###.###.###.###.###,00', 'spain')"/>
      </xsl:when>
      <xsl:when test="contains($typeReplaceText, 'integer') = true()">
        <xsl:value-of select="format-number($configNode/text(), '###.###.###.###.###',
'spain')"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$configNode/text()"/>
      </xsl:otherwise>
```

```
    </xsl:choose>

    <xsl:text disable-output-escaping="yes">&lt;/w:t&gt;</xsl:text>
  </xsl:template>
```

Recordar que los elementos `w:t` almacenan en los documento WordML el texto introducido por el usuario. Por lo tanto es necesario eliminar su contenido y reemplazarlo por el contenido proveniente del formulario, siempre y cuando se encuentren dentro de una marca `configNS:replaceText` (esta condición la expresa la expresión `configNS:replaceText//w:t`)

Para ello:

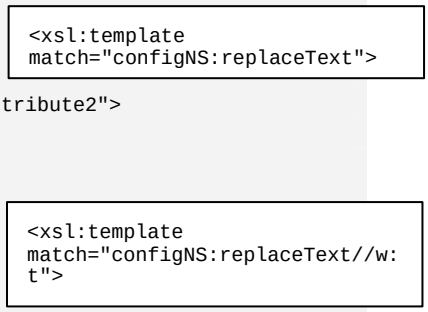
- Se obtiene el nombre del elemento `replaceText` a partir del atributo `replaceTextName` y el uso de las funciones `string-before`/`string-after`
- En la variable `allConfigNode` se seleccionan todos los nodos del documento Proposal1.xml cuyos nombres coincidan con el valor anterior. De acuerdo al diseño de instancia de datos adoptado, solo podrán existir uno o dos nodos.
- Del resultado de la anterior se selecciona el segundo nodo en la variable `configNode`. En caso de existir un único nodo, el valor de `configNode` será vacío.
- Una vez obtenido el valor por el cual reemplazar el contenido del elemento `w:t` es necesario formatearlo de forma correcta dependiendo del tipo de dato que se trate. Esta información se obtiene del fichero schema.xsd, más específicamente del atributo `type` del nodo `xsd:element` cuyo atributo `name` coincida con el nombre del elemento `replaceText`. El formateo de las cadenas se realiza utilizando la función `format-number`.

Es importante hacer notar que el documento final generado no incluirá ninguna marca del espacio de nombres `configNS` (`document`, `marker` y `replaceText`), ya que dichas etiquetas son procesadas y eliminadas.

A continuación se presenta una porción de un documento Word XML, indicando el contenido de que elementos son procesados y a través de que reglas:

```
<ns0:document>
        <w:p/>
                <w:r> <w:t>Oferta</w:t> </w:r>
        <w:p/>
        <ns0:replaceText replaceTextName="ent3_Attribute1|Attribute1
                <w:p>
                        <w:pPr>
                                <w:pStyle w:val="Nombreempresa"/>
                                <w:rPr>
                                        <w:lang w:val="ES-TRAD"/>
                                </w:rPr>
                        </w:pPr>
                        <w:r>
                                <w:rPr>
                                        <w:lang w:val="ES-TRAD"/>
                                </w:rPr>
                                <w:t>NOMBRE empresa</w:t>
                        </w:r>
                </w:p>
        </ns0:replaceText>
  <w:p>
    <w:pPr>
      <w:pStyle w:val="Direccin"/>
      <w:rPr>
        <w:lang w:val="ES-TRAD"/>
      </w:rPr>
    </w:pPr>
  </w:p>
  <ns0:marker markerName="ent2|ent2">
```

```
    <w:proofErr w:type="spellStart"/>
    <w:p>
      <w:pPr>
        <w:pStyle w:val="Modeloportada"/>
        <w:rPr>
          <w:lang w:val="ES-TRAD"/>
        </w:rPr>
      </w:pPr>
      <w:r>
        <w:rPr>
          <w:lang w:val="ES-TRAD"/>
        </w:rPr>
        <w:t>SteriMega</w:t>
      </w:r>
      <w:proofErr w:type="spellEnd"/>
      <w:r>
        <w:rPr>
          <w:lang w:val="ES-TRAD"/>
        </w:rPr>
        <w:t/>
      </w:r>
      <ns0:replaceText replaceTextName="ent3_Attribute2|Attribute2">
        <w:r>
          <w:rPr>
            <w:lang w:val="ES-TRAD"/>
          </w:rPr>
          <w:t>modelo</w:t>
        </w:r>
      </ns0:replaceText>
    </w:p>
  </ns0:marker>
</ns0:document>
```



9 SMART DOCUMENT

Smart Document es una extensión para Microsoft Word desarrollada en C#. Su finalidad es introducir marcas XML en un documento Word que cumplirá el papel de plantilla en la generación de ofertas.

Las marcas introducidas están vinculadas a las entidades y atributos creados con Smart Proposal de la siguiente manera:

- `Document`: Elemento XML que engloba el resto de marcas del documento. Se trata de una etiqueta raíz que cumple la función de contenedor.
- `Marker`: Una entidad tiene su equivalencia con una etiqueta de tipo `marker` en el documento Word.
- `ReplaceText`: Los atributos de las entidades se representan como marcas de este tipo en los documentos Word.

La inserción de marcas en el fichero Word debe cumplir las siguientes restricciones:

- Una etiqueta de tipo `marker` o `replaceText` debe estar incluida dentro de una etiqueta `document`.
- Dentro de una etiqueta `marker` solo puede existir etiquetas de tipo `marker` o `replaceText`.
- Una etiqueta `replaceText` no puede incluir ninguna otra etiqueta.

Si alguna de estas reglas no se cumple, Microsoft Word lo indicará con un subrayado de color púrpura (Figura 9-1)

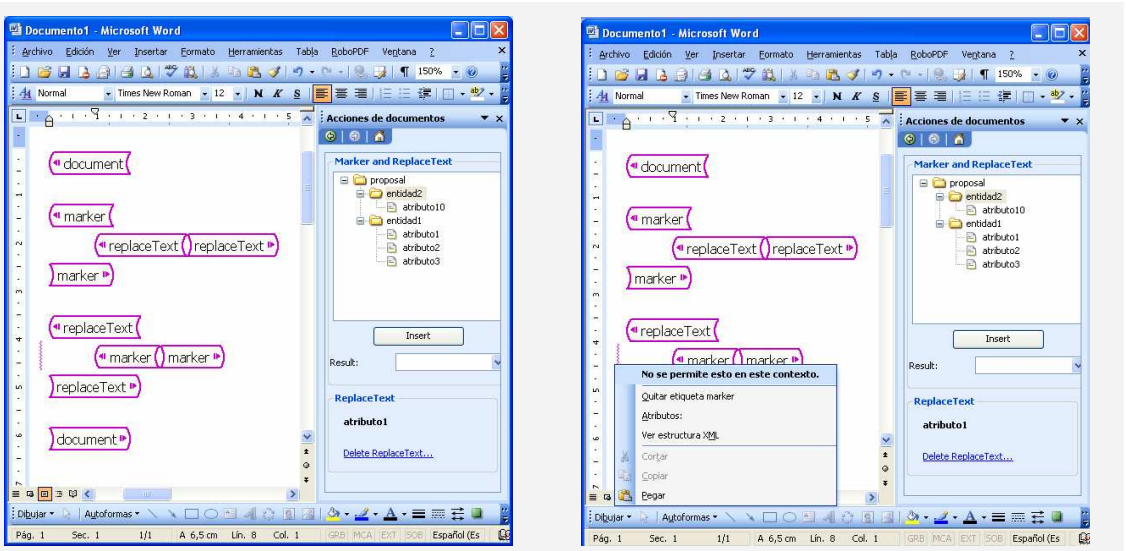


Figura 9-1 Error de estructura XML en Microsoft Word

La idea es asociar entidades (`markers`) a párrafos y secciones de texto, mientras que los atributos (`replaceText`) se asociarán a pequeñas porciones de texto: normalmente palabras.

¿Pero cómo se obtiene desde Word el listado de entidades y atributos? Durante el proceso de generación del formulario XForms en la aplicación Smart Proposal se crea un fichero XSD (Xml Schema Definition) con la definición completa de su estructura. Todos los documentos creados a partir de la plantilla SmartDocument.dot incluyen un panel de acciones personalizado. En él se deberá indicar la ruta del fichero XSD desde donde obtener la información de entidades y atributos.

A partir de este momento en el panel de acciones se hace visible una estructura en forma de árbol donde las entidades son nodos, y los atributos hojas. Al posicionarse sobre uno de estos elementos y pulsar sobre el botón Insert, en el documento Word se creará una marca de tipo `replaceText` o `marker` dependiendo del tipo de elemento que se trate.

La principal función de esta extensión es la de facilitar la gestión de documentos que integren una gran cantidad de marcas, buscando su ubicación además de indicar su nombre.

9.1 Clases

Esta extensión define el espacio de nombres SmartDocument, y para su implementación se ha utilizado el paquete de herramientas Visual Studio Tools for Office (VSTO) que se integra con Visual Studio 2005 y permite crear los llamados documentos inteligentes. VSTO permite crear soluciones basadas en la plataforma Office integrando toda la funcionalidad que brindan las clases .NET 2.0 utilizando para ello código C# o Visual Basic.

VSTO permite crear soluciones a **nivel de documento** pero no a **nivel de aplicación**. Es decir, la funcionalidad programada solo estará disponible para aquellos documentos basados en plantillas especialmente programadas, pero no para el resto. Los desarrollos a nivel de aplicación (Word, Excel, etc.) son los llamados add-ins y son creados con otras herramientas.

VSTO supone una evolución cualitativa en la creación de soluciones para Office frente a las tradicionales macros escritas en VBA (Visual Basic for Applications). El resultado es código más robusto y funcional. Todo el código generado desde Visual Studio se integra en el documento a través de una librería DLL.

9.2 Espacio de nombres SmartDocument

Las clases de este espacio de nombres están representados en la Figura 9-2, y son explicadas de forma detallada junto a sus métodos en las siguientes secciones.

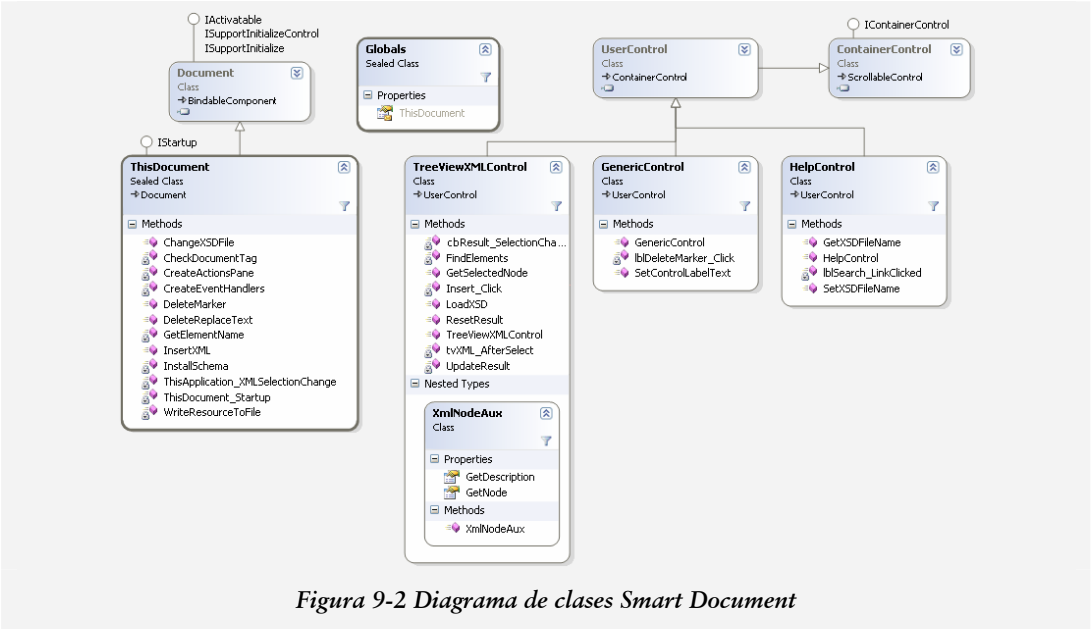


Figura 9-2 Diagrama de clases Smart Document

9.2.1 ThisDocument

Esta clase existe por defecto en todos los proyectos VSTO, y representa el documento sobre el cual se está integrando código .NET.

El método `ThisDocument_Startup` realiza tareas de inicialización y preparación y responde al evento `Startup` que es disparado al momento de abrir el documento. Desde él se llama a los métodos `InstallSchema`, `CreateActionsPane`, `CreateEventHandlers` y `CheckDocumentTag`.

El método `CreateEventHandlers` suscribe a los eventos de interés para la aplicación: `XMLSelectionChange` y `XMLBeforeDelete`, mientras que el método `CreateActionsPane` realiza la inicialización del panel de tareas de Word creando las instancias de los controles que en él aparecerán a lo largo de la ejecución de la aplicación.

El método `CheckDocumentTag` controla que en el documento exista un elemento de tipo `document`, que es el contenedor del resto de elementos. Si no existe, lo inserta automáticamente e informa de ello al usuario.

Los métodos `DeleteMarker`, `DeleteReplaceText` y `InsertXML` son los que permiten eliminar/insertar en el documento los distintos tipos de elementos. Cabe destacar que el método `InsertXML` controla antes de realizar la inserción si la operación que se intenta llevar a cabo cumple con las reglas del esquema asociado.

Los métodos `InstallSchema` y `ThisApplication_XMLSelectionChange` merecen un comentario aparte.

9.2.1.1 InstallSchema

Tal como se ha comentado al inicio de la sección, los elementos `marker` y `replaceText` no pueden combinarse de cualquier forma. Las reglas que definen la forma válida en que pueden aparecer estos elementos deben definirse en un esquema. Este esquema es almacenado en la biblioteca de esquemas mantenida por Microsoft Word. Dicha biblioteca se almacena de forma local en cada ordenador, por lo que es necesario comprobar su existencia al momento de abrir el documento. En caso de no estar disponible debe instalarse para que la solución opere de forma correcta.

Para comprobar manualmente los esquemas disponibles se debe acceder al cuadro de diálogo Plantillas y complementos a través del menú **Herramientas — Plantillas y complementos…** de Microsoft Word (Figura 9-3)

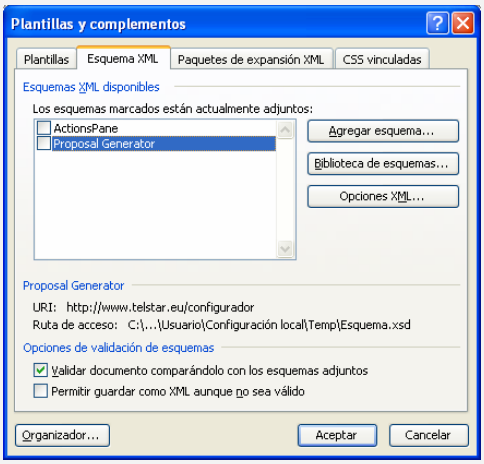


Figura 9-3 Esquemas XML disponibles en Word

En la figura anterior puede observarse la existencia del esquema llamado **Proposal Generator** creado para la ocasión. El contenido de dicho esquema es el siguiente:

