
Universidad de las Ciencias Informáticas
Facultad 3

**Título**

# Componente para la generación de formularios web a partir de un esquema XML de la Ventanilla Única del Comercio Exterior

Trabajo de Diploma para optar por el título
de Ingeniero en Ciencias Informáticas

**Autor**

Javier de León Álvarez

**Tutor**

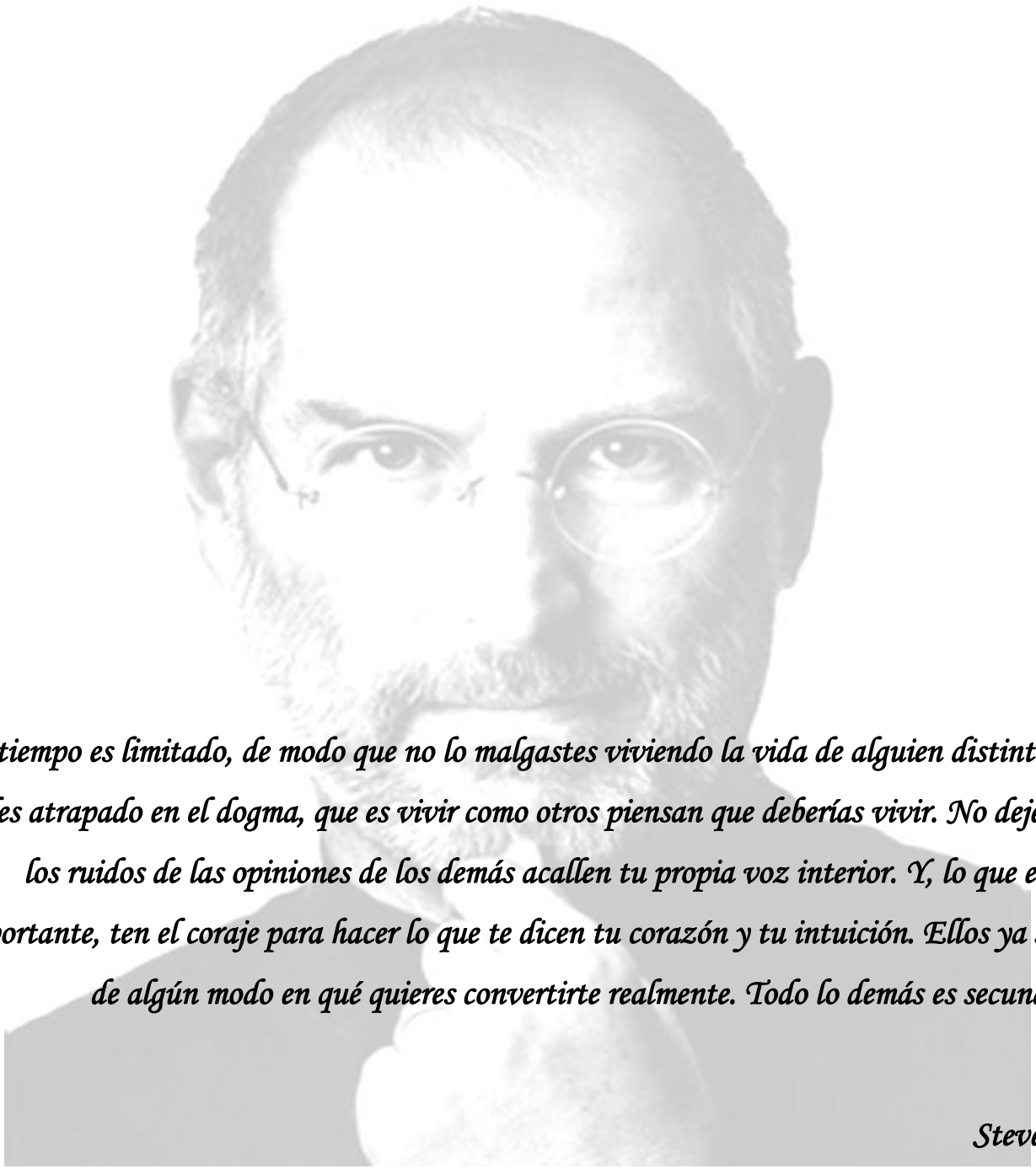
Ing. Liannis Soria Barreda

**Co-Tutor**

Ing. Bernardo Corrales Ambruster
Ing. Eddie Nelson Beltrán González
Ing. Pedro Carlos Abreu Jiménez

Ciudad de la Habana, Junio 2013

## PENSAMIENTO



*Tu tiempo es limitado, de modo que no lo malgastes viviendo la vida de alguien distinto. No quedes atrapado en el dogma, que es vivir como otros piensan que deberías vivir. No dejes que los ruidos de las opiniones de los demás acallen tu propia voz interior. Y, lo que es más importante, ten el coraje para hacer lo que te dicen tu corazón y tu intuición. Ellos ya saben de algún modo en qué quieres convertirte realmente. Todo lo demás es secundario.*

*Steve Jobs*

# AGRADECIMIENTOS

Quiero agradecer a mis padres por todo el amor y apoyo que me han brindado, por dedicar su vida a mi formación. Toda palabra es poca para agradecer tanto sacrificio.

Agradecer a mi abuela Obdulia, que ha dedicado toda su vida primero a mi mamá y después a mí. Gracias abuelita.

Agradecer a toda mi familia que siempre ha estado presente, tanto en los momentos malos como en los buenos. No menciono nombres por miedo a olvidar a alguien. A todos muchas gracias.

A mi novia Yennis, que ha sabido soportarme en toda la carrera, que me ha acompañado en todo momento y que ha sabido sacarme adelante cuando más necesitado he estado. Gracias Amor.

A la familia de Yennis, que me ha aceptado como un miembro más.

A mis más que amigos hermanos Yorjander, Alejandro y Raúl. A sus familias, que sin tener vínculos de sangre me han acogido como un hijo más.

Quiero agradecer también a esa otra familia que hice en la universidad, la gente de mi brigada, de mi proyecto, con la cual he pasado tan buenos momentos. Tampoco menciono nombres por temor a olvidar a alguien. A todos gracias por su amistad.

Agradecimientos muy especiales a mis cotutores Eddie, Bernardo y Charly, son los cuales hoy no estaría hoy aquí diciendo estos agradecimientos.

Agradecer a mi tutora de tesis, que me ha ayudado y me ha enseñado mucho en este tiempo.

Agradecer a todas las personas que de un modo u otro ha estado presente en mi vida y me han ayudado a formarme como la persona que soy hoy.

A la Revolución por permitirme estudiar en una Universidad como esta.

# DEDICATORIA

Dedico este trabajo a mis padres, a mi abuela, a mi novia, a mis suegros, a mis amigos, a todas esas personas que han aportado un granito de arena en mi formación como ser humano.

# RESUMEN

El uso del lenguaje XML para la transmisión de datos a través de aplicaciones está ampliamente difundido en la actualidad, por lo que se hace necesario validar que la información transmitida esté bien estructurada. Para este fin han surgido lenguajes como el esquema XML, el cual en ocasiones es utilizado con otros objetivos. El presente trabajo de diploma tiene como propósito la creación de un componente que permita la generación de un formulario web a partir de un esquema XML, que permitirá al proyecto Ventanilla Única del Comercio Exterior ampliar las vías de recepción de la información proveniente de los clientes. Para guiar el desarrollo del componente se utilizó el modelo de desarrollo del Centro de Informatización de la Gestión de Entidades (CEIGE). Además se utilizaron las herramientas NetBeans como entorno de desarrollo integrado y Visual Paradigm para el modelado de los procesos.

# ÍNDICE

# ÍNDICE DE TABLAS



# ÍNDICE DE FIGURAS

# INTRODUCCIÓN

Internet ha revolucionado la informática y las comunicaciones como ninguna otra cosa. La invención del telégrafo, el teléfono, la radio y el ordenador sentó las bases para esta integración de funcionalidades sin precedentes. Internet es a la vez una herramienta de emisión mundial, un mecanismo para diseminar información y un medio para la colaboración y la interacción entre personas y sus ordenadores, sin tener en cuenta su ubicación geográfica (1).

Con el surgimiento de Internet fueron apareciendo numerosas tecnologías para aprovecharlo, la más importante de todas es la Red Informática Mundial (WWW por sus siglas en inglés, World Wide Web), creada por Tim Berners-Lee, mediante la combinación del hipertexto y el Protocolo de Transferencia de Hipertexto (HTTP por sus siglas en inglés, Hipertext Transfer Protocol). La idea principal de la WWW era desarrollar un método eficiente y rápido para intercambiar información entre la comunidad científica. Años más tardes, en 1992, es creado el Consorcio WWW (W3C por sus siglas en inglés, World Wide Web Consortium).

El W3C es una comunidad internacional donde las organizaciones miembros, personal a tiempo completo y el público en general trabajan conjuntamente para desarrollar estándares Web. Liderado por el inventor de la Web y el Director Ejecutivo Jeffrey Jaffe, la misión del W3C es guiar la Web hacia su máximo potencial (2). Entre los estándares creados por esta comunidad se encuentran el estándar de Diseño y Aplicaciones Web, Arquitectura Web, Web Semántica, Tecnología de Lenguaje de Marcas Extensible (XML por sus siglas en inglés, eXtensible Markup Language), Web de los Servicios, Web de los Dispositivos, además de Navegadores y Herramientas de Autor. Para el desarrollo de este trabajo de investigación, se hará hincapié sobre los estándares de tecnología XML.

Sobre este último, XML es un lenguaje desarrollado por el W3C, deriva del Estándar de Lenguaje de Marcado Generalizado (SGML por sus siglas en inglés, Standard Generalized Markup Language) y permite definir la gramática de lenguajes específicos para estructurar grandes documentos, siendo muy útil en la comunicación entre aplicaciones.

XML es uno de los formatos más utilizados para el intercambio de información estructurada entre programas, entre personas, entre programas y personas, tanto a nivel local como a nivel de redes (3). Es la base de un gran número de normas, como el Lenguaje de Negocios Universal (UBL por sus siglas en inglés, Universal Business Language), es usado también en formatos de procesamiento de textos como Formato de Documentos Abierto (ODF por sus siglas en inglés, Open Document Format) y XML Abierto de Office (OOXML por sus siglas en inglés, Office Open XML), también se utilizan en formatos gráficos como Gráficos Vectoriales Redimensionables (SVG por sus siglas en inglés, Scalable Vector Graphics), en la comunicación con Llamada a Procedimiento Remoto que usa XML (XMLRPC por sus siglas en inglés, Remote Procedure Call XML) y Servicios Web, además tiene soporte en lenguajes de programación y bases de datos. (4).

El lenguaje XML se utiliza principalmente para el intercambio de datos entre aplicaciones, lo que introduce el problema de garantizar que los datos enviados sean correctos. Existen muchos mecanismos para lograr esto, entre ellos Definición de Tipo de Documento (DTD por sus siglas en inglés, Document Type Definition) y el Esquema XML, entre otros. Estos lenguajes permiten imponer restricciones sobre documentos XML, lo que permite comprobar si un documento XML recibido tiene la estructura necesaria y los valores que contiene sean del tipo requerido. Aunque no se diseñaron con este fin, estos lenguajes pueden ser utilizados para crear documentos XML que cumplan con las restricciones que estos últimos especifican, de ahí la importancia de su uso.

Aprovechando las novedosas tecnologías y las posibilidades que estas brindan, las empresas cubanas se han dado la tarea de ganar en eficiencia en la gestión de sus procesos, por lo que muchas de estas han optado por la total o parcial informatización de sus procesos empresariales, mediante la contratación de empresas de desarrollo de software especializadas como DSoft, entre otras.

En este escenario, la Aduana General de la República (AGR) constituye un órgano de control en la frontera y en la actividad interna vinculada al comercio exterior, que garantiza la seguridad y protección de la sociedad socialista y de la economía nacional, así como la recaudación fiscal y las estadísticas del comercio exterior, a través del cumplimiento de las políticas estatales de competencia aduanera para el tráfico internacional de viajeros, mercancías y medios de transporte (5). Esta entidad, en conjunto con la Universidad de las Ciencias Informáticas (UCI) desarrolla el Sistema de Gestión Integral de la Aduana

(GINA), cuyo objetivo principal es informatizar los procesos aduaneros del país, permitiéndole a la Aduana mayor agilidad y facilidad en las operaciones de importación y exportación a través de la simplificación de trámites y procesos.

En la realización de los trámites necesarios para efectuar una operación comercial los involucrados deben acudir a varias instituciones, organismos y entidades solicitando permisos y/o liberaciones que le avalen la operación frente a las autoridades comerciales del país. Para realizar estas gestiones el Operador de Comercio Exterior debe presentarse físicamente en la institución a la que se solicita el aval, provocando la demora, pérdida de tiempo, así como la insatisfacción del cliente. Con el fin de dar solución a estos inconvenientes, en la UCI surge el proyecto Ventanilla Única del Comercio Exterior en Cuba (VUCE), que tiene como objetivo fundamental la integración en un solo punto de todos los trámites comerciales necesarios para la importación y exportación de mercancías al país, garantizando que toda la documentación relacionada con estos trámites sea correctamente procesada y revisada por las personas y entidades designadas para esto, mediante la prestación de servicios integrados de tramitación y asesoramiento empresarial.

Para lograr esto, el proyecto debe proveer a los sistemas externos con los que se conecta, una infraestructura de comunicación, que permita la transferencia de archivos entre el sistema y los ministerios encargados de definir las legislaciones y normas de los documentos que son necesarios entregar para los trámites comerciales con la AGR. Estos ministerios envían al sistema un esquema XML donde está definida la estructura de los documentos XML que les debe facilitar la VUCE y que se procesan en el ministerio en cuestión, estos ministerios, una vez hecho uso de la información, deberán enviar una notificación a la VUCE, la cual lo mostrará al cliente.

Es de interés para el proyecto VUCE brindar la posibilidad a los usuarios de subir un archivo XML al sistema, o brindar un formulario web donde se puedan introducir los datos necesarios de manera directa, pero que estos datos estén estructurados de forma correcta atendiendo al esquema XML para su posterior validación y procesamiento. En estos momentos, cuando la Ventanilla Única del Comercio Exterior recibe un documento, el personal encargado del desarrollo y mantenimiento del proyecto tiene que construir el formulario de forma manual, provocando esto que el tiempo de prestación del servicio del formulario a los

usuarios sea alto, lo que trae como consecuencia algunas inconformidades y molestias a los usuarios, además demora el proceso de recepción de la información.