```
<xsd:element name="document" type="documentRootType" />
<xsd:complexType name="replaceTextType" mixed="true">
  <xsd:attribute name="replaceTextName" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="markerType" mixed="true">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="marker" type="markerType" minOccurs="0" />
    <xsd:element name="replaceText" type="replaceTextType" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="markerName" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="documentRootType" mixed="true">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="marker" type="markerType" minOccurs="0" />
    <xsd:element name="replaceText" type="replaceTextType" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

Es importante notar la presencia del atributo `mixed` en los tipos `documentRootType`, `markerType` y `replaceTextType`. Este atributo permite que entre los distintos elementos (`document`,

`marker` y `replaceText`) aparezcan otros elementos. Esto es vital para un documento Word, ya que las marcas estarán embebidas en medio de texto, tablas, imágenes, etc. Si dicho atributo tuviese el valor de `false`, no sería posible integrar marcas en el documento de forma correcta.

Los elementos `marker` y `replaceText` tienen información asociada: su nombre. Este nombre se presenta en forma de atributo y recibe el nombre de `markerName` y `replaceTextName` respectivamente.

9.2.1.2 ThisApplication_XMLSelectionChange

El contenido del panel de tareas se actualiza a medida que el usuario se desplaza por el documento. Para ello es necesario detectar cambios de contexto a través del evento `XMLSelectionChange`. Este evento proporciona la siguiente información:

- Nodo XML antes del cambio de selección
- Nuevo XML después del cambio de selección
- Razón del cambio de selección

La razón de cambio de contexto puede ser el desplazamiento por el documento, la inserción o la eliminación de un nodo.

Si el cursor se encuentra fuera de la marca `document`, se hace visible el control para seleccionar el esquema del formulario. La ubicación de dicho fichero será almacenada en una variable interna del documento Word llamada `strXSDFile`.

Si el cursor se encuentra dentro de la marca `document`, se hace visible una estructura dispuesta en forma de árbol que permite insertar en el documento las distintas marcas.

Si estando dentro de la marca `document` el cursor se desplaza a elemento de tipo `marker` o `replaceText`, se hará visible el control que informará de su nombre. Este nombre es recogido por el método `GetElementName` desde el atributo que se inserta junto la etiqueta correspondiente.

9.2.2 HelpControl

Este control presenta un pequeño texto de ayuda, al mismo tiempo que permite especificar la ubicación del fichero XSD que almacena la información de estructura (Figura 9-4) Dicho fichero es creado en el proceso de generación del formulario.

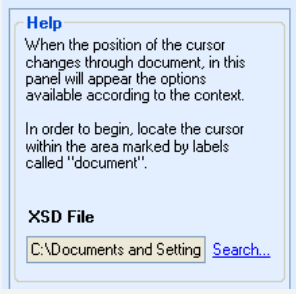


Figura 9-4 Control HelpControl

Su funcionalidad es sencilla, ya que la misión de los métodos que incluye (`lblSearch_LinkClicked`, `GetXSDFileName` y `SetXSDFileName`) es la de localizar un fichero con extensión XSD a través de un cuadro de diálogo estándar.

Si ocurre algún error (fichero inexistente, formato incorrecto, etc.) se informa de ello al usuario mediante un mensaje de error y se cancela la operación.

9.2.3 TreeViewXMLControl

Este control presenta información sobre la estructura de entidades y atributos de acuerdo al fichero especificado en el control `HelpControl` (Figura 9-5)

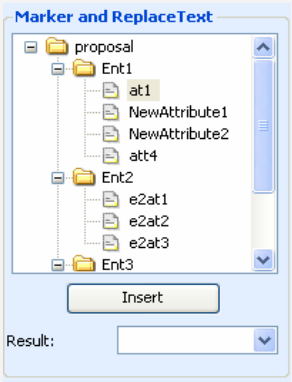


Figura 9-5 Control TreeViewXMLControl

Incluye un botón que permite insertar en el documento una etiqueta de tipo `marker` o `replaceText` de acuerdo al elemento seleccionado por el usuario. También, y de acuerdo a la selección del usuario sobre el control, brinda información sobre la ubicación en el documento de las distintas etiquetas a través del cuadro desplegable Result.

El método `LoadXSD` es el encargado de leer y cargar la información del fichero de esquema sobre un control de tipo `TreeView`.

Los métodos `ResetResult` y `UpdateResult` actualizan el resultado de la búsqueda del cuadro desplegable Result de acuerdo a las acciones realizadas por el usuario sobre el documento, así como sobre el elemento seleccionado en el `TreeView` gracias a la ayuda del método `tvXML_AfterSelect` que es llamado en respuesta del evento `AfterSelect`.

El método `Insert_Click` inserta en el documento una etiqueta de tipo `marker` o `replaceText` de acuerdo al tipo de elemento seleccionado en el `TreeView`.

Los métodos `FindResult` y `GetSelectedNode` son métodos auxiliares. El primero de ellos realiza una consulta XPath sobre el documento para hallar los elementos cuyo atributo `name` tenga el valor del nodo seleccionado en el `TreeView`. Por ejemplo, para un nodo de tipo `replaceText` la instrucción de selección es la siguiente:

```
Globals.ThisDocument.SelectNodes("//x:replaceText[./@replaceTextName = 'Ent2_at1|at1']",
"xmlns:x='http://www.telstar.eu/configurador'", false)
```

El tercer parámetro indica si se debe realizar una búsqueda rápida pasando por alto nodos de tipo texto (`true`) o buscar por todo el documento (`false`). Como resultado se obtiene una colección de clase `XMLNodes` con los nodos que cumplen el criterio especificado.

El método `GetSelectedNode` devuelve el nodo seleccionado en el control `TreeView`. Dicho nodo contiene la información necesaria para determinar el valor del atributo `name` en caso de ser insertada una marca.

El método `cbResult_SelectionChangeCommitted` es ejecutado como respuesta al evento `SelectionChangeCommitted` del combo box Result. Su función es la de seleccionar en el documento el nodo indicado por el usuario en el cuadro desplegable.

9.2.4 GenericControl

Este control solo incluye dos etiquetas comunes y una de tipo enlace: las primeras dos almacenan la definición del tipo de elemento (`marker` o `replaceText`) y su nombre (valor del atributo `name`), mientras que la otra dispara la acción que elimina el elemento.

Al momento de inicializar el panel de tareas se añaden dos instancias de este control: uno para las etiquetas de tipo `marker` y otra para las etiquetas de tipo `replaceText`. Los valores de las etiquetas se establecen a partir del método `SetControlLabelText`.

10 RESULTADOS

En este capítulo se detallarán los distintos ficheros ejecutables obtenidos (productos software), así como los requerimientos necesarios para su instalación y ejecución según el perfil de usuario: diseñador o comercial.

Para finalizar, se hará un repaso de los objetivos planteados y los finalmente alcanzados, así como los nuevos retos que han surgido consecuencia de nuevos requerimientos para este proyecto.

10.1 Smart Proposal

Esta aplicación debe ser instalada en los ordenadores de aquellas personas cuyo perfil será el de diseñador. Para su instalación se ha creado un proyecto que verifica los prerrequisitos necesarios, y en caso de no hallarse en el sistema permite instalarlos.

10.1.1 Prerrequisitos

Tal como se ha comentado en capítulos anteriores, Smart Proposal está desarrollada en C#, por lo que un prerrequisito fundamental es el paquete .NET Framework 2.0. Además, es necesario un conjunto de librerías que se utilizan en la aplicación y que permiten implementar en la interfaz de usuario el conjunto de pestañas que albergan los documentos con los que se trabaja.

10.1.2 Particularidades

Smart Proposal define su propio formato de ficheros (extensión SP) que contienen toda la información de entidades, atributos y conectores definida a través de la interfaz de la aplicación. A partir de los datos de los ficheros SP es capaz de generar un formulario XForms, y el conjunto de ficheros necesarios para su correcto funcionamiento (hoja de estilo CSS, instancia de datos, imágenes, etc)

Los formularios generados deberán ser alojados en una ubicación pública para que sean accesibles al resto de usuarios. Por ejemplo, podrían ser almacenados en una unidad de red.

10.2 Formulario

Los formularios generados desde la aplicación Smart Proposal deben ser accesibles a los usuarios de perfil comercial.

10.2.1 Prerrequisitos

El formulario XForms requiere la instalación previa de las siguientes aplicaciones:

- Procesador XForms FormsPlayer v1.5.4 (puede descargarse gratuitamente desde la página web del fabricante)
- Internet Explorer 6.0 o superior

10.2.2 Particularidades

Uno de los requisitos era la posibilidad de generar ofertas de forma offline y online. Para facilitar la gestión del cambio de los formularios, y hacerlos disponibles cuando los comerciales no se encuentren conectados a la red de la empresa, se han alojado en una unidad de red configurada como un “recurso sin conexión”. De esta forma, cuando los comerciales se marchan de viaje llevan consigo las últimas versiones de los formularios.

Esta configuración de recurso sin conexión establecida para los ficheros XForms, también es necesaria para los ficheros Word que contienen las marcas con la ubicación donde la información será integrada, creadas con la extensión Smart Document por el diseñador.

10.3 Proposal Generator

Este control ActiveX debe ser instalado a los usuarios interesados en generar ofertas, normalmente los comerciales. Su instalación se lleva a cabo a través de un instalador que verifica los prerrequisitos necesarios.

10.3.1 Prerrequisitos

Proposal Generator requiere para su correcto funcionamiento:

- Microsoft .NET Framework 2.0
- Microsoft Office 2003 Primary Interop Assemblies

10.3.2 Particularidades

Los documentos (ofertas) creados a través de este control son documentos Word que no necesitan la instalación de ningún componente especial para ser abiertos, ya que toda que la información adicional añadida por Smart Document (marcas XML) ha sido eliminada en el proceso de generación de los mismos.

10.4 Smart Document

Esta extensión de Microsoft Word debe ser instalada, al igual que Smart Proposal, en los ordenadores de los usuarios diseñadores. Ellos serán los encargados de marcar las posiciones en los documentos donde la información de las ofertas (proveniente de los datos recogidos por los formularios XForms) será integrada.

10.4.1 Prerrequisitos

Para funcionar correctamente, esta aplicación necesita de:

- Microsoft .NET Framework 2.0
- Microsoft Word 2003 con soporte para soluciones personalizadas (Versiones Standalone o Professional).
    - Se deberá incluir de forma explícita la instalación de los módulos de soporte .NET para Word, Microsoft Forms y Smart Tags (Figura 10-1)
- Microsoft Visual Studio 2005 Tools for Office Runtime
- Microsoft Office 2003 Primary Interop Assemblies

Para simplificar el despliegue de la aplicación, se ha creado un programa de instalación que verifique los requisitos necesarios y en caso de no encontrar alguno de ellos, permitir su instalación en los casos que sea posible.

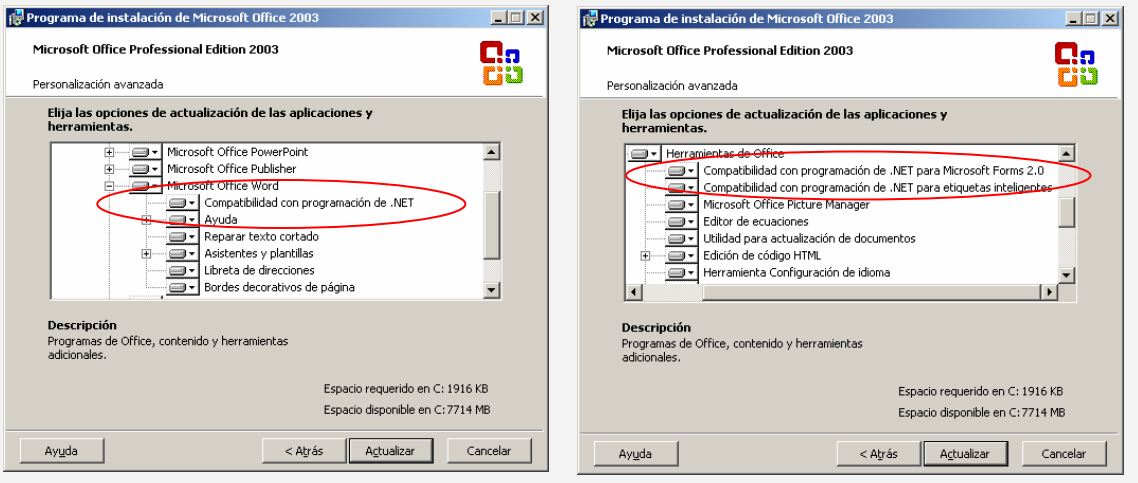


Figura 10-1 Instalación prerrequisitos Smart Document

10.4.2 Particularidades

Una vez instalada la extensión, en la carpeta de la aplicación se encontrarán una serie de ficheros de los cuales los más importantes son:

- SmartDocument.dot: Plantilla Microsoft Word a partir de la cual se crearán todos los documentos donde sean necesarios introducir marcas XML.
- SmartDocument.dll: Librería de aplicación que incluye toda la lógica programada.

Los documentos creados a partir de SmartDocument.dot serán guardados en formato XML y jugarán el papel de plantillas para las ofertas. Por tal motivo, deberán ser accesibles para el resto de usuarios. Pero ellos (no diseñadores) nunca tendrán necesidad de abrir directamente estos documentos (como un documento tradicional), y por lo tanto no es necesario que tengan instalada esta extensión. Los documentos basados en SmartDocument.dot serán manipulados por el control ActiveX Proposal Generator desde el formulario XForms.

10.4.2.1 Seguridad

La seguridad del código .NET –al tratarse de código de tipo administrado (managed), a diferencia del código C++ de tipo no administrado (unmanaged)- se rige a partir de las directivas de seguridad definidas en cada equipo.

Estas directivas, se definen mediante la utilidad `CASPOL.EXE` (Code Access Security Policy Tool) que permite definir la confianza del código en función de su ubicación (URL), una firma digital (certificado), un nombre fuerte (strong name) o una combinación de las anteriores.

`CASPOL.EXE` permite administrar las políticas CAS para cada versión de .NET instalada en el equipo. Es decir, habrá que configurar las políticas de seguridad de acuerdo a la versión de .NET Framework que utilice la aplicación.

En este caso en particular, la versión de framework utilizada es la 2.0.50727, por lo que la herramienta CASPOL.EXE se encontrará en la siguiente ubicación:

`C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727`; para otras versiones de framework la ubicación será `C:\WINDOWS\Microsoft.NET\Framework\vX.X.XXXXX`

Con CASPOL se puede definir políticas de seguridad a nivel de Empresa, Máquina y Usuario. En este caso, se definirán políticas a nivel de máquina para el correcto funcionamiento de los ensamblados diseñados para los documentos Word.

Por cada nivel, se definen grupos a los cuales se les asigna una identificación y una confianza en base a unos parámetros definidos (los comentados anteriormente: URL, certificado, etc.) Por defecto existen los grupos llamados `MyComputer`, `LocalIntranet`, `Internet`, `Restricted`, `Trusted` que tienen permisos asignados de forma predeterminada.

El acceso de ensamblados en un disco local (grupo `MyComputer`) tiene permisos `FullTrust`; es decir que se podrá cargar y ejecutar. El acceso de ensamblados desde una unidad de red (grupo `LocalIntranet`) no tiene permisos `FullTrust` (ya que no es una ubicación de confianza) y deben definirse explícitamente reglas para poder acceder a ensamblados en dichas unidades.