Por lo antes expuesto, surge el siguiente **problema a resolver**: ¿Cómo generar formularios web a partir de un esquema XML de la Ventanilla Única del Comercio Exterior?, de manera tal que permita reducir el tiempo de prestación del servicio de recepción de información por formulario web.

Siendo el **objeto de estudio** el proceso de generación de formularios web, y se define como **campo de acción** el proceso de generación de formularios web a partir de un esquema XML de la Ventanilla Única del Comercio Exterior.

Para dar solución al problema antes planteado, se define como objetivo general: implementar un componente que permita generar formularios web a partir de un esquema XML de la Ventanilla Única del Comercio Exterior.

Se definen como **objetivos específicos** de la investigación los siguientes:

1. Establecer el marco conceptual de referencia.
2. Establecer el estándar de generación de formularios.
3. Definir el modelo de componentes.
4. Implementar el modelo de componentes.
5. Probar la validez del resultado.

Para dar cumplimiento a los objetivos fueron trazadas las siguientes **tareas**:

1. Recopilación de la bibliografía referente al tema.
2. Selección de la bibliografía.
3. Análisis de la bibliografía.
4. Análisis de las variantes de estandarización de generación de formularios.
5. Formalización de la variante de estandarización.
6. Definición de un marco arquitectónico.
7. Modelación del diseño de la aplicación.

8. Implementación de las funciones asociadas al componente.
9. Validación del resultado obtenido.
10. Integración del componente a la arquitectura base del sistema VUCE.
11. Exposición de los resultados.

Los métodos científicos ayudan a los investigadores en la obtención y análisis de la información necesaria para desarrollar sus trabajos. Estos métodos se clasifican en dos grupos: métodos teóricos y métodos empíricos. Los métodos teóricos posibilitan a los investigadores descubrir en el objeto de investigación, las relaciones esenciales y las cualidades fundamentales, no detectables de manera senso-perceptual. Por ello se apoya básicamente en los procesos de abstracción, análisis, síntesis, inducción y deducción. A su vez, los métodos empíricos posibilitan revelar las relaciones esenciales y las características fundamentales del objeto de estudio, accesibles a la detección de la percepción, a través de procedimientos prácticos con el objeto y diversos medios de estudio. A continuación se detallan algunos de los métodos utilizados en la investigación.

**Métodos Teóricos:**

**Análisis-Síntesis:** Consiste en descomponer un objeto, fenómeno o proceso en los principales elementos que lo integran para analizar, valorar y conocer sus particularidades. En el presente trabajo, se utiliza para el entendimiento de los conceptos fundamentales relacionados con la investigación, como XML, Esquemas XML, componente de software, entre otros.

**Histórico-Lógico:** A través de este método se establece la necesaria correspondencia entre los elementos de los métodos lógico e histórico, proyectando el análisis de la evolución histórica de los fenómenos, con la proyección lógica de su comportamiento futuro. Se utiliza para estudiar la evolución en el tiempo de XML y de los mecanismos de validación de documentos XML.

**Métodos Empíricos:**

**Observación:** Es un método para reunir información visual sobre lo que ocurre, lo que el objeto de estudio hace o cómo se comporta. Es utilizado para reunir información sobre los procesos del proyecto Ventanilla Única del Comercio Exterior.

**Análisis de documentos:** Este método se utiliza para el análisis de los documentos relacionados con las metodologías a utilizar en la investigación y los métodos a implementar.

El presente trabajo de diploma posee la siguiente estructura:

En el primer capítulo se analizan los conceptos y definiciones asociados al dominio del problema que sirven de apoyo durante el desarrollo de la investigación. Como aspecto fundamental, se ofrece un enfoque de los términos relacionados con el XML y con los Esquemas XML, además se detallan las tecnologías a utilizar para dar cumplimiento al objetivo general.

En el segundo capítulo se tratan los temas relacionados con el análisis y el diseño de la solución propuesta. Se detallan los requisitos funcionales y los requisitos no funcionales. Por la parte del diseño se muestran los diagramas de clases, de interacción, de paquetes, así como los patrones de diseño utilizados.

En el tercer capítulo se aborda sobre los aspectos relacionados con la implementación del componente, así como los relacionados con las pruebas realizadas al mismo. Se muestra el diagrama de componentes del sistema, así como elementos referentes a nomenclaturas de código utilizadas. Para la validación del componente fueron aplicadas técnicas de caja blanca y de caja negra. Se realiza además una valoración sobre el cumplimiento de los objetivos propuestos al inicio de la investigación.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## INTRODUCCIÓN

El presente capítulo tiene como objetivo el estudio de los principales contenidos teóricos que sustentan la investigación a desarrollar. De manera general se tratan conceptos ligados al ámbito de la investigación, se documentan y fundamentan las herramientas, tecnologías y lenguajes usados para dar cumplimiento al objetivo general de la investigación, también se realiza un estudio del uso de los documentos XML en el mundo y en Cuba.

## 1.1 CONCEPTOS FUNDAMENTALES

### *1.1.1 XML*

El W3C define al XML como un formato de texto flexible derivado de SGML. Originalmente diseñado para afrontar los retos de la gran edición electrónica, está desempeñando un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y otros lugares (6).

La recomendación de la W3C: El Lenguaje Extensible de Marcas (XML), plantea que XML describe una clase de objetos de datos llamados documentos XML y parcialmente describe el comportamiento de los programas de computadora que los procesan. Los documentos XML se componen de unidades de almacenamiento llamadas entidades, que contienen datos analizados o no analizados. Los datos analizados se componen de caracteres, algunos son datos de caracteres, y algunos son de marcado. Los datos de marcado codifican una descripción del diseño de almacenamiento del documento y su estructura lógica. XML proporciona un mecanismo para imponer restricciones en el diseño de almacenamiento y estructura lógica (7).

Cada documento XML tiene una estructura lógica y una estructura física. Físicamente, el documento se compone de unidades llamadas entidades. Una entidad puede referirse a otras entidades para causar su inclusión en el documento. Un documento comienza en una "raíz" o entidad documento. Lógicamente, el documento está compuesto de declaraciones, elementos, comentarios, referencias de caracteres, e instrucciones de procesamiento, todos los cuales se indican en el documento por marcado explícito (7).

La propia especificación del W3C plantea que los objetivos del diseño de XML son (7):

- XML debe ser utilizable directamente sobre internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil escribir programas que procesen documentos XML.
- El número de características opcionales en XML debe ser mantenido en un mínimo, idealmente cero.
- Los documentos XML deben ser legibles por un humano y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fáciles de crear.
- La brevedad en la marcación es de mínima importancia.

XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información. Estas partes se llaman entidades, y se les representan mediante etiquetas.

Una etiqueta consiste en una parte del documento, que señala una porción de este como una entidad, es un pedazo de información con un sentido claro y bien definido. Las etiquetas tienen la forma `<nombre>`, donde nombre es el nombre de la entidad que se está señalando. Cada etiqueta del tipo `<nombre>` tiene su correspondiente etiqueta de cierre `</nombre>`, que indica que hasta ahí llego la declaración de los datos de tipo “nombre”.

Los documentos XML denominados bien formados son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico que cumpla con la norma.

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus entidades. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas.
- Los documentos XML sólo permiten una entidad raíz del que todos los demás sean parte, es decir, solo pueden tener una entidad inicial.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos «entendibles» por las personas.

Como ventajas del XML se pueden mencionar:

- Es extensible, ya que una vez diseñado y puesto en producción, es posible agregar nuevas etiquetas, de modo que se pueda continuar utilizando sin complicaciones.
- El analizador es un componente estándar, no es necesario implementar uno nuevo para cada versión de XML, ya que se puede usar alguno de los ya existentes.
- Es sencillo entender su estructura y procesarla.
- Posibilidad de comunicar aplicaciones sin importar el origen de los datos.

Para ver un ejemplo de un documento XML bien formado, ir al Anexo #1: Ejemplo de Documento XML.

### *1.1.2 Componente de software*

Diversos autores han emitido su criterio acerca de la definición de componente de software, aunque muchas de estas tienen muchos puntos en común, es difícil llegar a una única definición del término. A continuación se citan algunas de estas definiciones:

Kai Koskimies (8) define componente como una entidad independiente del sistema binario que implementa una o más interfaces. A su vez, una interfaz es una colección de firmas de servicios lógicamente juntos.

Clemens Szyperski (9) lo define como una unidad de composición con interfaces especificadas contractualmente y dependencias de contexto explícitas únicamente. Un componente de software puede ser desplegado de forma independiente y está sujeto a la composición por terceras partes.

A su vez, Michael Stal (8) proyecta el componente como una unidad binaria que exporta e importa funcionalidades utilizando un mecanismo de interfaz normalizada. La infraestructura subyacente soporta composición de componentes mediante mecanismo de introspección, gestión de eventos, persistencia, vinculación dinámica y gestión de esquemas.

Para la investigación en cuestión, luego del estudio y análisis de las definiciones anteriores, se asume como concepto de componente de software al conjunto de funciones organizadas de forma lógica que realizan una tarea específica. Este conjunto es autónomo, no tiene dependencias con otros y además utiliza interfaces específicas para brindar o utilizar funcionalidades de terceros.

Algunas de las características de un componente de software según Ian Sommerville (10) son.

| **Características del componente** | **Descripción** |
|---|---|
| Estandarizado | El componente debe de ajustarse a algún modelo estandarizado de componentes. Este modelo puede definir interfaces de componentes, metadatos de componentes, documentación, composición y despliegue. |
| Independiente | Un componente debería ser independiente, debería ser posible componerlo y desplegarlo sin tener que usar otros componentes específicos. Si el componente necesitas servicios externos, estos deben de hacer implícitos en una especificación de interfaz. |
| Componible | Todas las interacciones externas deben tener lugar a través de interfaces definidas públicamente. Además, debe proporcionar acceso externo a la información sobre sí mismo. |

| | |
|---|---|
| Desplegable | Debe ser independiente y debe ser capaz de funcionar como una entidad autónoma o sobre una plataforma de componentes que implemente el modelo de componentes. |
| Documentado | Tienen que estar completamente documentados para que los usuarios potenciales puedan decidir si satisface o no sus necesidades. |

**Tabla 1: Características de un componente**

## 1.2 VALIDACIÓN DE DOCUMENTOS XML

XML resuelve el problema del envío de información de manera estructurada a través de aplicaciones, pero en la mayoría de las veces es necesario validar que la información transmitida cumpla con ciertos requerimientos de estructura, de tipos de datos, etc.

El propósito de la validación de archivos XML es definir la estructura de los documentos XML y los tipos de datos válidos para cada elemento y atributo. Al restringir el contenido de los ficheros XML es posible el intercambio de información entre aplicaciones con gran seguridad, además que disminuye el trabajo de comprobar la estructura del fichero y los tipos de datos.

Existen muchos mecanismos para validar documentos XML, entre los que se encuentran el DTD, el Esquema XML, Datos XML Reducidos (XDR por sus siglas en inglés, XML Data Reduced), entre otros. A continuación se realiza un estudio acerca de las características y ventajas que los anteriormente mencionados brindan.

### *1.2.1 DTD*

Según W3Schools (11), el propósito de DTD es definir los componentes básicos permitidos en un XML. DTD define la estructura de un documento XML mediante una lista de elementos y atributos permitidos.

DTD cumple las siguientes funciones (12):

- DTD especifica la clase de documento:

  - Describe un formato de datos.
  - Usa un formato común de datos entre aplicaciones.
  - Verifica los datos al intercambiarlos.
  - Verifica un mismo conjunto de datos.
- DTD describe:
  - Elementos: cuáles son las etiquetas permitidas y cuál es el contenido de cada etiqueta.
  - Estructura: en qué orden van las etiquetas en el documento.
  - Anidamiento: qué etiquetas van dentro de cuáles.

Los documentos DTD no siguen una sintaxis XML, al contrario, posee un lenguaje propio de escritura, lo que puede ocasionar problemas a la hora del aprendizaje. Otra de sus limitaciones es que DTD indica sólo qué elementos y atributos tiene un documento, pero no dice nada acerca de los tipos de datos. El único tipo de dato que conoce es el texto plano. Otra limitante es que DTD no soporta el uso de los espacios de nombre, tampoco tiene soporte para incluir o importar fragmentos de esquema en archivos exteriores. Estas limitantes hacen que su uso en la presente investigación no sea factible.

### *1.2.2 XDR*

En primer lugar conocido como XML-Data, luego XDR, este lenguaje es un esfuerzo conjunto de Microsoft y otros, y se está utilizando en el marco de Microsoft BizTalk. XDR está fuertemente influenciado por otra propuesta de IBM[1]y Microsoft, DCD[2], por lo que comparten muchas características similares (13).

Según Microsoft (14), XDR es un formato XML que especifica la estructura de otros documentos XML. Consiste en los siguientes elementos:

- **ElementType**: especifica las etiquetas de los elementos permitidos en el documento válido, así como su contenido, que puede incluir texto y otros elementos.
- **AttributeType**: especifica los atributos válidos, si son necesarios, tipos de datos y valores por defecto en el documento.

[1] International Business Machines
[2] Descripción de contenido de documento

- **element**: especifica el contexto en el que los elementos de ElementType pueden ocurrir.
- **attribute**: especifica los atributos de AttributeType que pueden tener los elementos de cierto ElementType determinado.
- **group**: refiere sobre la estructura del documento, en referencia a las definiciones ElementType.

Aunque XDR también permite el uso de espacios de nombre, no soporta la inclusión o importación de fragmentos de esquemas en ficheros externos y tiene soporte de forma parcial para restricciones de dominio sobre valores posibles en los atributos, además que no soporta tipos de datos, la ausencia en el lenguaje de los elementos antes mencionados provoca que no sea posible su uso en la presente investigación.

### *1.2.3 Esquemas XML*

Algunos autores definen al Esquema XML como un esfuerzo continuo de W3C para ayudar y eventualmente sustituir DTD en el mundo XML. El Esquema XML pretende ser más expresivo que el DTD y más fácil de usar por una variedad más amplia de aplicaciones. Tiene muchos mecanismos novedosos, como la herencia de atributos y elementos, además de tipos de datos definidos por el usuario (13).