En el caso de Smart Document, y para conceder permisos de ejecución a una solución almacenada en una unidad de red, en cada equipo se deberá ejecutar los siguientes comandos:

- `caspol -pp off -m -ag LocalIntranet_Zone -url "\\ficheros\Ofertas\*" FullTrust -n "Telstar_SmartOffice_GeneradorOfertas"`
- `caspol -pp on`

`-pp off`: No se realizarán preguntas al usuario al modificar la CAS. Si el usuario tiene derechos de administrador sobre el equipo, las modificaciones se harán de forma "silenciosa", en caso contrario fallarán. Este modificador altera el comportamiento predeterminado de CASPOL (preguntar al usuario por cada cambio realizado) hasta que no se ejecute el comando "`caspol -pp on`"

`-m`: Hace referencia a las reglas a nivel de máquina.

`-ag LocalIntranet_Zone`: Añadir el nuevo grupo al grupo llamado `LocalIntranet_Zone`. El nombre de este nuevo grupo vendrá definido por el parámetro `-n`. En el caso que los ensamblados sean instalados de forma local, el grupo que debe especificarse es `My_Computer_Zone`

`-url "\\ficheros\Ofertas\*"`: Se da permisos de ejecución en base a la ubicación de los ensamblados y documentos. En este caso, se confiará en todos los ejecutables (dlls, docs, xls, etc) ubicados en el recurso `\\ficheros\Ofertas\` y sus subdirectorios.

`FullTrust`: Se da permisos de total confianza a los ensamblados.

Notas:

- La url también podría especificarse por la letra de unidad que se tiene mapeada en el equipo, pero un cambio en la letra de unidad obligará a actualizar la política de seguridad definida con CASPOL.

- Los documentos y ensamblados –para funcionar correctamente- deben estar ubicados en una carpeta con permisos `FullTrust`.
- Para simplificar el despliegue y actualización de los ficheros, la instalación de la extensión puede realizarse una única vez en un sitio de red.
- Adicionalmente, la configuración de las políticas de seguridad puede realizarse a través de la aplicación instalada por .NET Framework y ubicada en **Panel de control — Herramientas administrativas — Microsoft .NET Framework 2.0 Configuration**

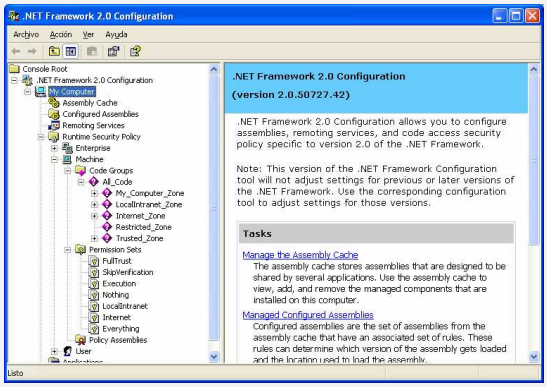


Figura 10-2 Interfaz gráfica configuración CAS

10.5 Objetivos propuestos y alcanzados

El objetivo global de este proyecto era el de diseñar e implementar un sistema de generación de ofertas que acabe con los problemas que existían desde hace tiempo en TELSTAR.

Para ello se ha propuesto la creación de un conjunto de aplicaciones que de forma global cumplieran los requerimientos planteados por las distintas partes del departamento comercial implicadas. A medida que se iban presentando resultados intermedios, aparecían nuevos requerimientos mientras otros cambiaban.

El principal objetivo del proyecto ha sido cumplido: separar los datos contenidos en las ofertas de su presentación. Esto se ha logrado almacenando la información en ficheros XML lo que permite su reutilización y explotación por parte de otras aplicaciones. Por ejemplo, la información de las ofertas podría ser almacenada en un sistema de bases de datos, o en este caso, integrada en documentos Word.

Pero una vez el principal problema ha sido solucionado, al departamento comercial de TELSTAR ha planteado nuevas inquietudes que podrían transformarse en nuevos requerimientos. Por ejemplo, se ha comentado el interés existente en integrar la información económica de las ofertas en el sistema de ERP de la empresa, así como la idea de centralizar todos los datos técnicos para su reutilización en los documentos del departamento de calidad. También se propuso la idea de incluir en las ofertas la información de los clientes proveniente desde el ERP.

Algunas de estas nuevas características obligarían a la aplicación Smart Proposal a “abrirse” al mundo exterior haciéndola capaz de procesar fuentes de información externa para su uso en valores y condiciones de atributos y conectores. A día de hoy, esta información está “codificada” en los documentos SP. Otras de las nuevas características propuestas podrían ser resueltas de forma sencilla utilizando XForms, junto al uso de servicios web allí donde sea posible.

Otro punto mejorable es el referente al aspecto del formulario XForms generado. A pesar que se haya definido un primer modelo de presentación de forma conjunta con el departamento comercial, no se ha tardado en solicitar algunos cambios. Muchos de ellos pueden llevarse a cabo modificando la hoja de estilos CSS, pero lo ideal sería que ellos mismos pudieran hacerlo de forma gráfica a través de algún tipo de editor.

Además de los puntos mencionados anteriormente, existen algunos requerimientos iniciales (no funcionales) que no se han implementado de forma completa, como por ejemplo la impresión de diagramas. Esta pausa en el desarrollo de Smart Proposal está motivada por las dudas existentes sobre su reemplazo por InfoPath (Para más detalles sobre InfoPath, su integración con otras aplicaciones y comparativa con XForms consultar el capítulo 13) Por lo

tanto, la implementación de los requerimientos pendientes y los nuevos objetivos planteados a partir del desarrollo de este proyecto están a la espera de saber si finalmente la aplicación utilizada para recolectar información será InfoPath o Smart Proposal-XForms.

11 CONCLUSIONES

La forma de confeccionar ofertas en TELSTAR requería un gran cambio, ya que el sistema utilizado no era adecuado: El uso de documentos Word con macros que requerían constantes actualizaciones y no funcionaban de forma esperada, la imposibilidad de recuperar la información una vez plasmada en un documento -entre otras razones- han motivado el cambio.

El sistema propuesto y desarrollado ha sido visto con buenos ojos por todas las partes implicadas (departamentos comerciales y departamento de IT), pero a día de hoy no se encuentra implantado de forma total. Uno de los motivos que originan el retraso en su puesta en marcha es que el plug-in para Internet Explorer formsPlayer presenta algunos problemas a la hora de manejar formularios con una interfaz de usuario algo compleja, lo que origina errores en su representación. Se verá el comportamiento del plug-in en sus próximas versiones (en las que prometen mejorar aspectos relacionados con la interfaz y el rendimiento). En caso de no satisfacer las necesidades requeridas, se estudiará la implementación utilizando el motor XForms de la empresa Orbeon.

Pero el principal motivo por el cual aún el sistema no se ha puesto en marcha es debido a que se está estudiando la utilización de InfoPath para la captura de datos a través de formularios. Antes del inicio de este proyecto no se contaba con ningún contrato de licencia con Microsoft, por lo que el uso de esta aplicación había sido descartado. Pero las cosas cambiaron, y en este tiempo se ha llegado a un acuerdo económico con Microsoft que (entre otras tecnologías y aplicaciones) permite a TELSTAR utilizar InfoPath. Este hecho, sumado a la implantación de Microsoft Dynamics como gestor CRM (Customer Relationship Management) hacen que la utilización de InfoPath –en detrimento de XForms- sea replanteada.

El uso de InfoPath como medio para desarrollar formularios y capturar información reemplazaría a la aplicación Smart Proposal y el formulario XForms generado. El resto de elementos presentados en este proyecto (Smart Document para el marcado de plantillas y el control ActiveX Proposal Generator) se mantendrían sin cambios; ya que el concepto principal sigue siendo el mismo: separar en todo momento funcionalidad, contenido y presentación.

Pero a pesar que finalmente se tome la decisión de utilizar InfoPath, considero un hecho positivo haber podido descubrir XForms, tecnología que abre un gran abanico de aplicaciones y que aún no es lo suficientemente popular. Esta situación cambiará cuando los navegadores web brinden soporte a XForms de forma nativa (tal como se hace con CSS, por ejemplo) o existan extensiones que permitan su uso sin presentar problemas de compatibilidad o de funcionalidad. Estos inconvenientes vienen dados por la interpretación que las distintas implementaciones hacen del estándar, las suposiciones tomadas allí donde no hay nada dicho, o como herencia de las incompatibilidades de los navegadores que existen al día de hoy.

12 ANEXO A - REQUERIMIENTOS

Inicialmente los requerimientos son ideas o intenciones, que no siempre son coherentes o fáciles de formalizar. La especificación de los requerimientos es la base para construir el producto: Debe contener un claro, completo y verificable conjunto de instrucciones para su construcción.

A continuación se detallan los requerimientos especificados en las distintas reuniones con los usuarios del departamento comercial.

12.1 Requerimientos iniciales

Número requerimiento: 1	**Tipo**: Funcional
Descripción: La aplicación deberá generar la oferta en formato DOC.	
Justificación: Los usuarios utilizan este programa para la creación y edición de documentos. Después de generada, la oferta podrá ser editada para añadir modificaciones si fuese necesario. Ejemplo: cuando una oferta es enviada a la atención de un distribuidor local y éste a su vez la reenvía al cliente final cambiando los datos de contacto de la misma. La utilización de otro programa de edición de textos implicaría un coste adicional en formación a los usuarios.	
Fuente: TL (Jefe de comerciales)	
Criterio de prueba: El documento generado deberá tener formato DOC y deberá ser abierto con Microsoft Word sin presentar problemas.	
Historia: 14/11/2006 - Toma de requerimientos	

Número requerimiento: 2	**Tipo**: No Funcional
Descripción: El contenido de los documentos (ofertas) deberá ser gestionado por el propio departamento comercial.	
Justificación: La dependencia de otros departamentos para modificar o alterar el contenido de las plantillas de oferta no es práctico, ya que varían con cierta periodicidad. Es necesario por lo tanto una autogestión.	
Fuente: TL (Jefe de comerciales)	
Criterio de prueba: El contenido de las ofertas podrá ser editado por los usuarios, independientemente de la solución implementada.	
Historia: 14/11/2006 - Toma de requerimientos	

Número requerimiento: 3	**Tipo**: No Funcional
Descripción: La generación de las ofertas y las plantillas deberán realizarse de forma sencilla. No se deberá requerir de conocimientos técnicos específicos. (extensión de requerimiento #2)	

Justificación:
A día de hoy la modificación de una plantilla implica el conocimiento de Visual Basic, con lo que muy pocas personas del departamento están en condiciones de modificarlas; y si lo hacen es con muchas limitaciones.
Fuente:
TL (Jefe de comerciales)
Criterio de prueba:
Presentar al usuario el proceso de generación una vez se llegue a un estado lo suficientemente avanzado del proyecto, para que pueda evaluar la dificultad del mismo.
Historia:
14/11/2006 - Toma de requerimientos

Número requerimiento: 4	**Tipo**: Funcional
Descripción:	
Una misma oferta podrá ser generada en distintos idiomas. El esfuerzo para generar las distintas versiones deberá ser mínimo. La cantidad de información contenida en los documentos variará, para adaptarlos a las características del mercado al que va dirigido.	
Justificación:	
Las ofertas van dirigidas a clientes de diferentes mercados, los cuales están ubicados en distintas zonas geográficas y utilizan distintos idiomas. Un comercial deberá ser capaz de generar una oferta en distintos idiomas y formatos: carta, e-mail, detallada, resumida, etc. en base a unos datos comunes a todos ellas.	
Fuente:	
TL (Jefe de comerciales)	
Criterio de prueba:	
En base a una serie de datos recogidos para una oferta, generar documentos en distintos idiomas y/o formatos.	
Historia:	
14/11/2006 - Toma de requerimientos	

Número requerimiento: 5	**Tipo**: Funcional
Descripción:	
Los documentos podrán incluir cualquier tipo de contenido: tablas, imágenes, viñetas, dibujos, hipervínculos, comentarios, etc. Los formatos de fuentes deberán ser respetados en todo el documento, así como la distribución de los distintos objetos.	
Justificación:	
El diseño de la imagen (aspecto general) de una oferta se lleva una parte importante del esfuerzo empleado en su creación. Por ello, el sistema de generación deberá respetar todos los formatos y objetos contenidos en ella.	
Fuente:	
TL (Jefe de comerciales)	
Criterio de prueba:	
Generación de una oferta en la que se incluya distintas combinaciones de fuente y colores además de otros elementos de Microsoft Word.	
Historia:	
14/11/2006 - Toma de requerimientos	

Número requerimiento: 6	**Tipo**: Funcional

Descripción:
En una oferta podrán existir elementos opcionales. Asociados a ellos habrá distintas secciones en el documento. La no inclusión de un opcional implica la eliminación de dichas secciones en toda la oferta.
Justificación:
El formato de ofertas utilizado normalmente por TELSTAR está dividido en dos bloques. El primero incluye las descripciones y características de los elementos ofertados, el segundo incluye el detalle económico. La no inclusión de un apartado en uno de los bloques, implica su eliminación de la otra sección.
Fuente:
TL (Jefe de comerciales)
Criterio de prueba:
Generar una oferta donde no se incluyan una serie de opcionales. Verificar el contenido del documento creado.
Historia:
14/11/2006 - Toma de requerimientos

Número requerimiento: 7	**Tipo**: Funcional
Descripción:	
La generación de ofertas no debe estar condicionada a una conexión a Internet o a la conexión a la red corporativa.	
Justificación:	
Un gran número de comerciales están la mayor parte del tiempo viajando por distintos países, y no siempre se contará con una conexión a Internet al momento de crear una oferta.	
Fuente:	
TL (Jefe de comerciales)	
Criterio de prueba:	
Generar una oferta desde un ordenador que no disponga conexión a red.	
Historia:	
14/11/2006 - Toma de requerimientos	

Número requerimiento: 10	**Tipo**: Funcional
Descripción:	
La información introducida por el usuario para la generación del documento deberá ser validada, impidiendo que una oferta sea creada hasta que los datos no cumplan una serie de criterios.	
Justificación:	
Existen máquinas que debido a sus características no permiten ciertas configuraciones. Validando la información, los comerciales más inexpertos evitarán cometer errores de forma involuntaria.	
Fuente:	
TL (Jefe de comerciales)	
Criterio de prueba:	
Introducir datos de forma tal que no se cumplan las reglas de validación definidas. Comprobar que el documento no puede ser generado. Al corregirlos, comprobar que si es posible generar la oferta.	
Historia:	
14/11/2006 - Toma de requerimientos	

12.2 Requerimientos adicionales

Una vez formalizados los principales requerimientos, se acordó una segunda reunión en donde los futuros usuarios utilizarían un prototipo de la aplicación.

El prototipo pretende presentar un producto suficientemente real para que los potenciales usuarios puedan imaginar nuevos requerimientos que de otra forma se perderían. En ocasiones los requerimientos no son evidentes hasta que alguien utiliza realmente el producto.

Número requerimiento: 11	**Tipo**: No Funcional
Descripción: La aplicación Smart Proposal debe permitir cambiar la forma en que las entidades son representadas en pantalla: colores de líneas y rellenos, tipo y tamaño de los textos.	
Justificación: Motivado por el aspecto estético de los documentos presentados en Smart Proposal.	
Fuente: JM (Comercial)	
Criterio de prueba: Al seleccionar un tipo de letra, tamaño o color de texto; o color y grosor de líneas, los cambios deben reflejarse en la interfaz de usuario.	
Historia: 02/04/2007 - Toma de requerimientos adicionales	

Número requerimiento: 12	**Tipo**: Funcional
Descripción: La aplicación Smart Proposal debe permitir imprimir los diagramas creados.	
Justificación: Contar con un soporte papel de los diagramas generados por la aplicación facilitará el análisis y discusión del mismo en distintas reuniones.	
Fuente: JM (Comercial)	
Criterio de prueba: Realizar una impresión de un diagrama.	
Historia: 02/04/2007 - Toma de requerimientos adicionales	

Número requerimiento: 13	**Tipo**: Funcional
Descripción: La aplicación Smart Proposal debe permitir cambiar el orden en que las entidades y atributos son representados en pantalla en el formulario creado.	
Justificación: El orden "natural" en que deben ser rellenados los distintos valores (atributos) puede diferir al orden en que los mismos fueron definidos. Lo mismo sucede con las opciones (entidades) que aparecen en el menú lateral del formulario.	
Fuente: JM (Comercial)	
Criterio de prueba: Al crear un documento Smart Proposal, cambiar el orden de las entidades y atributos.	

Historia:
02/04/2007 - Toma de requerimientos adicionales

13 ANEXO B – INFOPATH

13.1 InfoPath

InfoPath es una nueva aplicación disponible a partir de la versión 2003 de Microsoft Office. A diferencia de las otras aplicaciones Office, InfoPath está diseñado para conectarse a datos de otros usuarios y servicios, además de estar desarrollado desde su inicio con el propósito de trabajar con XML. InfoPath provee un entorno para crear formularios que contienen información estructurada (almacenada en XML) y un framework para conectar esta información a la web, servicios web o cualquier aplicación que procese XML.

InfoPath rellena el hueco dejado entre la visión orientada a documento de Word y el enfoque a datos de Excel y Access.

InfoPath brinda un conjunto de herramientas para crear formularios de acuerdo a la estructura definida en un esquema XSD, permitiendo de forma gráfica definir e integrar componentes (botones, cuadros de texto, listas…) Un ejemplo de formulario puede verse en la Figura 13-1. La misma información puede presentarse en distintas vistas, haciendo posible –por ejemplo- que un usuario rellene un formulario con los datos que conoce y en otra fase de un proceso añadir más información.

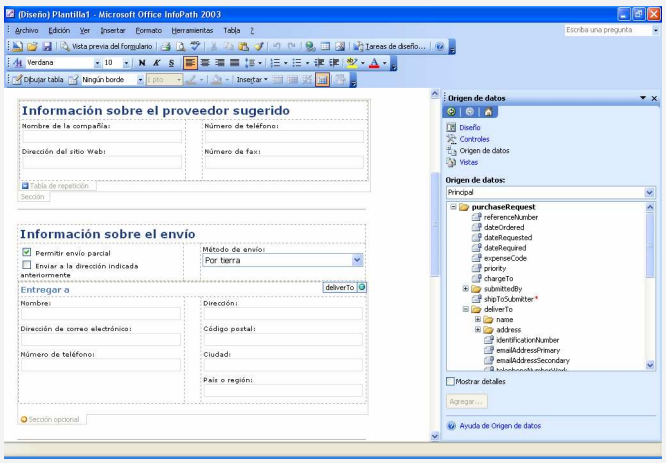


Figura 13-1 Formulario InfoPath

InfoPath incluye un entorno de desarrollo para construir formularios y los módulos necesarios para que puedan ser abiertos y rellenados por los usuarios. Sus ficheros reciben el nombre de "soluciones" y permite crear y editar documentos XML sin la necesidad de tener conocimientos específicos. Por cada tipo de documento-formulario que se quiera construir es necesario crear una solución, donde la expresión "tipo de documento" se refiere a la especificación de un esquema. La Figura 13-1 enseña una solución en modo diseño donde a la derecha puede verse el "origen de datos" –esquema del documento- y en la parte central la distribución de estos contenedores de información representados por distintos tipos de controles. Los tipos de control InfoPath más importantes son las tablas de repetición, secciones de petición y secciones opcionales. Las primeras dos permiten insertar ítems de forma repetitiva: por ejemplo, líneas en un pedido de compra; mientras que la otra representa partes opcionales de un formulario que el usuario puede eliminar o insertar.

Las soluciones InfoPath están basadas en XSLT, XPath, HTML, CSS, WSDL, SOAP, DOM, EcmaScript –en otras palabras, el núcleo de XML- sin definir ningún nuevo estándar. La distribución de los controles y textos se realiza por medio de HTML y CSS, además de elementos propios definidos en InfoPath. XSLT es utilizado para transformar el documento XML que está siendo editado en una vista de formulario HTML. La información contenida en el esquema XSD asociado se utiliza para garantizar la validación de los datos al mismo tiempo que el formulario se rellena. Utilizando el modelo de objetos InfoPath es posible programar scripts (JScript o VBScript) utilizando DOM para personalizar el comportamiento del editor cuando sea necesario. Los datos captados a través de InfoPath son en formato XML puro, válido de acuerdo al esquema definido.

Tal como se ha comentado, InfoPath soporta dos tipos de tareas: el diseño y el rellenado de los formularios. La tarea de completar un formulario es responsabilidad de un usuario final, pero la tarea de su diseño corresponde a un usuario con conocimientos algo más avanzados. En modo diseño, InfoPath es necesario para diseñar soluciones, pero no es la única forma de hacerlo. Ya que se encuentra basado en tecnologías XML las soluciones pueden diseñarse "a mano" utilizando un conjunto de herramientas XML a elección. Esto último es desaconsejable, ya que las características añadidas no soportadas por el editor pueden hacer que una solución no sea modificable desde InfoPath.