Según la Especificación del W3C XML Schema, los esquemas expresan vocabularios compartidos que permiten a las máquinas extraer las reglas hechas por las personas. Los esquemas proveen un significado para definir la estructura, contenido y semántica de los documentos XML (6).

Al igual que el XDR, los esquemas XML siguen una sintaxis XML, lo que hace posible el uso de los diferentes analizadores existentes, facilitando así el trabajo con los mimos.

Los esquemas XML soportan el uso de los espacios de nombre (namespaces, término en inglés). Estos pueden definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento. Además, soportan restricciones sobre el dominio de valores de los atributos y también tiene soporte para la inclusión e importación de ficheros externos y soportan gran cantidad de tipos de datos. Los esquemas XML son los que mayor información brindan sobre el XML que validan, lo que hace posible la obtención de la información necesaria para la generación de formularios

que respondan a las restricciones que estos imponen, esto hace que los esquemas XML sean los idóneos para el desarrollo de la investigación.

## 1.3 TECNOLOGÍAS, LENGUAJES Y HERRAMIENTAS A UTILIZAR

Actualmente existen numerosas herramientas, tecnologías y metodologías que ayudan al proceso de desarrollo de software. Existen gran variedad de entornos de desarrollo integrado que simplifican la construcción de aplicaciones, herramientas cases que facilitan el desarrollo de software mediante el diseño y la gestión de los diferentes artefactos generados durante el desarrollo de un producto. Además existen numerosas metodologías que guían el proceso de construcción de un software garantizando la calidad del mismo. En los siguientes epígrafes se detallan las características de las herramientas y metodologías definidas a usar por el Centro de Informatización de la Gestión de Entidades (CEIGE), particularmente en el proyecto Ventanilla Única del Comercio Exterior.

### *1.3.1 Modelo de Desarrollo del CEIGE*

El modelo de desarrollo de software del Centro de Informatización de la Gestión de Entidades (CEIGE) contiene las especificaciones de las actividades de cada una de las fases del ciclo de vida de los proyectos pertenecientes a este centro teniendo en cuenta los procesos de CMMI (Capability Maturity Model Integration) nivel 2 para la Universidad de las Ciencias Informáticas. La figura siguiente define las fases definidas en el modelo de desarrollo del CEIGE.

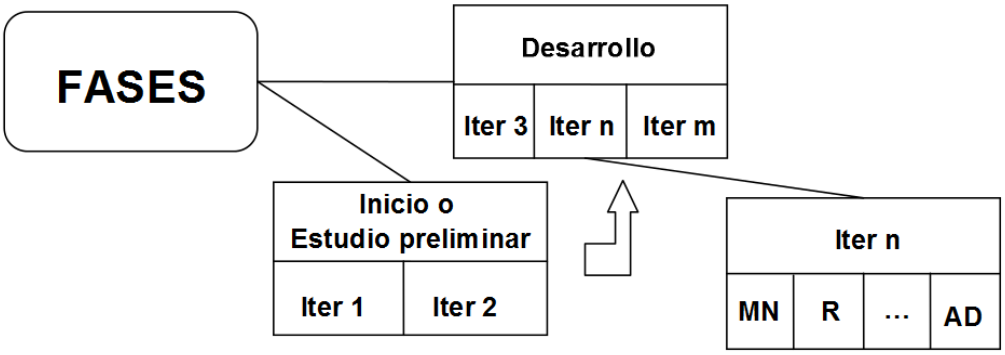



**Figura 1: Fases del ciclo de vida de proyectos del CEIGE (15)**

Durante el Inicio o Estudio Preliminar se llevan a cabo todas las actividades relacionadas con la planeación del proyecto a un alto nivel, la evaluación de la factibilidad del proyecto y el registro de este. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo, y decidir si se ejecuta o no el proyecto.

En la fase de Desarrollo se ejecutan las fases requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se refinan los requisitos, se elaboran la arquitectura y diseño, se implementa y se libera el producto (15).

La fase de desarrollo tiene como objetivo obtener un sistema que satisfaga las necesidades del cliente y usuarios finales. Como hito se tiene un producto liberado por la entidad certificadora de calidad. En estas fases de ejecutan las disciplinas Modelado de Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas Internas y Pruebas de Liberación.

A continuación se detallan los artefactos a generar durante el desarrollo de la presente investigación.

| Artefacto | Descripción |
|---|---|
| Modelo conceptual. | Explica cuáles son y cómo se relacionan los conceptos relevantes del problema. |
| Especificación de requisitos de software. | Describe los requisitos de software del componente. |
| Modelo de diseño. | Describe los elementos del diseño, incluye Diagrama de Clases y Diagrama de Componentes. |
| Diagrama de componente. | Describe la relación entre el componente y los sistemas externos. |
| Diseño de casos de prueba. | Describe los diseños de casos de prueba realizados. |
| Manual de Usuario. | Describe los pasos necesarios para utilizar el componente. |

**Tabla 2: Relación de artefactos a generar**

En los siguientes epígrafes se describen las herramientas, tecnologías y lenguajes a utilizar en el desarrollo de la investigación.

### *1.3.2 NetBeans*

El NetBeans es un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés, integrated development environment) disponible para Windows, Mac, Linux y Solaris. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicaciones que permite a los desarrolladores crear rápidamente aplicaciones web, para empresas, de escritorio y para móviles utilizando la plataforma Java, así como PHP, JavaScript, Ajax, Groovy y Grails, C y C++ (16).

Entre sus principales características se destacan (16):

- Edición rápida e inteligente de código.
- Gestión de proyectos de manera fácil y eficiente.
- Desarrollo rápido de interfaces de usuario.
- Escritura de código libre de errores.
- Soporte para múltiples lenguajes.
- Soporte para varias plataformas y sistemas operativos.
- Amplio conjunto de complementos.

### *1.3.3 UML*

Lenguaje Unificado de Modelado (UML por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para describir un plano del software, incluye aspectos conceptuales tales como procesos de negocios, funciones del sistema, expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables (17) (18). En concreto ayuda a comprender y a mantener de una mejor forma un sistema basado en un área que el analista o desarrollador puede desconocer.

La decisión de utilizar UML como notación para el desarrollo del software se debe a que se ha convertido en un estándar que tiene las siguientes características:

- Permite modelar sistemas haciendo uso de técnicas orientadas a objetos (OO).

- Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- Es independiente del proceso, aunque para utilizarlo óptimamente se debería emplear en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

### *1.3.4 Visual Paradigm*

Visual Paradigm es una de las herramientas UML presentes en el mercado hoy en día, considerada muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Fue creada para el ciclo completo del desarrollo del software, permitiendo la captura de requisitos, análisis, diseño e implementación. Además apoya los estándares más recientes de las notaciones de JAVA y UML e incorpora soporte para el trabajo en equipo.

Entre sus principales características de pueden citar:

- Producto de calidad.
- Soporte para aplicaciones web y para varios idiomas.
- Fácil de instalar y actualizar.
- Posibilidad de integración con las siguientes herramientas JAVA.
    - Eclipse.
    - Jbuilder.
    - NetBeans IDE.

Ventajas:

- Apoyo en cuanto a artefactos generados durante las etapas de definición de requerimientos y especificación de componentes.
- Generación de documentación en formatos HTML y PDF[3].
- Disponibilidad en múltiples plataformas (Microsoft Windows, Linux, Mac OS[4]).
- Posibilidad de intercambio de información con aplicaciones como Visio y Rational Rose.
- Generación de código e ingeniería inversa: posibilidad de generar código a partir de los diagramas y obtener diagramas a partir del código.
- Posibilidad de generación de documentación sin la necesidad de herramientas externas.

### *1.3.5 PHP*

PHP es un lenguaje de programación interpretado de propósito general. Es ampliamente utilizado en el desarrollo web y puede ser embebido en páginas HTML (19).

Entre las funciones más destacadas que presenta se pueden citar (20):

- Soporte para una gran cantidad de bases de datos: MySQL, PostgreSQL, Oracle, MSSQL Server, SybasemSQL e Informix.
- Integración con varias bibliotecas externas, permite generar documentos en PDF y analizar código XML.
- Perceptiblemente más fácil de mantener y poner al día que el código desarrollado en otros lenguajes.
- Soportado por una gran comunidad de desarrolladores, como producto de código abierto, PHP goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se encuentren y reparen rápidamente.
- El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP.

[3] Formato de documento portable (*portable document format en inglés*)
[4] Sistema Operativo (Operating Symtem en inglés)

### *1.3.6 Servidor Web Apache*

Un servidor web es un programa que sirve para atender y responder a las diferentes peticiones de los navegadores, proporcionando los recursos que soliciten usando el protocolo HTTP o el protocolo HTTPS (la versión cifrada y autenticada).

En la actualidad es el más usado, encontrándose por encima de sus competidores, tanto gratuitos como comerciales. Es un software de código abierto que funciona sobre cualquier plataforma.

Está estructurado en módulos, es decir, está dividido en muchas porciones de código que hacen referencia a diferentes aspectos o funcionalidades del servidor Web.

Algunas funciones de Apache se consiguen por medio de módulos adicionales que se pueden cargar. Para añadir un conjunto de utilidades al servidor, simplemente hay que añadirle un módulo, de forma que no es necesario volver a instalar el software.

Ventajas (21):

- Modular.
- Código Abierto.
- Multiplataforma.
- Extensible.
- Fácil de conseguir ayuda y soporte.

## CONCLUSIONES DEL CAPÍTULO

Luego del estudio de los principales conceptos y la elaboración del marco teórico de la investigación se pudo concluir:

- Se utilizará el esquema XML como base para la generación de los formularios.
- Se utilizará el modelo de desarrollo del Centro de la Informatización de la Gestión de Entidades y las herramientas definidas por el proyecto Ventanilla Única del Comercio Exterior.

# CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA SOLUCIÓN PROPUESTA

## INTRODUCCIÓN

En el presente capítulo se dan a conocer las principales características de la solución propuesta. Se realiza una breve descripción de las características del componente a construir, se describen los requisitos funcionales y no funcionales identificados. Se detallan algunos elementos del análisis y diseño que se consideran necesarios para un mejor entendimiento del sistema a construir.

## 2.1 PROPUESTA DE SOLUCIÓN

La solución que se propone para dar respuesta al problema rector de este trabajo es la construcción de un componente de software que se encargue de transformar un esquema XML a un formulario web. Este componente se integrará a la arquitectura del proyecto Ventanilla Única del Comercio Exterior, para el cual brindará la funcionalidad antes mencionada. Se realizará la construcción del componente mediante la introspección en las etiquetas del esquema XML, construyendo clases que serán la representación de estas etiquetas. Estas clases serán utilizadas para la construcción de una lista de tipos presentes en el esquema. Los tipos se componen por un nombre igual al que tienen en el esquema XML y la clase que contiene su información, llamada nodo.

Después de haber obtenido los elementos antes mencionados, se procede a ir revisando el listado de los tipos e ir construyendo clases que representarán a los elementos del formulario, entiéndase campos de entradas (inputs) y listados de elementos (grids). Un input es un campo sencillo de texto y un grid está compuesto por uno o más input, lo que significa que un grid es un listado donde cada elemento de la lista es un conjunto de datos.

Una vez realizado el análisis de la propuesta de solución se proceden a identificar los requisitos de software que el componente deberá poseer.

## 2.2 REQUISITOS DE SOFTWARE

Según Sommerville (10), un requisito de software es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. La correcta o incorrecta identificación de los mismos puede converger a un producto bien elaborado y que satisface las necesidades del cliente o a un software prácticamente inútil.

Según Roger S. Pressman (22), la comprensión de los requisitos de un problema está entre las tareas más difíciles que enfrenta un ingeniero de software, de ahí la importancia para el desarrollo de un proyecto que los requisitos identificados sean los que verdaderamente responden a las necesidades de los clientes. Debido a esto, es común que se utilicen técnicas para la captura de requisitos, lo que posibilita una mejor obtención de los mismos. Las técnicas utilizadas para la obtención de los requisitos en la presente investigación se listan a continuación:

Entrevista: se realizaron entrevistas con el equipo de desarrollo del proyecto Ventanilla Única del Comercio Exterior con el objetivo de identificar las necesidades de las funcionalidades del componente a desarrollar.

Tormenta de Ideas: conocido también como brainstorming, consistió en reuniones con el equipo de desarrollo del proyecto, donde se generaron diversas ideas que contribuyeron a la identificación de los requisitos que serán detallados más adelante.

En los siguientes epígrafes se brinda una descripción de los requisitos identificados con que debe cumplir el componente de generación de formularios web.

### *2.2.1 Requisitos Funcionales*

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar. Los mismos se centran en *qué* y no *cómo* se deben hacer esas funciones. A continuación se exponen los requisitos funcionales identificados.

RF 1: Construir formulario web a partir de esquema XML.

RF 1.1: Parsear esquema XML.

RF 1.2: Construir los elementos del formulario.

RF 1.3: Generar código HMTL a partir de los elementos del formulario.