13.2 InfoPath y Sharepoint

Microsoft SharePoint es un grupo de tecnologías que permiten compartir información en Internet. La familia SharePoint incluye dos versiones: Windows Sharepoint Services (WSS) y Sharepoint Portal Server (SPS). Gracias a estas tecnologías es posible acceder a la información y organizarla, gestionar documentos, trabajar en equipo en proyectos y todo ello en un entorno integrado con Microsoft Office. La versión WSS de Sharepoint es gratuita (solo disponible para versiones Server de los sistemas operativos Windows), e integra funcionalidades básicas que son ampliadas y mejoradas en la versión SPS que debe ser adquirida bajo licencia.

Sharepoint es una herramienta de administración de contenidos que permite compartir información dentro de una organización creando páginas web de forma muy rápida y sencilla. Para ello, la información se organiza en conjuntos llamados listas y librerías. De estas últimas existen varios tipos, mereciendo especial mención las siguientes:

- **Librerías de documentos**: permite almacenar distintos tipos de ficheros, no solo documentos Word o Excel.
- **Librería de imágenes**: Almacena ficheros de imágenes y brinda funcionalidades útiles para este tipo de documentos: vista previa, modo presentación, etc.
- **Librerías de formularios**: Se integran de forma completa con InfoPath, permitiendo la publicación de las soluciones generadas desde esta aplicación en este tipo de librerías. (Figura 13-2)

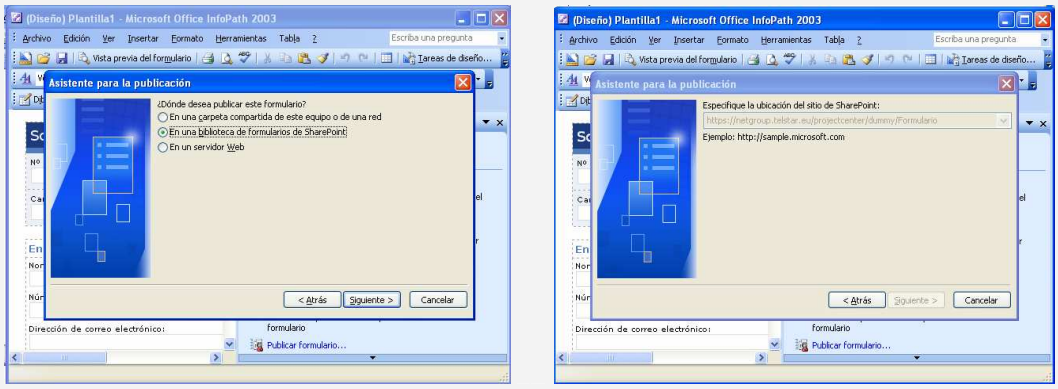


Figura 13-2 Publicación solución InfoPath en Sharepoint

De esta forma, la edición y recolección de datos estará disponible en todo momento a través de Internet (Figura 13-3 y Figura 13-4) Solo existe un requisito: todos los usuarios deben tener instalado InfoPath en sus equipos para poder utilizar los documentos de este tipo de librerías.

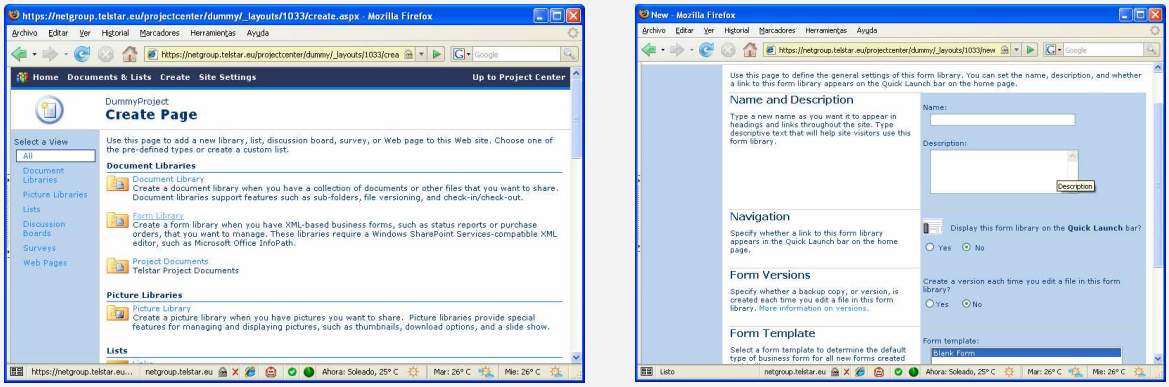


Figura 13-3 Creación de una librería de formularios

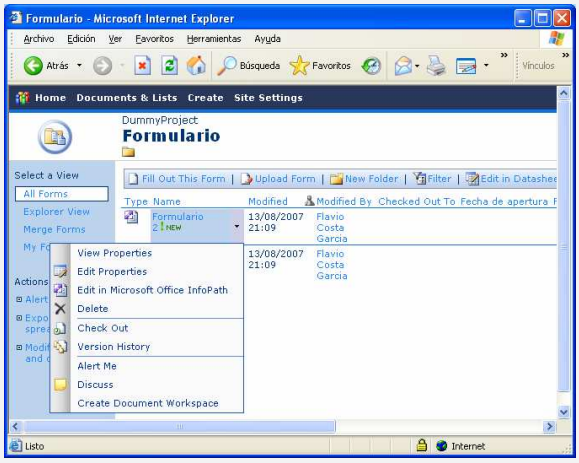


Figura 13-4 Formulario InfoPath en librearía Sharepoint

13.3 InfoPath vs XForms

El problema que intenta solucionar tanto InfoPath como XForms es el siguiente:

Existe una aplicación que necesita recoger datos, almacenarlos y presentarlos. Además es necesario reutilizar esta información en diferentes contextos, ya sea para realizar análisis o presentarla en diferentes formatos. Por lo tanto, XML es la solución. A continuación se diseña un esquema XML para los documentos. ¿Cómo introducirán los usuarios la información en el formato correcto? Es decir, ¿Cómo se les permite crear documentos XML? A través de una interfaz de usuario amigable que les permita hacerlo, sin necesidad de que tengan que conocer detalles de diseño del fichero XML que están creando.

InfoPath y XForms comparten las siguientes características:

- Permiten al usuario final editar documentos XML a través de una interfaz basada en formularios.
- Utilizan esquemas XSD como mecanismo para validar el contenido.
- Ambos están diseñados para consumir servicios web.
- Basados en un modelo centrado en la repetición de ítems/elementos, que permite extender los formularios de forma dinámica.

Aparte de estas semejanzas, XForms e InfoPath tienen muchas diferencias. InfoPath es una implementación única que solo funciona en entornos Windows. En contraste, XForms es una especificación sobre la cual se han creado diferentes implementaciones, permitiendo la interoperabilidad de formularios web con un gran número de clientes. Más que optimizado para un interfaz de usuario única, XForms realiza una abstracción separando los controles del formulario de la forma como serán presentados.

A pesar del énfasis puesto en InfoPath en la integración de servicios web, no representa la solución definitiva para la próxima generación de clientes web. En su lugar, está diseñado para formar parte de la intranet de las empresas, remplazando los formularios de papel y brindando soporte a los datos de la compañía y aplicaciones de gestión de contenidos. Como parte de las aplicaciones de la suite Microsoft Office, **provee funcionalidad offline** no soportadas por tecnologías basadas en web como XForms.

Ha existido cierta polémica por el hecho que InfoPath no brinde soporte a XForms a pesar de que en muchos aspectos InfoPath parece una aplicación dirigida al mismo segmento que XForms. Algunos ven InfoPath como otro producto Microsoft que elije ir por su propio camino en lugar de seguir los estándares, a pesar de estar basado en alguno de ellos como HTML, XSD y XSLT. En otras palabras: formato propietario construido sobre especificaciones públicas, integrando un conjunto de herramientas que no son de dominio público ni creadas de forma cooperativa. Las voces más críticas acusan a Microsoft de reinventar XForms, ya que InfoPath brinda similar funcionalidad pero a través de un enfoque distinto.

En definitiva, XForms e InfoPath son similares, pero distintos. Ambos dan solución al problema de convertir las acciones de los usuarios en XML pero difieren en el objetivo, la forma de hacerlo y al público al que va dirigido.

- **Objetivo:** InfoPath está centrado en brindar un gran entorno visual, de la misma calidad que el resto de aplicaciones Office, para crear y rellenar formularios. En contraste, la especificación XForms está diseñada para definir el comportamiento asociado a los controles que recogen información, además de ofrecer una gran variedad de formas en la que los mismos pueden ser representados a través de nuevas propiedades CSS. Además mientras XForms parece ser diseñado para ser generado fácilmente por herramientas automatizas, InfoPath se encuentra en el otro extremo.
- **Público**: Los requerimientos recomendados para ejecutar InfoPath son elevados: como mínimo un ordenador Pentium III con Windows 2000 SP3 o superior. Además InfoPath forma parte del paquete profesional de Office que en la práctica es utilizado por grandes organizaciones. En contraste XForms fue diseñado para trabajar sobre el mayor número de dispositivos disponibles, desde teléfonos, PDAs o servidores. El software XForms está disponible en paquetes –comerciales o de libre distribución- y para un gran número de plataformas.
- **Forma**: Los formularios XForms se desarrollan utilizando sintaxis XML declarativa, mientras que InfoPath –de la misma forma que los formularios HTML- fomenta la programación de scripts. Otras diferencias se encuentran en los controles soportados: Por ejemplo, InfoPath soporta listas ordenadas y desordenadas pero no brinda un control que permita selección múltiple.

13.4 OpenOffice

Mientras InfoPath está disponible en la versión profesional de Microsoft Office o como producto independiente, la suite ofimática OpenOffice en su versión 2.0 ha añadido a su editor de textos soporte para XForms como nueva característica.

De esta forma, es posible diseñar formularios simples de manera sencilla sin necesidad de programar código.

14 ANEXO C – XML EN OFFICE 2003

La mayoría de personas utiliza Office como un conjunto de aplicaciones específicas para hacer su trabajo y no como una interfaz de propósito general para manejar información. Esto es así ya que habitualmente se intercambian ficheros por email y siempre que se necesita reutilizar información se recurre al método de cortar-pegar o conversiones de formato.