Seguidamente se muestra una breve descripción de los requisitos funcionales identificados.

| Número | Requisito Funcional | Descripción |
|---|---|---|
| 1.1 | Parsear esquema XML | Inspeccionar las etiquetas del esquema XML y construir el listado de tipos y el árbol de clases. |
| 1.2 | Construir los elementos del formulario | Construir las clases que representan los elementos del formulario a partir del listado de tipos. |
| 1.3 | Generar código HMTL a partir de los elementos del formulario | A partir de los elementos del formulario obtenidos anteriormente generar el código HTML del formulario. |

**Tabla 3: Breve descripción de los requisitos funcionales**

### *2.2.2 DESCRIPCIÓN DE LOS REQUISITOS*

2.2.2.1 Descripción RF Parsear Esquema XML.

| | |
|---|---|
| **Precondiciones** | Se ha recibido un esquema XML bien formado. |
| **Flujo de eventos** | |
| **Flujo básico Parsear Esquema XML** | |
| 1 | El sistema selecciona una etiqueta del esquema XML y revisa la información de la misma. |
| 2 | El sistema obtiene la información necesaria para la ejecución del requisito funcional: “Construir los elementos del formulario”:<br>• Nombre de la etiqueta.<br>• El valor del atributo “name” de la etiqueta.<br>• El valor del atributo “type” de la etiqueta.<br>• El valor del atributo “minOccurs” de la etiqueta.<br>• El valor del atributo “maxOccurs” de la etiqueta. |
| 3 | Se crea un Nodo para que pueda representar la información de dicha etiqueta. |
| 4 | En caso de existir el atributo “name” ver flujo alterno *4.a Atributo name.* |

| | |
|---|---|
| | En caso de existir el atributo “type” ver flujo alterno *4.b Atributo type.*<br>En caso de existir el atributo “minOccurs” ver flujo alterno *4.c Atributo minOccurs.*<br>En caso de existir el atributo “maxOccurs” ver flujo alterno *4.a Atributo maxOccurs.* |
| 5 | Ejecutar el paso 1 del flujo básico como tantas etiquetas exista en el esquema XML. |
| 6 | Concluye requisito. |
| **Pos-condiciones** | |
| 1 | Devuelve los types. |
| 2 | Devuelve el listado de los nodos. |
| **Flujos alternativos** | |
| **Flujo alternativo 4.a Atributo name** | |
| 1 | El sistema crea un type con la información obtenida de la etiqueta. |
| 2 | El sistema agrega el valor del atributo “name” al Nodo. |
| 3 | Ejecutar el paso 6 del flujo básico. |
| **Pos-condiciones** | |
| 1 | Se agrega el valor del atributo “name” al Nodo. |
| **Flujo alternativo 4.b Atributo type** | |
| 1 | El sistema agrega el valor del atributo “type” al Nodo. |
| 2 | Ejecutar el paso 6 del flujo básico. |
| **Pos-condiciones** | |
| 1 | Se agrega el valor del atributo “type” al Nodo. |
| **Flujo alternativo 4.c Atributo minOccurs** | |
| 1 | El sistema agrega el valor del atributo “minOccurs” al Nodo. |
| 2 | Ejecutar el paso 6 del flujo básico. |
| **Pos-condiciones** | |
| 1 | Se agrega el valor del atributo “minOccurs” al Nodo |
| **Flujo alternativo 4.d Atributo maxOccurs** | |
| 1 | El sistema agrega el valor del atributo “maxOccurs” al Nodo. |
| 2 | Ejecutar el paso 6 del flujo básico. |
| **Pos-condiciones** | |
| 1 | Se agrega el valor del atributo “maxOccurs” al Nodo. |
| **Validaciones** | |
| 1 | N/A |
| **Requisitose speciales** | N/A |
| **Asuntos pendientes** | N/A |

**Tabla 4: Descripción RF Parsear Esquema XML**

2.2.2.2 Descripción RF Construir los elementos del formulario.

| | |
|---|---|
| **Precondiciones** | Se tiene el listado de types y el listado de nodos de partida. |
| **Flujo de eventos** | |
| **Flujo básico Construir los elementos del formulario** | |
| 1 | El sistema selecciona un nodo del listado y verifica si representa en una lista de elementos.<br>En caso de representar un listado ver flujo alternativo *1.a Listado de elementos.*<br>En caso de no representar un listado ver flujo alternativo *1.b Elemento.* |
| 2 | Concluye requisite. |
| **Pos-condiciones** | |
| 1 | Listado de elementos. |
| 2 | Listado de listas de elementos. |
| **Flujos alternativos** | |
| **Flujo alternativo 1.a Listado de elementos** | |
| 1 | El sistema verifica que el atributo “type” sea de tipo base (tipo de dato simple: String, date, integer, boolean, entre otros).<br>En caso no ser de tipo base, ver flujo alternativo *1.a.1.a No tipo base listado.* |
| 2 | Se guarda la información en el listado de listas de elementos. |
| 3 | Ejecutar paso 2 del flujo básico. |
| **Pos-condiciones** | |
| 1 | Guardar la información en el listado de listas de elementos. |
| **Flujo alternativo 1.a.1.a No tipo base listado** | |
| 1 | Buscar el type en el listado de types. |
| 2 | Ejecutar el paso 2 del flujo alternativo *1.a Listado de elementos.* |
| **Pos-condiciones** | |
| 1 | Ejecutar el paso 2 del flujo alternativo *1.a Listado de elementos* |
| **Flujo alternativo 1.b Elemento** | |
| 1 | El sistema verifica que el atributo “type” sea de tipo base (tipo de dato simple: String, date, integer, boolean, entre otros).<br>En caso no ser de tipo base, ver flujo alternativo *1.b.1.a No tipo base elemento.* |
| 2 | Se guarda la información en el listado de elementos. |
| 3 | Ejecutar paso 2 del flujo básico. |
| **Pos-condiciones** | |
| 1 | N/A |

| | |
|---|---|
| **Flujo alternativo 1.a.1.a No tipo base elemento** | |
| 1 | Buscar el type en el listado de types. |
| 2 | Ejecutar el paso 2 del flujo alternativo *1.b Elemento.* |
| **Pos-condiciones** | |
| 1 | Ejecutar el paso 2 del flujo alternativo *1.b Elemento.* |
| **Validaciones** | |
| 1 | N/A |
| **Requisitos especiales** | N/A |
| **Asuntos pendientes** | N/A |

**Tabla 5: Descripción RF Construir los elementos del formulario**

2.2.2.3 Descripción RF Generar código HMTL a partir de los elementos del formulario.

| | |
|---|---|
| **Precondiciones** | Se tiene el listado de elementos y el listado de listas de elementos. |
| **Flujo de eventos** | |
| **Flujo básico Construir los elementos del formulario** | |
| 1 | El sistema selecciona una lista de elementos del listado de listas de elementos. |
| 2 | El sistema genera el código HTML a partir de la lista de elementos seleccionadas. |
| 3 | Ejecutar el paso 1 del flujo básico cuantas veces sean necesarias. |
| 4 | El sistema selecciona un elemento de la lista de elementos. |
| 5 | El sistema genera el código HTML a partir del elemento seleccionado. |
| 6 | Ejecutar el paso 4 del flujo básico cuantas veces sean necesarias. |
| 7 | Concluye requisito. |
| **Pos-condiciones** | |
| 1 | Se genera el código HTML. |
| **Validaciones** | |
| 1 | N/A |
| **Requisitos especiales** | N/A |
| **Asuntos pendientes** | N/A |

**Tabla 6: Generar código HTML a partir de los elementos del formulario**

### *2.2.3 VALIDACIÓN DE LOS REQUISITOS*

Para la validación de los requisitos antes descritos se procedió a la revisión técnica de los mismos con el analista principal del proyecto Ventanilla Única del Comercio Exterior, el cual constató que los mismos están correctamente descritos y no presentan ambigüedades.

### *2.2.4 REQUISITOS NO FUNCIONALES*

Los requisitos no funcionales imponen restricciones en el diseño y la implementación del software a desarrollar. Son propiedades o cualidades que el producto debe poseer (18).

Teniendo en cuenta que el resultado de esta investigación formará parte del proyecto Ventanilla Única del Comercio Exterior, los requisitos no funcionales se acogen a los identificados por el equipo del proyecto en la fase inicial del desarrollo del proyecto. Estos requisitos son listados a continuación.

#### *2.2.4.1 RNF Rendimiento*

El tiempo de generación del formulario web en los archivos debe de ser lo más rápido posible, no debiendo exceder de los 5 segundos una vez que se haya recibido el esquema XML.

#### *2.2.4.2 RNF Seguridad*

Confiabilidad: El sistema debe prevenir posibles fallos y/o errores y recuperarse ante ellos. La información manejada por el sistema está protegida de acceso no autorizado.

Integridad: La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes.

Disponibilidad: Los usuarios autorizados tendrán garantizado el acceso a la información en todo momento independientemente de los mecanismos utilizados para manipular los datos.

#### *2.2.4.3 RNF Software*

Para el cliente:

- Navegador Mozilla Firefox 7.0 o superior o Chrome 1.0 o superior.
- Sistema operativo Windows 98 o superior o Linux.

Para el servidor:

- Sistema operativo Linux en cualquiera de sus distribuciones.
- Un servidor Apache 2.0 o superior.
- Lenguaje PHP en su versión 5.3.6 o superior.

#### *2.2.4.4 RNF Hardware*

Para el servidor:

- Requerimientos mínimos: Procesador Core 2 Duo a 2.0GHz de velocidad de procesamiento y 1GB de memoria RAM.
- Tarjeta de red.

Para el cliente:

- Requerimientos mínimos: Procesador Pentium IV a 2.0GHz con 512MB de memoria RAM.
- Tarjeta de red.

#### *2.2.4.5 RNF Soporte*

- El componente debe de ser extensible.
- El componente debe ser fácil de mantener.

Luego de describir los requisitos de software que estarán presentes en la solución propuesta, se pasa a describir el flujo de datos entre las partes que compondrán la solución.

## 2.3 MODELO DE DISEÑO

La fase del diseño en el proceso de construcción de software expande y detalla los modelos de análisis tomando en cuenta todas las implicaciones y restricciones técnicas. El propósito del diseño es especificar una solución que trabaje y pueda ser fácilmente convertida en código fuente y construir una arquitectura simple y fácilmente extensible (23). En los epígrafes siguientes se detallarán los elementos del diseño tenidos en cuenta para la elaboración de la solución propuesta.

### *2.3.1 Diagrama de Clases del diseño*

Este diagrama es usado para mostrar cómo se realizará la implementación de las funciones de gestión y de control del software, que permiten que el sistema opere y se comunique dentro de su entorno y con el exterior. Además es usado para exponer los atributos y métodos de las clases (22). A continuación se muestra el diagrama de clases correspondiente al componente de generación de formularios.

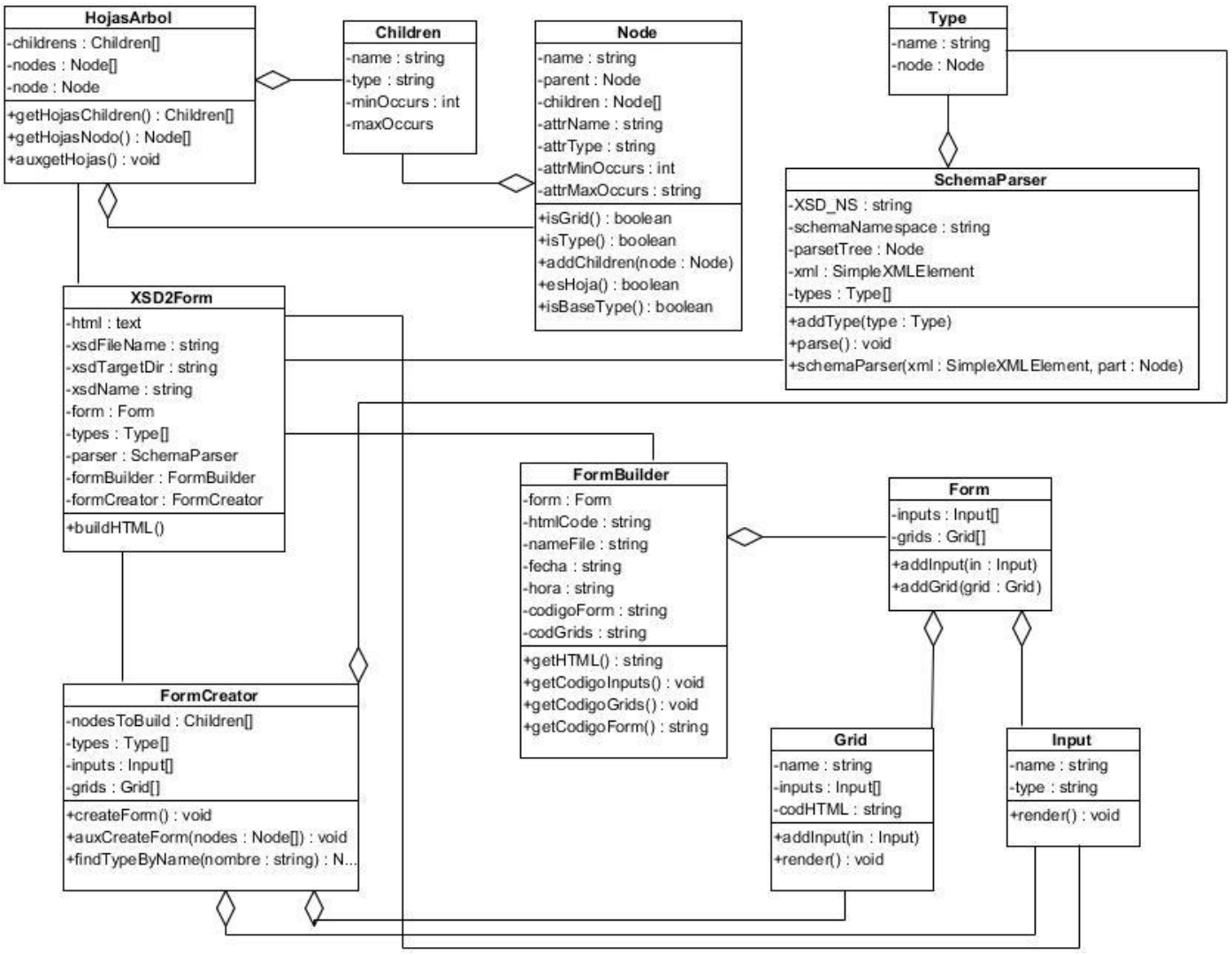



**Figura 2: Diagrama de Clases**

Descripción de las clases del diseño:

| **Nombre:** Node | |
|---|---|
| **Atributo** | **Tipo** |
| name | string |
| parent | Node |
| children | Node[] |
| attrName | string |

| | |
|---|---|
| attrType | string |
| attrMinOccurs | integer |
| attrMaxOccurs | string |
| **Métodos**: | |
| Nombre: | isGrid |
| Descripción: | Devuelve verdadero o falso en caso de que el nodo se pueda representar como un grid. |
| Nombre: | isType |
| Descripción: | Devuelve verdadero o falso en caso de que el nodo representa un tipo de dato. |
| Nombre: | addChildren |
| Descripción: | Adiciona una hija al nodo actual. La hija se le pasa como parámetro. |
| Nombre: | esHoja |
| Descripción: | Devuelve verdadero o falso en caso de que el nodo no tenga hijas. |
| Nombre: | isBaseType |
| Descripción: | Devuelve verdadero o falso en caso de que el atributo attrType coincida con los tipos de datos básicos (string, date, integer). |

**Tabla 7: Descripción de la clase Node**

| **Nombre:** Type | |
|---|---|
| **Atributo** | **Tipo** |
| name | string |
| node | Node |

**Tabla 8: Descripción de la clase Type**

| **Nombre:** Children | |
|---|---|
| **Atributo** | **Tipo** |
| name | string |
| type | string |
| minOccurs | integer |
| maxOccurs | string |

**Tabla 9: Descripción de la clase Children**

| **Nombre:** HojasArbol | |
|---|---|
| **Atributo** | **Tipo** |
| childrens | Children[] |
| nodes | Node[] |
| node | Node |
| **Métodos**: | |
| Nombre: | getHojasChildren() |
| Descripción: | Devuelve un listado con las hojas del atributo node en forma de objetos de tipo Children. |
| Nombre: | getHojasNodo() |
| Descripción: | Devuelve un listado con las hojas del atributo node en forma de objetos de tipo Node. |

| | |
|---|---|
| Nombre: | auxgetHojas() |
| Descripción: | Método recursivo encargado de conformar los listados de las hojas del nodo. |

**Tabla 10: Descripción de la clase HojasArbol**

| **Nombre:** SchemaParser | |
|---|---|
| **Atributo** | **Tipo** |
| XSD_NS | string |
| schemaNamespace | string |
| parseTree | Node |
| xml | SimpleXMLElement |
| types | Type[] |
| **Métodos**: | |
| Nombre: | addType() |
| Descripción: | Adiciona un objeto de tipo Type al listado de tipos |
| Nombre: | parse() |
| Descripción: | Devuelve un listado con las hojas del atributo node en forma de objetos de tipo Node |
| Nombre: | schemaParser() |
| Descripción: | Método recursivo encargado de convertir las etiquetas XML a clases y conformar el listado de tipos. |

**Tabla 11: Descripción de la clase SchemaParser**

| **Nombre:** FormCreator | |
|---|---|
| **Atributo** | **Tipo** |
| nodesToBuild | Children[] |
| types | Type[] |
| inputs | Input[] |
| grids | Grid[] |
| **Métodos**: | |
| Nombre: | createForm() |
| Descripción: | Crea los elementos del formulario a partir del listado de tipos. |
| Nombre: | auxCreateForm() |
| Descripción: | Contiene la implementación del algoritmo de creación de los elementos del formulario. |
| Nombre: | findTypeByName() |
| Descripción: | Busca si un nombre está contenido en el listado de tipos. |

**Tabla 12: Descripción de la clase FormCreator**

| **Nombre:** FormBuilder | |
|---|---|
| **Atributo** | **Tipo** |
| form | Form |
| htmlCode | string |
| javascriptCode | string |

| | |
|---|---|
| nameFile | string |
| fecha | string |
| hora | string |
| codigoForm | string |
| codigoGrids | string |
| **Métodos**: | |
| Nombre: | getHTML() |
| Descripción: | Devuelve el código del formulario. |
| Nombre: | getCodigoInputs() |
| Descripción: | Construye el código HTML de los inputs. |
| Nombre: | getCodigoGrids() |
| Descripción: | Construye el código JavaScript y HTML de los grids. |
| Nombre: | getCodigoForm() |
| Descripción: | Construye el código JavaScript y HTML del formulario |

**Tabla 13: Descripción de la clase FormBuilder**

| **Nombre:** Form | |
|---|---|
| **Atributo** | **Tipo** |
| inputs | Input[] |
| grids | Grid[] |
| **Métodos**: | |
| Nombre: | addInput() |
| Descripción: | Adiciona un objeto de tipo Input al listado |
| Nombre: | addGrid() |
| Descripción: | Adiciona un objeto de tipo Grid al listado |

**Tabla 14: Descripción de la clase Form**

| **Nombre:** Grid | |
|---|---|
| **Atributo** | **Tipo** |
| name | string |
| inputs | Grid[] |
| codHTML | string |
| codJS | string |
| **Métodos**: | |
| Nombre: | addInput() |
| Descripción: | Adiciona un objeto de tipo Input al listado |
| Nombre: | render() |
| Descripción: | Construye el código HTML y JavaScript del grid. |

**Tabla 15: Descripción de la clase Grid**

| **Nombre:** Input |
|---|

| **Atributo** | **Tipo** |
|---|---|
| name | string |
| type | string |
| **Métodos**: | |
| Nombre: | render() |
| Descripción: | Construye el código HTML y JavaScript del input. |

**Tabla 16: Descripción de la clase Input**

| **Nombre:** XSD2Form | |
|---|---|
| **Atributo** | **Tipo** |
| html | string |
| xsdFileName | string |
| form | Form |
| xsdTargetDir | string |
| xsdName | string |
| types | Type[] |
| parser | SchemaParser |
| formBuilder | FormBuilder |
| formCreator | FormCreator |
| **Métodos**: | |
| Nombre: | buildHTML() |
| Descripción: | Se encarga de realizar todas las fases del proceso de construcción del formulario a partir de un esquema XML. |

**Tabla 17: Descripción de la clase XSD2Form**

Realizada la descripción de las clases presentes en el diseño de la solución propuesta, se procede con la descripción de los demás artefactos generados en el diseño del componente.

### *2.3.2 Diagrama de Paquetes*

El diagrama de paquetes es utilizado para categorizar diferentes elementos del modelo de análisis (por ejemplo, casos de usos, clases del análisis) (22). Este diagrama muestra cómo un sistema está dividido en agrupaciones lógicas, las relaciones entre estas agrupaciones y suministra una descomposición de la jerarquía lógica del sistema.

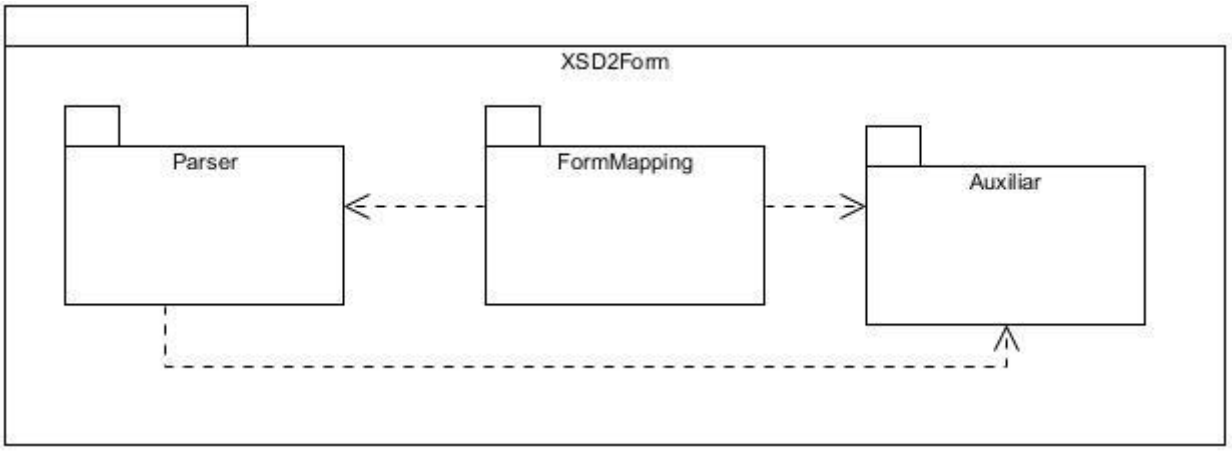



**Figura 3: Diagrama de paquetes**

A continuación se muestra el diagrama de paquetes con las clases pertenecientes a cada paquete.

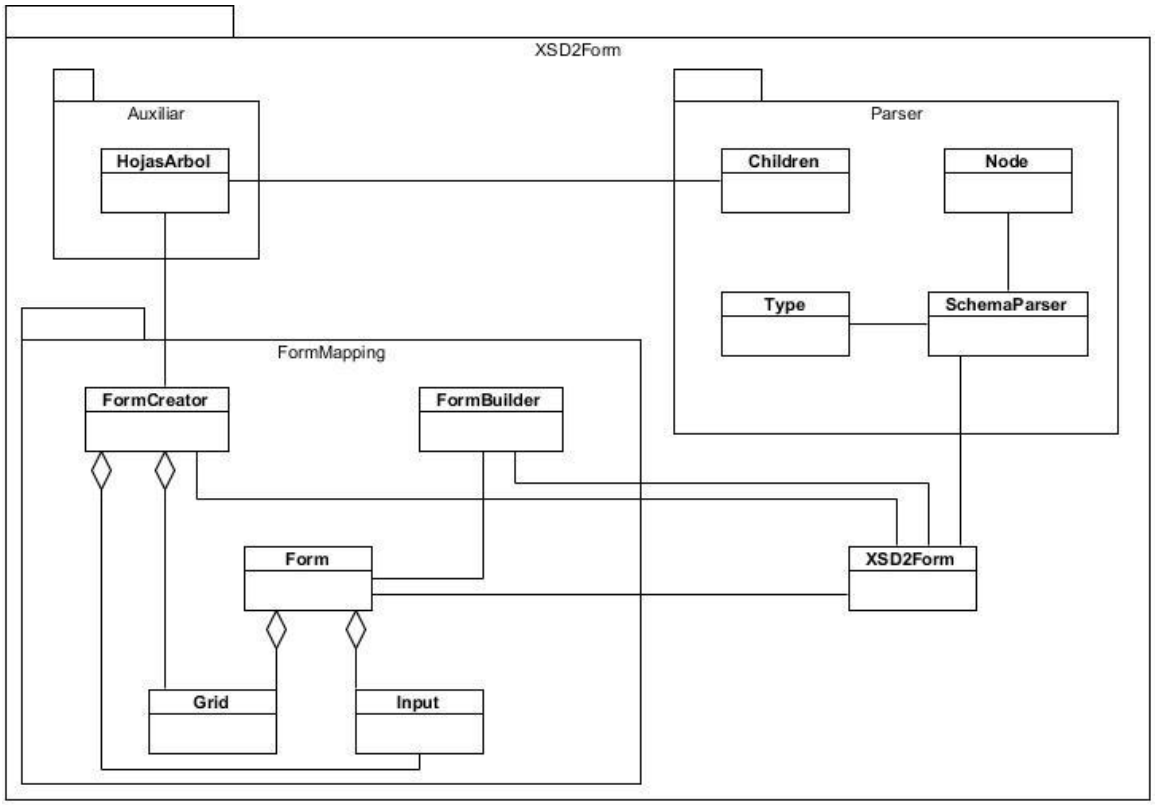



**Figura 4: Diagrama de paquetes con sus clases**

### *2.3.3 PATRONES UTILIZADOS*

Los patrones de diseño son utilizados para resolver problemas comunes del desarrollo del software, entre sus características es que deben de haber probado su efectividad en diferentes ocasiones y ser reutilizables, lo que significa que debe ser aplicable en diferentes circunstancias (24). El uso de patrones en la construcción de un producto de software es siempre una buena práctica, por lo que se tuvo en cuenta para la construcción del componente. A continuación se muestra el patrón utilizado para la construcción de la solución.

Fachada (Facade): Está motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando la comunicación y dependencias entre ellos.

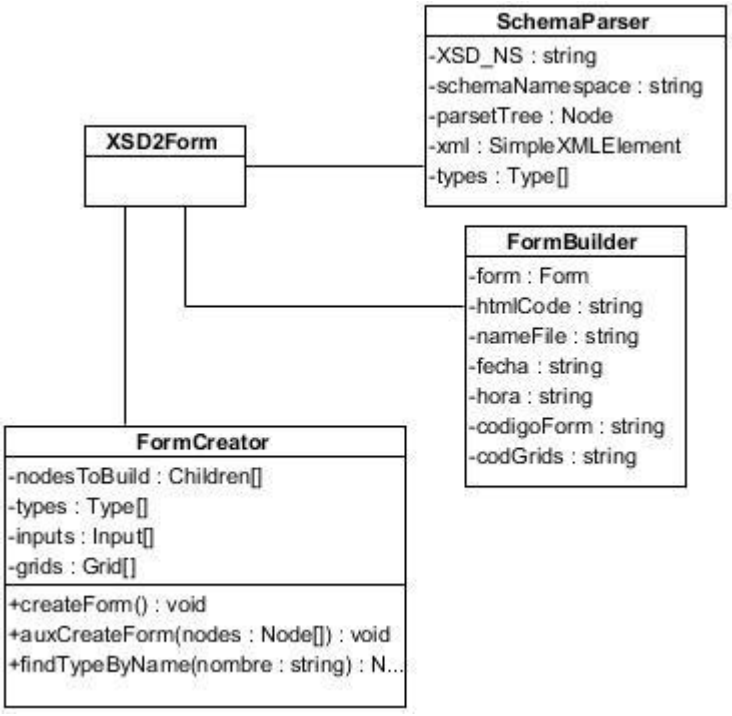



**Figura 5: Ejemplo de patrón Fachada**

En este caso el patrón fachada se evidencia en que la clase XSD2Form tiene conocimiento de los demás subsistemas que se encargan de realizar tareas específicas, y estas clases no conocen a la fachada.