A partir de la versión 2003 del paquete de herramientas Office, Microsoft brinda nuevas oportunidades a los desarrolladores permitiendo procesar la información contenida en los documentos. En lugar de crear documentos Word, el usuario puede crear fichero de datos que pueden ser compartidos por otros procesos y sistemas. Esto es posible ya que XML se ha integrado en Office, siendo Word el principal exponente de esta nueva tendencia.

14.1 ¿Por qué XML?

XML define un formato basado en texto que contiene etiquetas y estructuras. XML se asemeja mucho a HTML pero permite a los usuarios crear sus propios formatos en lugar de limitarlos a uno único. Es decir, XML permite definir formatos abiertos y públicos -por la propia definición y diseño de XML- a diferencia de los clásicos formatos propietarios.

La incorporación de XML en Office permite crear de forma sencilla software que se integre con los programas que forman parte de esta suite ofimática.

XML es solo una pieza de un amplio conjunto de tecnologías. XSLT es un lenguaje de transformación basado en XML que permite transformar un documento utilizando reglas. XSLT realiza muchas de las tareas en Office; es la clave para convertir información en el formato requerido por las aplicaciones Office y viceversa. Otra especificación, XML Schema o XSD, proporciona la descripción de las estructuras de los distintos formatos de documentos Office para que las aplicaciones que trabajan con ellas puedan utilizarlas como base para su procesado.

Cada aplicación Office que trabaja con XML está orientada a un uso particular de esta tecnología. Los programadores normalmente ven XML como un contenedor para guardar datos, ya que su cometido es intercambiar información. Por ello comienzan definiendo una estructura y escribiendo código a partir de ella. Muchos documentos contienen información estructurada (como tablas y listas) que encajan bien en estos modelos "fuertemente estructurados" pero a veces es necesaria una alternativa para situaciones que no se pueden predecir por adelantado. XML brinda esta flexibilidad para representar distintos tipos de formatos/estructuras.

XML no es la solución a todos los problemas, pero puede hacerlos más fáciles de resolver.

14.2 XML en Word 2003

Word comenzó como un programa que permitía a la gente volcar al ordenador textos, a modo de máquina de escribir "electrónica". Con el paso del tiempo Word comenzó a integrar más características como la combinación de correspondencia y la edición de páginas web, pero seguía orientado enteramente a los documentos, finalidad para la cual fue concebido.

A partir de la versión 2003, Word ha extendido su modelo orientado a documentos tradicionales para añadir funcionalidades sobre documentos XML.

Word abarca XML en dos niveles. El primero de ellos, y sin mucho esfuerzo para el usuario, consiste en guardar los documentos como XML utilizando un vocabulario que refleja la comprensión del documento propia de Word. Estilos, formatos, comentarios, marcas de revisión, metadatos y todo aquello que es posible crear en un fichero DOC se mantiene. Toda esta información (exceptuando los objetos embebidos, almacenados en cadenas codificadas en Base64) es legible y accesible, por lo que es posible utilizar editores de texto o herramientas específicas para procesarla. Word puede abrir estos ficheros de la misma forma que un documento en formato DOC, haciendo posible que otras aplicaciones creen documentos XML para Word.

Word lleva estas características a otro nivel permitiendo la creación de vocabularios XML propios y permitir la edición de documentos insertando estas marcas. La creación de este tipo de documentos requiere del conocimiento de XSD, XML y XSLT; pero no son necesarios para su uso. Por lo que una vez que las plantillas son creadas, los usuarios pueden simplemente editar XML a través de la interfaz de Word (Figura 14-1)

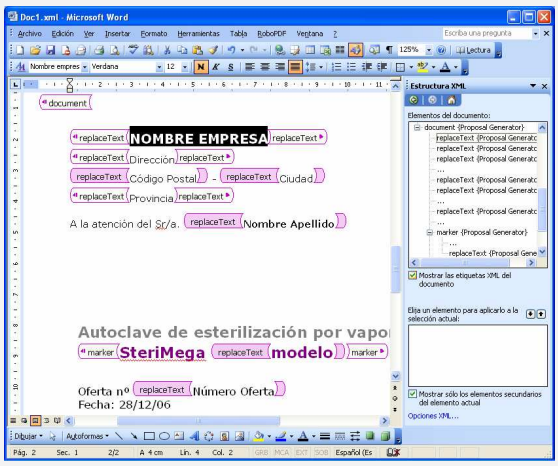


Figura 14-1 Estructura XLM de un documento

La integración de XML en Word es un gran avance, ya que Microsoft ha llevado los cimientos de XML dentro de la aplicación permitiendo a los usuarios editar XML. Esto hace mucho más fácil utilizar Word como una interfaz para un gran número de aplicaciones y sistemas que trabajan con XML.

14.3 Nuevas posibilidades

Mientras que durante años el formato DOC ha sido el estándar de facto para los documentos, se han creado herramientas que permitían extraer y almacenar datos en este formato. Funcionalidades como combinación de correspondencia o conexiones ODBC han hecho posible conectar las aplicaciones Office con otras herramientas, pero la verdadera integración siempre estaba entre aplicaciones del mismo paquete ofimático. Con la llegada de XML, el origen de los datos juega un papel secundario. Mientras la mayoría de información que es tratada por el usuario es creada y manipulada en Office, existe mucha más información disponible además de la manejada por el conjunto de aplicaciones de Microsoft. Hay muchas razones sobre el porqué las empresas están interesadas en mantener su información (inclusive la contenida en sus documentos) almacenada de una forma más conveniente. A pesar que Office tiene métodos para conectarse a diversos sistemas para extraer información, volcar el contenido de una tabla de una base de datos relacional –por ejemplo- no es trivial. Las nuevas funcionalidades XML abren nuevas posibilidades para realizar este tipo de tareas.

La clave está en la creación de un formato XML específico para cada aplicación. Los formatos de documento WordprocessingML (o simplemente WordML) para Word y SpreadsheetML (o SpreadML) para Excel son exclusivos de estos programas, con los cuales pueden trabajar de la misma forma que un documento DOC o XLS respectivamente.

14.3.1 Word: Separar contenido de presentación

Los usuarios usan Word como una herramienta para crear contenido que tenga el aspecto que desean. Mientras que el enfoque de la presentación funciona bien en la mayoría de aplicaciones, este se vuelve un problema cuando se intenta utilizar la interfaz de Word para crear información que deba ser reutilizada.

El soporte que brinda Word para esquemas XML personalizados permite resolver este problema. Pueden crearse documentos en donde se ponga énfasis en su estructura en lugar de su presentación, al mismo tiempo que se utiliza una interfaz que ya es familiar. Además es posible ofrecer al usuario la opción de seleccionar la forma de presentar el contenido, utilizando una vista que sea indicada para él.

14.4 Vocabulario WordprocessingML

Microsoft Word 2003 introduce XML como formato nativo para documentos Word. Cualquier documento Word puede ahora ser abierto y guardado como XML.

WordprocessingML es un formato sin pérdida, lo que significa que contiene toda la información que Word necesita para reabrir el documento, de la misma forma que si hubiese sido

guardado en formato DOC: formatos, estilos, imágenes, macros… Para hacerse una idea de la complejidad que existe detrás de un fichero Word, solo alcanza indicar que el esquema que define este formato tiene unas 7000 líneas en total.

Pero no es necesario conocer todos los elementos del lenguaje WordML para aprovechar sus ventajas, solo aquellos que tengan algún interés especial según la operación a realizar.

Por ejemplo, el siguiente fichero WordML es el documento más sencillo que puede crearse:

```
<?xml version="1.0"?>
<?mso-application progid="Word.Document"?>
<w:wordDocument
xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Texto de prueba.</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:wordDocument>
```

El elemento `mso-application` es una instrucción de procesamiento que indica a Windows que el fichero XML debe ser abierto en Word, y no por otro editor de ficheros XML.

El elemento `w:body` es el único elemento hijo requerido por el elemento raíz `w:wordDocument`. El elemento `w:p` indica la existencia de un párrafo, el elemento `w:r` indica un "run" -concatenación de textos que tienen las mismas propiedades-, y `w:t` representa una porción de texto. El prefijo `w` hace referencia al espacio de nombres del lenguaje WordML: `http://schemas.microsoft.com/office/word/2003/wordml`.

Al abrir el fichero anterior en Word se obtiene el resultado mostrado por la Figura 14-2. En ella el texto es representado con el formato y vista por defecto, ya que esta información no ha sido especificada en el documento.

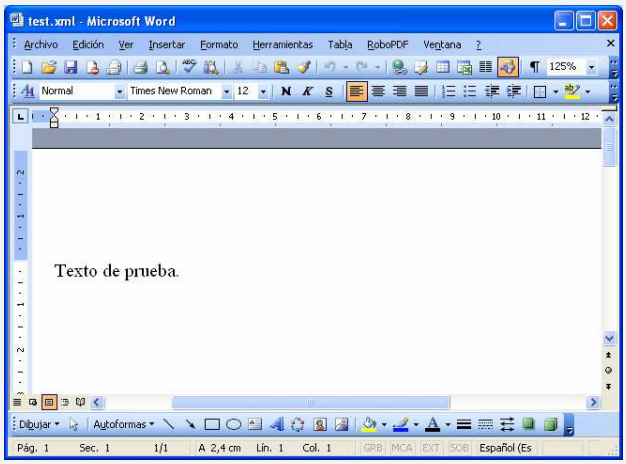


Figura 14-2 Documento Word XML

Todo el texto del documento debe estar contenido dentro de elementos `w:t`, que están contenidos dentro de un elemento `w:r`, que a su vez forman parte de un elemento `w:p`. Notar también que a excepción del elemento `w:wordDocument`, ninguno de los otros elementos del ejemplo tiene atributos. En su lugar, las propiedades son asignadas por medio de subelementos.