## 2.4 VALIDACIÓN DEL DISEÑO PROPUESTO

Para la validación del diseño propuesto se utiliza la métrica Tamaño Operacional de Clases (TOC). Esta métrica utiliza la cantidad de operaciones que realiza una determinada clase, a esta cantidad se le denomina tamaño operacional. Hace énfasis en tres atributos (Responsabilidad, Complejidad de Implementación y Reutilización) y la forma en que estos son afectados por el tamaño operacional. A continuación se muestran una serie de elementos para ayudar a la comprensión de dicha métrica.

| Atributo que afecta | Modo en que lo afecta |
|---|---|
| Responsabilidad | Un aumento del TOC implica un aumento de la responsabilidad de la clase. |
| Complejidad | Un aumento del TOC implica un aumento de la complejidad de la clase. |
| Reutilización | Un aumento del TOC implica una disminución en el grado de reutilización de la clase. |

**Tabla 18: Relación entre atributo y tamaño operacional de clase**

Para la aplicación de esta métrica se calcula el promedio de operaciones de las clases, el cual es usado para asignar una categoría a la clase en cada uno de los atributos mencionados anteriormente, atendiendo a los rangos mostrados en la siguiente tabla.

| Atributo | Categoría | Criterio |
|---|---|---|
| Responsabilidad | Baja | <= promedio |
| | Media | Entre promedio y 2*promedio |
| | Alta | > 2*promedio |
| Complejidad | Baja | <= promedio |
| | Media | Entre promedio y 2*promedio |
| | Alta | > 2*promedio |
| Reutilización | Baja | > 2*promedio |
| | Media | Entre promedio y 2*promedio |
| | Alta | <= promedio |

**Tabla 19: Criterios de categorización de atributos**

A continuación se muestran las clases del diseño a las cuales se les aplicó la métrica y los resultados obtenidos para cada atributo. Para determinar el valor de los atributos se calcula el promedio de la columna *Cantidad de Procedimientos* y este promedio (en este caso es de 2.27 aproximadamente) es el empleado para aplicar el criterio de evaluación del atributo.

| No | Subsistema | Clase | Cantidad de Procedimientos | Responsabilidad | Complejidad | Reutilización |
|---|---|---|---|---|---|---|
| 1 | XSD2Form | HojasArbol | 3 | Media | Media | Media |
| 2 | XSD2Form | Children | 0 | Baja | Baja | Alta |
| 3 | XSD2Form | Node | 5 | Alta | Alta | Baja |
| 4 | XSD2Form | Type | 0 | Baja | Baja | Alta |
| 5 | XSD2Form | SchemaParser | 3 | Media | Media | Media |
| 6 | XSD2Form | FormCreator | 3 | Media | Media | Media |
| 7 | XSD2Form | FormBuilder | 6 | Alta | Alta | Baja |
| 8 | XSD2Form | Form | 2 | Baja | Baja | Alta |
| 9 | XSD2Form | Grid | 2 | Baja | Baja | Alta |
| 10 | XSD2Form | Input | 1 | Baja | Baja | Alta |
| 11 | XSD2Form | XSD2Form | 0 | Baja | Baja | Alta |

**Tabla 20: Listado de clases con los valores de atributo**

A continuación se muestran de manera gráfica los resultados obtenidos.

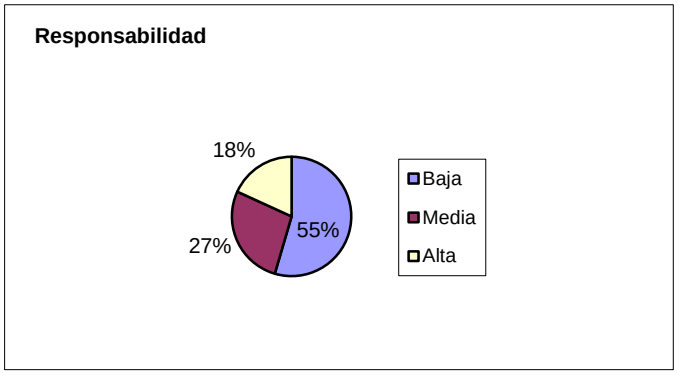



**Figura 6: Gráfico del atributo Responsabilidad**

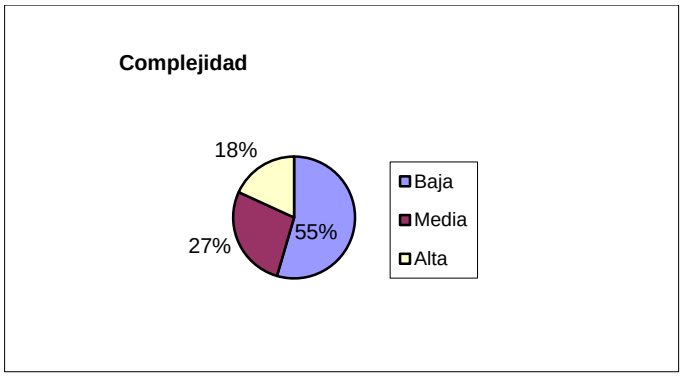



**Figura 7: Gráfico del atributo Complejidad**

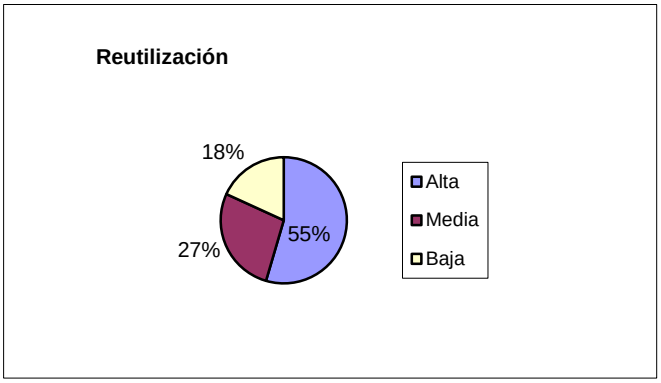



**Figura 8: Gráfico del atributo Reutilización**

Mediante el análisis de los resultados obtenidos de la métrica TOC en la evaluación del diseño propuesto, se puede observar que la mayoría de clases (82%) para los atributos responsabilidad y complejidad están dentro de la categoría Media y Baja, mientras que ese mismo por ciento para el atributo reutilización se encuentran dentro de la categoría Alta y Media, esto indica que el diseño propuesto tiene una alta reutilización y una baja complejidad y responsabilidad. La alta reutilización del diseño es un aspecto importante, debido a que la solución a construir tiene que ser lo más reutilizable posible, debido a las características que posee. Según la métrica TOC y de acuerdo a los criterios de ponderación de los atributos, una alta reutilización tiene el mismo criterio que una baja responsabilidad. Teniendo en cuenta esto se decide maximizar la reutilización del diseño, aunque esto implique una baja responsabilidad.

## CONCLUSIONES DEL CAPÍTULO

Mediante el análisis y diseño de la solución se generaron los diagramas y artefactos que permiten una mejor comprensión del software que se quiere construir, además se realizó la validación del diseño propuesto, por lo que se concluye:

- El análisis de las características del componente permitió sentar las bases para la implementación del componente.
- Se obtuvieron los requisitos funcionales y no funcionales que deberá cumplir el componente.
- Mediante la aplicación se la métrica Tamaño Operacional de Clases se validó el diseño propuesto atendiendo a la complejidad, reutilización y responsabilidad de las clases.

# CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

## INTRODUCCIÓN

En el presente capítulo se muestra como fue implementada la solución propuesta a partir del proceso de análisis y diseño anterior. Se presenta el modelo de implementación, haciendo énfasis en el diagrama de componentes. También se fundamentan las validaciones aplicadas al componente de generación de formularios web.

## 3.1 DIAGRAMA DE COMPONENTES

Un diagrama de componentes contiene generalmente componentes, interfaces y las relaciones que se establecen entre ellos. Además puede tener paquetes con el objetivo de agrupar los elementos del modelo. Además enseña la forma en que quedan organizados los componentes y sus dependencias lógicas, estos pueden ser ejecutables, binarios o código fuente (25).

A continuación se presenta el diagrama de componentes:

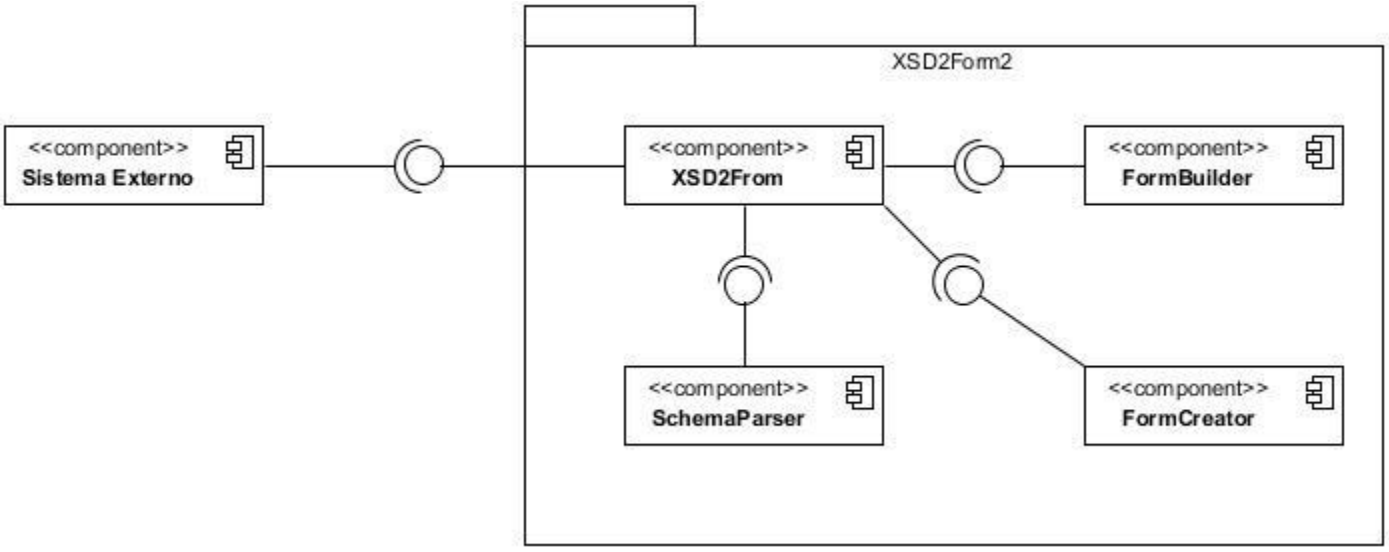



**Figura 9: Diagrama de componentes**

## 3.2 CÓDIGO FUENTE

Escribir un código legible y fácil de entender es de vital importancia para el desarrollo y posterior mantenimiento de un proyecto de software, para esto suelen usarse algunos estilos de escrituras que brindan pautas a la hora de escribir un determinado código. En el siguiente epígrafe se muestran algunas de las convenciones utilizadas para escribir el código fuente del componente.

### *3.2.1 NOMENCLATURA DE CÓDIGO UTILIZADA*

Para el nombre de las variables se utilizará la nomenclatura lowerCamelCase, la cual indica que la primera palabra se escribe en minúsculas, en caso de haber más de una palabra estas comienza con mayúsculas y no se separan de la anterior. El nombre de las variables puede contener cualquier carácter exceptuando a los siguientes:

- Corchetes angulares (< >).
- Dos puntos (:).
- Asterisco (*).
- Barra diagonal (/) y barra diagonal inversa (\).
- Signo de interrogación (?).
- Marca de comillas dobles (").
- Barra vertical (|).

Ejemplo de nombre de variable:

```
$inputs = array();
$schemaParser = new SchemaParser($XSDFileName);
$schemaParser->parse();
$typesList = $schemaParser->getTypes();
$hojasArbol = new HojasArbol($typesList[0]->getNode());
```

**Figura 10: Ejemplo de nombre de variables en la clase XSD2Form**

Para el nombre de las clases se utilizará la nomenclatura UpperCamelCase, que indica que la primera letra de cada palabra se escribe en mayúsculas.

Ejemplo de nombre de clases:

```
class HojasArbol {

    private $childrens = array();
    private $nodes = array();
    private $node;

    function __construct(Node $node) {
        $this->node = $node;
    }
```

**Figura 11: Ejemplo de nombre de la clase HojasArbol**

## 3.3 PRUEBAS

Las pruebas de software son una actividad en la cual el sistema es ejecutado bajo condiciones específicas, para demostrar que tiene o no la madurez necesaria para ser implantado. Permiten determinar la calidad del producto, detectar todo posible mal funcionamiento y comprobar el grado de cumplimiento de las especificaciones iniciales del sistema. Se considera una prueba exitosa si se demuestra deficiencias en el software. Las pruebas no pueden asegurar la ausencia de defectos; solo pueden demostrar que existen defectos en el software (22).

### *3.3.1 PRUEBAS DE CAJA BLANCA O ESTRUCTURALES.*

Las pruebas de caja blanca, denominada a veces prueba de caja de cristal, es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, se pueden obtener casos de prueba que:

- Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.

- Ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas.
- Ejerciten todos los bucles en y con sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez (22).

Para la realización de las pruebas de caja blanca se selecciona la técnica de Camino Básico. Para aplicar esta técnica se debe introducir la notación para la representación del flujo de control, este puede representarse por un Grafo de Flujo en el cual:

- Cada nodo del grafo se corresponde a una o más sentencias de código fuente.
- Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.
- Se calcula la complejidad ciclomática del grafo.

Las instrucciones se representan de la siguiente manera:

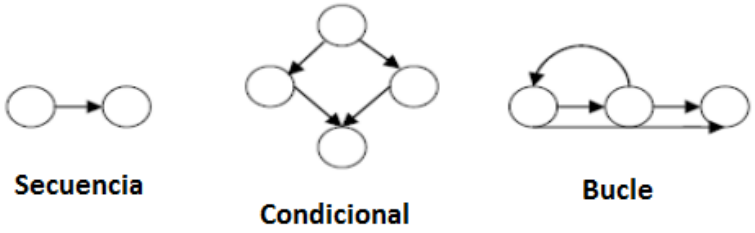



**Figura 12: Representación de las instrucciones Sequencia, Condicional y Bucle**

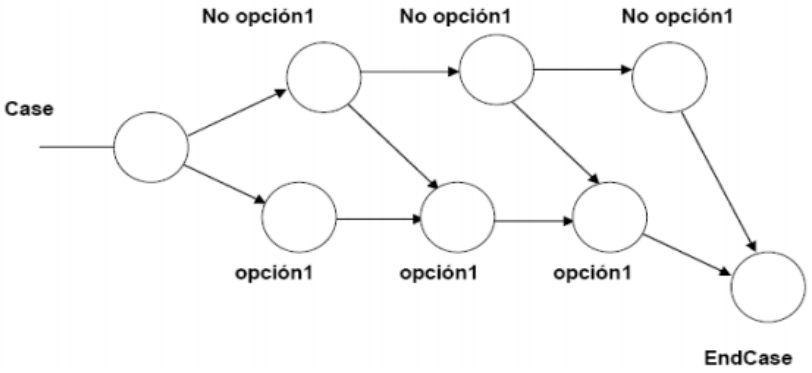



**Figura 13: Representación de la instrucción Case**

Un grafo de flujo está conformado por tres componentes fundamentales:

**Nodo**: Son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y al final del grafo.

**Aristas**: Son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control de procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa una sentencia del procedimiento.