Si el documento anterior es modificado de forma mínima (por ejemplo añadiendo un espacio) y guardado, el contenido de su definición variará significativamente, ya que Word añadirá información que no se había especificado y que automáticamente había inferido:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<?mso-application progid="Word.Document"?>
<w:wordDocument
xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml"
xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:sl="http://schemas.microsoft.com/schemaLibrary/2003/core"
xmlns:aml="http://schemas.microsoft.com/aml/2001/core"
xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
w:macrosPresent="no"
w:embeddedObjPresent="no"
w:ocxPresent="no"
xml:space="preserve">
        <o:DocumentProperties>
                <o:LastAuthor>PC</o:LastAuthor>
                <o:Revision>2</o:Revision>
                <o:TotalTime>0</o:TotalTime>
                <o:Created>2007-08-09T20:13:00Z</o:Created>
                <o:LastSaved>2007-08-09T20:13:00Z</o:LastSaved>
                <o:Pages>1</o:Pages>
                <o:Words>2</o:Words>
                <o:Characters>16</o:Characters>
                <o:Lines>1</o:Lines>
                <o:Paragraphs>1</o:Paragraphs>
                <o:CharactersWithSpaces>17</o:CharactersWithSpaces>
                <o:Version>11.6568</o:Version>
        </o:DocumentProperties>
        <w:fonts>
                <w:defaultFonts w:ascii="Times New Roman" w:fareast="Times New Roman"
w:h-ansi="Times New Roman" w:cs="Times New Roman"/>
        </w:fonts>
        <w:styles>
                <w:versionOfBuiltInStylenames w:val="4"/>
                <w:latentStyles w:defLockedState="off" w:latentStyleCount="156"/>
                <w:style w:type="paragraph" w:default="on" w:styleId="Normal">
                        <w:name w:val="Normal"/>
                        <w:rPr>
                                <wx:font wx:val="Times New Roman"/>
                                <w:sz w:val="24"/>
                                <w:sz-cs w:val="24"/>
                                <w:lang w:val="ES" w:fareast="ES" w:bidi="AR-SA"/>
                        </w:rPr>
                </w:style>
                <w:style w:type="character" w:default="on"
w:styleId="Fuentedeprrafopredeter">
                        <w:name w:val="Default Paragraph Font"/>
                        <wx:uiName wx:val="Fuente de párrafo predeter."/>
                        <w:semiHidden/>
                </w:style>
                <w:style w:type="table" w:default="on" w:styleId="Tablanormal">
                        <w:name w:val="Normal Table"/>
                        <wx:uiName wx:val="Tabla normal"/>
                        <w:semiHidden/>
                        <w:rPr>
                                <wx:font wx:val="Times New Roman"/>
                        </w:rPr>
                        <w:tblPr>
                                <w:tblInd w:w="0" w:type="dxa"/>
                                <w:tblCellMar>
                                        <w:top w:w="0" w:type="dxa"/>
                                        <w:left w:w="108" w:type="dxa"/>
```

```
                                        <w:bottom w:w="0" w:type="dxa"/>
                                        <w:right w:w="108" w:type="dxa"/>
                                </w:tblCellMar>
                        </w:tblPr>
                </w:style>
                <w:style w:type="list" w:default="on" w:styleId="Sinlista">
                        <w:name w:val="No List"/>
                        <wx:uiName wx:val="Sin lista"/>
                        <w:semiHidden/>
                </w:style>
        </w:styles>
        <w:docPr>
                <w:view w:val="web"/>
                <w:zoom w:percent="125"/>
                <w:proofState w:grammar="clean"/>
                <w:attachedTemplate w:val=""/>
                <w:defaultTabStop w:val="708"/>
                <w:hyphenationZone w:val="425"/>
                <w:characterSpacingControl w:val="DontCompress"/>
                <w:validateAgainstSchema/>
                <w:saveInvalidXML w:val="off"/>
                <w:ignoreMixedContent w:val="off"/>
                <w:alwaysShowPlaceholderText w:val="off"/>
                <w:compat/>
        </w:docPr>
        <w:body>
                <wx:sect>
                        <w:p>
                                <w:r>
                                        <w:t>Texto de prueba. </w:t>
                                </w:r>
                        </w:p>
                        <w:sectPr>
                                <w:pgSz w:w="11906" w:h="16838"/>
                                <w:pgMar w:top="1417" w:right="1701" w:bottom="1417"
w:left="1701" w:header="708" w:footer="708" w:gutter="0"/>
                                <w:cols w:space="708"/>
                                <w:docGrid w:line-pitch="360"/>
                        </w:sectPr>
                </wx:sect>
        </w:body>
</w:wordDocument>
```

La información añadida corresponde a las propiedades del documento (revisión, número de páginas, caracteres, líneas...), y a la información de secciones, estilos, formatos de texto y formato de página.

Para el desarrollo de este proyecto sólo se han tenido en cuenta los elementos `w:t` ya que ellos almacenan el texto contenido en el documento. El resto de marcas WordML son mantenidas sin realizar modificación alguna al documento de salida.

14.4.1 Espacios de nombres

Todos los documentos WordML presentan en el elemento `w:wordDocument` la definición de todos los espacios de nombres disponibles, independientemente que alguno de sus elementos haya sido utilizado en el documento.

A continuación se lista el conjunto de espacios de nombres junto una breve descripción del objetivo de cada uno de ellos:

`http://schemas.microsoft.com/office/word/2003/wordml`: Asociado al prefijo `w`. Los principales elementos y atributos WordML están en este espacio de nombres.

`urn:schemas-microsoft-com:vml`: Asociado al prefijo `v`. Los elementos de espacio de nombres representas imágenes incrustadas en formato VML (Vector Markup Language)

`urn:schemas-microsoft-com:office:word`: Asociado al prefijo `w10`. Este espacio de nombres es usado para elementos antiguos de la versión 10 de Word.

`http://schemas.microsoft.com/schemaLibrary/2003/core`: Asociado al prefijo `sl`. Los elementos sl:schema y sl:schemaLibrary son usados con los esquemas personalizados de Word.

`http://schemas.microsoft.com/aml/2001/core`: Asociado al prefijo `aml`. Los elementos AML (Annotation Markup Language) son utilizados para describir comentarios, marcadores y control de cambios.

`http://schemas.microsoft.com/office/word/2003/auxHint`: Asociado al prefijo `wx`. Los elementos en este espacio de nombres proporcionan "auxiliary hints" (pistas o indicios auxiliares) para procesar documentos WordLM desde aplicaciones externas. Representan información derivada que no tiene ningún significado para Word.

`urn:schemas-microsoft-com:office:office`: Asignado al espacio de nombres `o`. Este es un espacio de nombres para las propiedades del documento y las propiedades personalizadas.

`uuid:C2F41010-65B3-11d1-A29F-00AA00C14882`: Asociado al prefijo `dt`. Este es el espacio de nombres XDR (XML Data Reduced) utilizado para describir los atributos de los elementos que representan las propiedades personalizadas del documento.

14.5 ODF: Alternativa a WordML

WordML no es el único formato XML para guardar documentos. Una alternativa es el formato OpenDocument u ODF (Open Document Format for Office Applications) basado en el formato XML introducido por la suite ofimática gratuita OpenOffice.org.

ODF cuenta con la categoría de estándar -a diferencia de WordML que simplemente fue definido por el grupo de trabajo de Office-, y contiene definiciones para los distintos tipos de ficheros (hojas de cálculo, presentaciones, bases de datos…) incluyendo –por supuesto- documentos de texto.

Un fichero OpenDocument es un archivo comprimido en formato JAR que incluye otros ficheros organizados en distintos directorios (muy similar al nuevo formato presentado por Microsoft para Office 2007). El lenguaje de marcas utilizado fue diseñado con la idea de cubrir diferentes necesidades, a diferencia de WordML que parece una deserialización de un documento Word.

El principal problema a la hora de utilizar este formato es que Microsoft proporciona un plugin para importar ficheros OpenDocument que presenta grandes limitaciones: no pueden guardarse ficheros ODF, o establecerlo como formato por defecto.

14.6 Conclusión

En este anexo se ha intentado presentar la estructura y elementos básicos presentes en un documento WordML, sin entrar en detalles acerca de la forma en que son representados todos los elementos de Word (tablas, listas…) Estos elementos básicos son los que se han utilizado a lo largo del proyecto para procesar documentos.

Esto demuestra el final del aislamiento de las aplicaciones de escritorio, una época donde estuvieron separadas del resto de aplicaciones debido a los diferentes formatos de almacenamiento y por la idea de que solo deben ser utilizadas para trabajo de oficina. Tanto Microsoft como OpenOffice utilizan XML para comunicar sus aplicaciones con el exterior; y otras compañías están tomando el mismo camino.

Para finalizar se ha presentado una alternativa al formato WordML: ODF.

15 BIBLIOGRAFÍA Y SITIOS DE REFERENCIA

(1) FormsPlayer. *[En línea]* <http://www.formsplayer.com>

(2) Mozilla XForms Project. *[En línea]* <www.mozilla.org/projects/xforms>

(3) Orbeon Forms - Web Forms for the Enterprise *[En línea]*<http://www.orbeon.com>

(4) The Forms Working Group *[En línea]* <http://www.w3.org/MarkUp/Forms/>

(5) Dubinko, Micah. *XForms Essentials.* O'Reilly, 2003.

(6) Raman, T.V. *XForms-XML-powered Web forms*. Addison Wesley, 2003.

(7) Lenz, Evan; McRae, Mary; St. Laurent, Simon. *Office 2003 XML*. O'Reilly, 2004.

(8) Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert. *Head First Design Patterns*. O'Reilly, 2004.

(9) Gross, Christian. *Foundations of Object-Oriented Programming Using .NET 2.0 Patterns*. Apress, 2006.

(10) Ducharme, Bob. *XSLT Quickly*. Manning, 2001.

(11) Harold Rusty, Elliotte. *XML Bible (2nd Edition)*. Wiley, 2001.

(12) Ray, Erik T. *Learning XML, 2nd Edition*. O'Reilly, 2003.

(13) Meyer, Eric. *CSS: The Definitive Guide, Third Edition*. O'Reilly, 2006.

(14) Robinson, Simon. *Professional C# 2005*. Wiley, 2005.

(15) Eisenberg, J. David. *XForms and OpenDocument in OpenOffice.org [En línea]* <http://opendocument.xml.org/files/xforms_ooo_06_10_25.pdf> [Consulta: 01/03/2007]

(16) Udell, Jon. *InfoPath and XForms*. *[En línea]* <http://weblog.infoworld.com/udell/2003/02/26.html> [Consulta: 01/03/2007]

(17) Dubinko, Micah. *XForms and Microsoft InfoPath*. *[En línea]* < http://www.xml.com/lpt/a/1311> [Consulta: 01/03/2007]