**Regiones**: Son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

Para realizar la técnica del camino básico es necesario calcular la complejidad ciclomática del algoritmo o fragmento de código a analizar, pues esta proporciona una medida cuantitativa de la complejidad lógica de un programa. A continuación se presenta un ejemplo de la prueba del camino básico aplicada a un fragmento de código que tiene como función la construcción de los objetos del formulario a partir de un listado de nodos del esquema XML.

```
function createForm() {
    foreach ($this->nodesToBuild as $value) {    1
        if ($this->findTypeByName($value->getType())) {    2
            $node = $this->findTypeByName($value->getType());    3
            $hojasNodo = new HojasArbol($node);    3
            $this->auxCreateForm($hojasNodo->getHojasNodos());    3
        } else {    4
            $in = new Input($value->getName(), 'string');    5
            $this->inputs[] =$in;    5
        }
    }
}
```

**Figura 14: Ejemplo de técnica de camino básico a la funcionalidad createForm()**

A continuación se muestra el grafo de flujo asociado al código anterior.

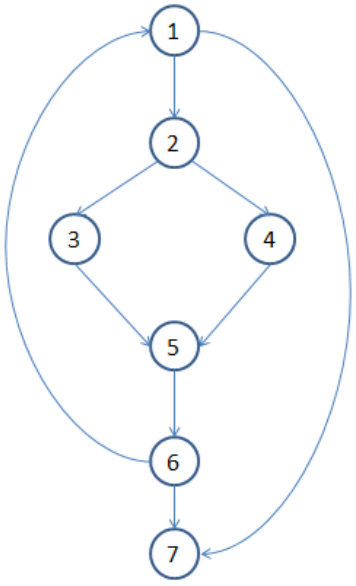



**Figura 15: Grafo de flujo de la funcionalidad createForm()**

Cálculo de la complejidad ciclomática a partir de un segmento de código.

El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y define un límite superior para el número de pruebas que se deben de realizar, para asegurar que se ejecute cada sentencia al menos una vez. Para efectuar el cálculo de la complejidad ciclomática del código es necesario tener varios parámetros como son, la cantidad total de aristas del grafo y la cantidad total de nodos, que se emplean en la siguiente fórmula.

V (G) = (Aristas – Nodos) + 2
V (G) = (9 - 7) + 2
V (G) = 4

También se utiliza la siguiente fórmula:

V (G) = Nodos predicados + 1
V (G) = 3 + 1
V (G) = 4

Los nodos predicados son aquellos desde los que salen más de una arista.

La otra fórmula para realizar el cálculo de la complejidad ciclomática del grafo es la siguiente:

V (G) = Cantidad de áreas del grafo incluyendo la externa.
V (G) = 4

El resultado del cálculo de la complejidad ciclomática utilizando las tres fórmulas anteriores para el fragmento de código usado como ejemplo fue 4, lo que indica que existen 4 posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

| Número | Camino Básico |
|---|---|
| 1 | 1-2-3-5-6-7 |
| 2 | 1-2-3-5-6-1-7 |
| 3 | 1-2-4-5-6-7 |

| | |
|---|---|
| 4 | 1-2-4-5-6-1-7 |

**Tabla 21: Ejemplos de caminos básicos del grafo de flujo anterior**

Resultados de las pruebas de caja blanca.

Luego de la aplicación de las pruebas de caja blanca al componente construido se obtuvo como resultado que del 100% aproximadamente del 87% de las pruebas dieron como resultado satisfactorio y el 13% restante fallidas. Para la segunda iteración de pruebas y luego de haber corregidos los errores detectados en la fase anterior el porciento de pruebas satisfactorias alcanzo el 96%, dejando solamente un 4% de pruebas fallidas. Para la tercera iteración el 100% de las pruebas aplicadas arrojó resultado satisfactorio.

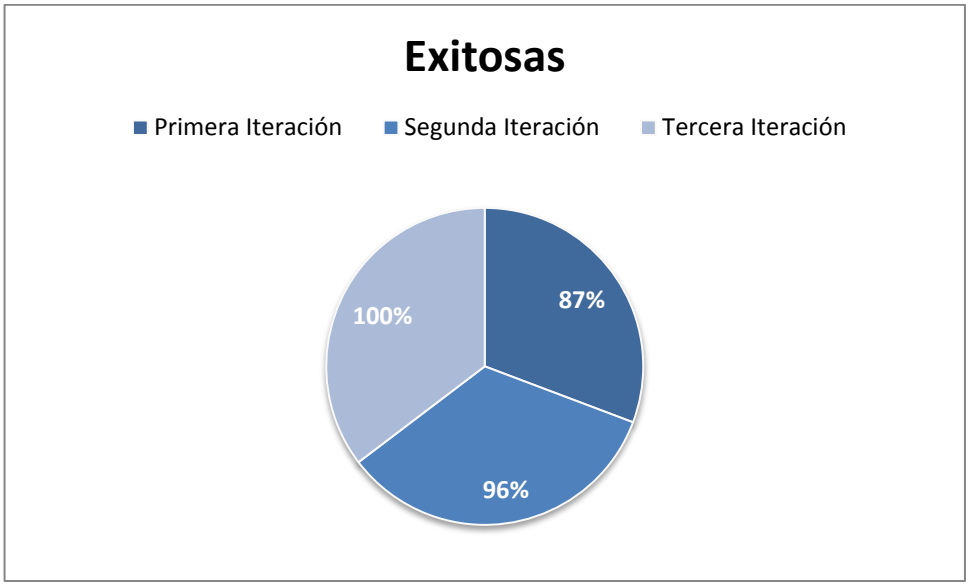



**Figura 16: Relación de pruebas de caja blanca exitosas por iteración**

### *3.3.2 PRUEBAS DE CAJA NEGRA.*

Las pruebas de caja negra son realizadas sobre el software de manera tal que se proporcionan entradas y se analizan las salidas para ver si son o no las esperadas. Conociendo la función para la cual fue diseñado, se hacen pruebas que demuestren que cada función es operativa y al mismo tiempo se buscan errores en cada una. Las llamadas pruebas de caja negra se basan en la especificación del programa o

componente a ser probado, para elaborar los casos de prueba. También son conocidas como pruebas de comportamiento o pruebas inducidas por los datos (22).

- Permiten obtener un conjunto de condiciones de entrada que ejecutan todos los requisitos funcionales de un programa.
- Las pruebas de caja negra no son alternativas a las técnicas de prueba de caja blanca. Es un enfoque complementario.

Las pruebas de caja negra intentan hallar errores tales como:

- Funciones incorrectas o ausentes.
- Errores en estructuras de datos.
- Errores de rendimiento.

Algunas técnicas utilizadas en la prueba de caja negra son:

- **Partición de equivalencia**: Técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición equivalente se dirige a una definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.
- **Análisis de valores limite**: La experiencia muestra que los casos de prueba que exploran las condiciones límites producen mejores resultados que aquellos que no lo hacen. Las condiciones límites son aquellas que se hayan en los márgenes de la clase de equivalencia, tanto de entrada como de salida. Por ello, se ha desarrollado el análisis de valores límite como técnica de prueba. Esta técnica lleva a elegir los casos de prueba que ejerciten los valores límites.
- **Grafos de causa y efecto**: Es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones o condiciones.

A continuación se muestra un ejemplo de la aplicación de la prueba de partición de equivalencia sobre el componente. La Partición de Equivalencia divide el dominio de entrada de un programa en un número finito de variables de equivalencia. Las variables de equivalencia representan un conjunto de estados

válidos y no válidos para las condiciones de entrada de un programa. Estas variables se definen en dos tipos, las válidas, que representan entradas válidas al programa y las no válidas, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor real de dato.

El proceso de generación de formularios web comienza con la entrada de un esquema XML al componente, y se espera como salida el código de un formulario web que responda a la estructura del esquema XML. Para ver el esquema XML utilizado para la realización del ejemplo ver Anexo #2: Esquema XML utilizado para el ejemplo de las pruebas.

Para el proceso de prueba de partición de equivalencia se utiliza la siguiente tabla. En esta primera prueba el documento del esquema XML está bien formado.

| Datos de entrada | Resultados esperados | Resultados obtenidos |
|---|---|---|
| <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"><br><xs:element name="DeclaracionJurada"><br><xs:complexType><br><xs:sequence><br><xs:element name="codigoImportador" minOccurs="1" maxOccurs="1" type="codigoImportador"/><br><xs:element name="codigoDeclarante" minOccurs="1" maxOccurs="1" type="codigoDeclarante"/><br><xs:element name="fechaEmision" type="xs:date" minOccurs="1" maxOccurs="1"/><br><xs:element name="numeroDeclaracion" minOccurs="1" maxOccurs="1" type="xs:string"/> | <html><br><head><script type="text/javascript" src="js/jquery-1.8.2.js"></script><br></head><br><body><br><form action="#" method="post"><br><label for="codigoImportador">codigoImportador</label><input type="text" name="codigoImportador"><br /><br><label for="codigoDeclarante">codigoDeclarante</label><input type="text" name="codigoDeclarante"><br /><br><label for="fechaEmision">fechaEmision</label><input type="text" name="fechaEmision"><br /><br><label for="numeroDeclaracion">numeroDeclaracion</label><input type="text" name="numeroDeclaracion"><br /><br><label for="codigoMonedaDM">codigoMonedaDM</label><input type="text" name="codigoMonedaDM"><br /><br><hr /><input type="submit" value="Enviar"></form><br><script type="text/javascript"><br></script></body><br></html> | <html><br><head><script type="text/javascript" src="js/jquery-1.8.2.js"></script><br></head><br><body><br><form action="#" method="post"><br><label for="codigoImportador">codigoImportador</label><input type="text" name="codigoImportador"><br /><br><label for="codigoDeclarante">codigoDeclarante</label><input type="text" name="codigoDeclarante"><br /><br><label for="fechaEmision">fechaEmision</label><input type="text" name="fechaEmision"><br /><br><label for="numeroDeclaracion">numeroDeclaracion</label><input type="text" name="numeroDeclaracion"><br /><br><label for="codigoMonedaDM">codigoMonedaDM</label><input type="text" name="codigoMonedaDM"><br /><br><hr /><input type="submit" value="Enviar"></form><br><script type="text/javascript"><br></script></body><br></html> |

| | | |
|---|---|---|
| <xs:element name="codigoMonedaDM" minOccurs="1" maxOccurs="1" type="monedaDC"/><br></xs:sequence><br></xs:complexType><br></xs:element><br></xs:schema> | | |

Para este ejemplo, el resultado esperado coincide con el resultado obtenido por el componente, por lo que la prueba se considera exitosa. Para el siguiente caso de prueba se le brinda al componente un esquema XML que no está bien formado, la etiqueta de cierre `</xs:complexType>` no existe.

| Datos de entrada | Resultados esperados | Resultados obtenidos |
|---|---|---|
| <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"><br><xs:element name="DeclaracionJurada"><br><xs:complexType><br><xs:sequence><br><xs:element name="codigoImportador" minOccurs="1" maxOccurs="1" type="codigoImportador"/><br><xs:element name="codigoDeclarante" minOccurs="1" maxOccurs="1" type="codigoDeclarante"/><br><xs:element name="fechaEmision" type="xs:date" minOccurs="1" maxOccurs="1"/><br><xs:element name="numeroDeclaracion" minOccurs="1" maxOccurs="1" type="xs:string"/><br><xs:element name="codigoMonedaDM" minOccurs="1" maxOccurs="1" type="monedaDC"/><br></xs:sequence><br></xs:element><br></xs:schema> | XSD Inválido | XSD Inválido |

Para este ejemplo la prueba al componente resultó exitosa. De manera general se realizaron un total de 7 pruebas de este tipo, para la primera iteración el 83% de las pruebas fueron exitosas y el 17 restante fallidas. Ya en la segunda iteración todas las pruebas resultaros exitosas.

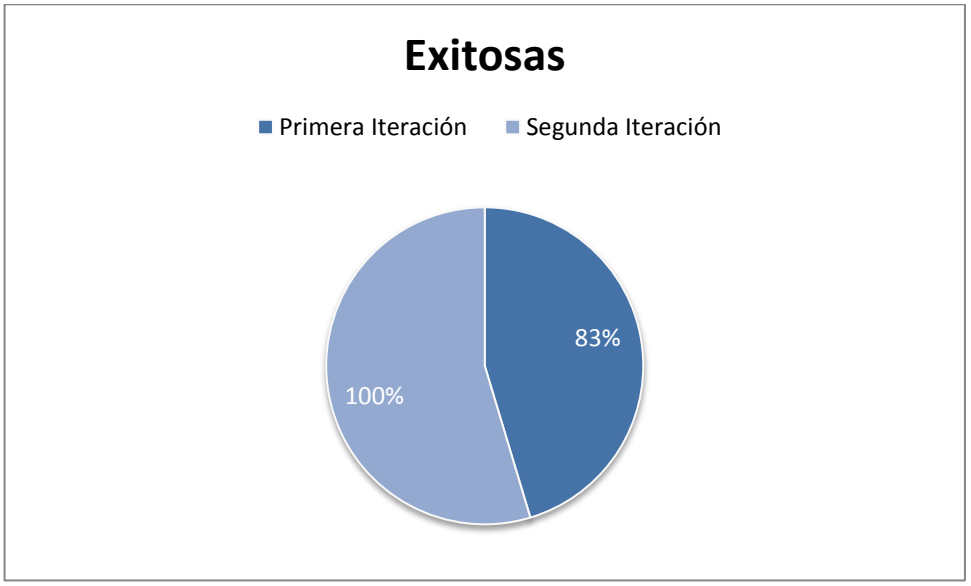



**Figura 17: Relación de pruebas de caja negra exitosas por iteración**

La aplicación de las pruebas antes mencionadas al software elaborado permitió la identificación y posterior corrección de los errores presentes durante la construcción del mismo, posibilitando la obtención de un producto de software con la calidad requerida. En el siguiente epígrafe se realiza la validación de las afirmaciones realizadas al comienzo de la investigación.

## 3.4 VALIDACIÓN DE LA SOLUCIÓN

Una vez terminada la investigación, es necesario evaluar y validar que las afirmaciones realizadas sobre la solución propuesta sean aceptables para la comunidad y el proyecto para la cual se construye. Los siguientes métodos (26) ofrecen medios para la evaluación y validación de la solución desarrollada.

### *3.4.1 DEMOSTRACIÓN*

El objetivo fundamental de este método es demostrar que la solución a un problema es realizable y válida en situaciones predefinidas. Se emplea cuando se ha desarrollado una situación de un problema, cuya solución no es posible demostrar matemáticamente con exactitud. Sin embargo, se desea demostrar que la solución es realizable y trabaja para un conjunto de situaciones predefinidas.

El método es especialmente relevante cuando la demostración de una solución en sí misma se considera una contribución. Para desarrollarlo es necesario tener en cuenta los siguientes pasos:

1. Construir la solución. Esto puede significar la construcción de un prototipo de la solución. La construcción de la solución mostrará que la misma es realizable.
2. Demostrar que la solución construida es razonable para un conjunto de situaciones predefinidas. Estas situaciones deben ser predefinidas y no creadas para adaptarse a la solución. Deberán estar construidas para ejercer las variaciones del problema.

La demostración puede mostrar las deficiencias de la solución. Por otro lado, se puede mostrar que la solución es viable y aceptable. Las pruebas exhaustivas aumentará la confianza en la solución. Si las situaciones de prueba están diseñadas apropiadamente, entonces la construcción de la solución y sus pruebas para estas situaciones puede demostrar su validez. A pesar de que teóricamente la Demostración constituye el método de validación menos formal, ha sido utilizado en innumerables artículos científicos de la especialidad hasta el punto de ser el tipo de validación que más se utiliza desde el año 1999 (27).

### *3.4.2 EXPERIMENTACIÓN*

Su objetivo es validar o rechazar un conjunto de hipótesis relacionadas con las afirmaciones acerca de la solución. Es empleado cuando se han desarrollado un conjunto de hipótesis relacionadas con las afirmaciones acerca de la solución (por lo general de un sistema) que no pueden ser probadas matemática o lógicamente. Es necesario para generar los datos del sistema y luego utilizar esta información para validar o rechazar las hipótesis.

La naturaleza del experimento y la validación de hipótesis dependen del tipo de experimento. Estos tipos, a su vez, dependen del método utilizado en el desarrollo de la solución. En general, un experimento debe

cumplir los siguientes criterios que inciden en la confianza o la generalidad de los resultados establecidos por el experimento:

1. La validez de la construcción. Los sustitutos de las construcciones que no pueden ser fácilmente observados en el experimento, deben ser sustitutos válidos.
2. La validez interna. El experimento no debe involucrar a las construcciones que influyen en el comportamiento observado, con los que forman parte de las hipótesis.
3. La validez externa. Si se supone que los resultados del experimento no son generales, sino que se prueban en un entorno limitado, simulado, se debe argumentar que los resultados son generalizables.
4. Fiabilidad. El experimento debe ser reproducible.

Este método ayudará a establecer resultados asociados con la solución del problema de investigación en situaciones donde la recogida y análisis de datos es el único método factible de validación.

### *3.4.3 SIMULACIÓN*

Se encarga de evaluar y validar una solución al problema de la investigación. Es aplicable cuando el problema de investigación es complejo, tal que una solución, no puede ser demostrada matemáticamente como válida. Además cuando la evaluación y validación de la solución en el ámbito de la vida real no es viable o muy costosa.

En la utilización de este método se tienen en cuenta los siguientes pasos:

1. Desarrollar el modelo conceptual del problema y su solución que será simulado en una computadora. Esto implica decidir qué entidades e interacciones deben ser capturadas en la simulación, cuyo propósito es evaluar el desempeño de la solución al problema y probar su validez.
2. Desarrollar un conjunto inicial de datos de prueba que pueden ejercer el modelo. Esto debe tener en cuenta los objetivos de la solución (artefacto) y el entorno exterior en el que la misma debe operar. Esto implicará la modelización del entorno exterior.
3. Seleccionar un paquete de simulación que está diseñado específicamente para el dominio del problema. Esto implicará la menor cantidad de programación.

4. Ejecutar el programa de simulación para el conjunto de pruebas desarrollado previamente. Recoger datos de rendimiento y analizarlo para evaluar la solución. Si el desempeño no cumple con las expectativas, entonces se puede volver a analizar y revisar la solución. En caso contrario, comprobar la solución sobre una amplia gama de condiciones. Probar la solución en condiciones extremas para ver la gama de condiciones ambientales exteriores sobre el cual la solución es válida.
5. Argumentar que el análisis de los datos apoya la validez de las hipótesis acerca de la solución.

La Simulación ofrece una forma razonable de coste efectivo de evaluación y validación de una solución. La alternativa de poner a prueba la solución en la configuración de la vida real puede ser a la vez costosa y consume mucho tiempo, o tal vez ni siquiera sea factible.

### *3.4.4 USO DE MÉTRICA*

Tiene como propósito usar medidas establecidas para ayudar a la validación de la propia solución al problema de investigación. Se emplean indicadores establecidos que existen en la literatura para evaluar el desempeño de la solución; para probar o argumentar la corrección de las hipótesis que se han hecho en relación con el rendimiento de la solución. En el caso de que las métricas no estén disponibles para medir el rendimiento de la solución, se pueden usar para un problema similar.

Este método sigue las siguientes formalidades:

1 Determinar si existen o no las métricas establecidas que son apropiadas para medir el rendimiento de la solución y compararlo con el rendimiento de soluciones anteriores (si es que existen). Si tales parámetros no existen, determinar si existen o no las métricas para medir el desempeño de problemas similares a nuestro propio problema. En tal caso, se debe argumentar que el uso de las métricas elegidas es una forma razonable de evaluación y validación.
2 Analizar y medir la solución usando la o las métricas elegidas. Esto puede implicar las pruebas matemáticas, las mediciones experimentales, o simulación.
3 Demostrar que la solución tiene la hipótesis de rendimiento de acuerdo a las métricas seleccionadas.

El uso de uno o más métodos pueden ayudar a convencerse a sí mismo y la comunidad investigadora, de la validez y valor de la solución. Esto, a su vez, es muy importante en la publicación de los resultados. En

este escenario, para evaluar y probar la validez de la solución propuesta en esta investigación, se adoptará el método Demostración, por ser apropiado para un procedimiento novedoso y que resuelve un problema para el cual no existe solución previa. Por último, la demostración constituye el método de validación más utilizado en las publicaciones científicas para el área de conocimiento de las **c**iencias de la Computación, de acuerdo a Shaw, M (27)**.**

Para la demostración de la validez de la solución es necesario el entendimiento del proceso actual de generación de formularios a partir de un esquema XML dentro del proyecto Ventanilla Única del Comercio en Cuba. El proceso inicia cuando un esquema XML es subido a la Ventanilla por parte de algunas de las entidades que harán uso de la misma, posteriormente este documento es revisado por el analista del proyecto, el cual extrae la información del documento XML (campos, tipos de datos complejos, restricciones sobre los campos). Con estos datos el analista realiza la descripción de la información obtenida, elabora un prototipo de interfaz de usuario y describe el funcionamiento de lo que se hará con la información del formulario. Teniendo todos los elementos antes citados, el analista hace entrega de los mismos al diseñador del proyecto.

El diseñador del proyecto al recibir el prototipo de interfaz de usuario y la descripción de su funcionamiento, construye un diagrama de clases que dé respuesta a las necesidades solicitadas por el analista. Una vez alcanzado este punto, el desarrollador elabora el código HTML del formulario. En la figura siguiente se muestra el proceso descrito anteriormente.

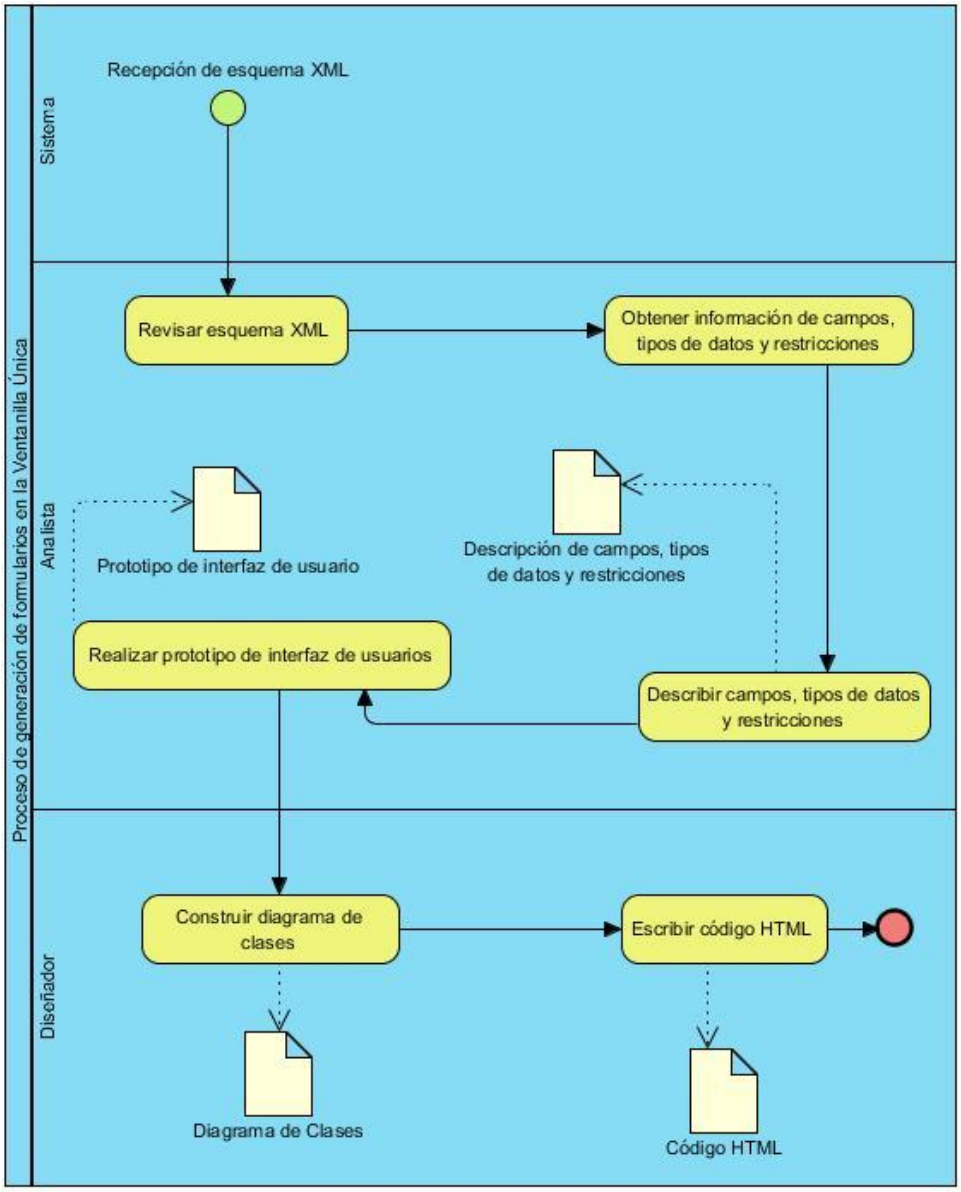



**Figura 18: Proceso de generación de formularios web en el proyecto Ventanilla Única del Comercio Exterior**

Se constató con los analistas y los diseñadores del proyecto que para escribir el código HTML a partir de un esquema XML en el proyecto es necesario entre 7 y 10 días, debido a la complejidad del esquema XML y en otros casos a la propia planificación de trabajo de los integrantes del proyecto.

### 3.4.5 DESCRIPCIÓN DE LOS CASOS DE PRUEBAS

A continuación se relacionan una serie de pruebas para demostrar que mediante la utilización del componente de generación de formularios web el tiempo entre la recepción de un esquema XML y la construcción del código HTML se reduce considerablemente.

Mencionar que para la realización de estas pruebas se le realizó una pequeña modificación al código fuente del componente, permitiendo contar la cantidad de etiquetas analizadas por este.

Para el primer caso de prueba se utiliza un esquema XML que representa a una Declaración Jurada. Una declaración jurada está compuesta por 7 elementos, 2 de los cuales son tipos compuestos. Los 5 elementos no compuestos son: *codigoImportador, codigoDeclarante, fechaEmision, numeroDeclaracion y codigoMonedaDM*. Se entiende como un elemento no compuesto al elemento que su entrada de datos se puede considerar como un campo de texto. El primer elemento compuesto de denomina *totales*, el cual está compuesto por los siguientes elementos no compuestos: *totalFacturaOriginal, totalFleteOriginal, totalSeguroOriginal, totalOtrosGastosOriginal, codigoMoneda, tasaCambio, totalFacturaConvertido, totalFleteConvertido, totalSeguroConvertido, totalOtrosGastosConvertido*. El elemento *totales* se representa como un listado de los elementos mencionados. El segundo elemento compuesto se denomina *partidas,* el cual está compuesto por los siguientes elementos no compuestos: *partida, facturaOriginal, fleteOriginal, seguroOriginal, otrosGastosOriginal, codigoMoneda, tasaCambio, facturaConvertido, fleteConvertido, seguroConvertido, otrosGastosConvertido*. Los nombres de los elementos son tomados textualmente del esquema XML utilizado para la prueba. Para ver el esquema XML utilizado para esta prueba, ver Anexo 3: Esquema XML Declaración Jurada.

Para la medición del tiempo de generación del código HTML se utilizó la herramienta Firebug. Es un complemento que se le añade al navegador Mozilla Firefox y permite la revisión de estadísticas relacionadas con las peticiones HTTP como el tiempo de la petición, parámetros de envío y de respuesta, entre otros. Para la medición del tiempo de generación se realizaron 5 peticiones al componente, los tiempos de estas peticiones quedaron registrados en la siguiente tabla.

| Número de Petición | Tiempo |
|---|---|

| | |
|---|---|
| 1 | 33 ms |
| 2 | 27 ms |
| 3 | 39 ms |
| 4 | 190 ms |
| 5 | 51 ms |

**Tabla 22: Tiempo de respuesta del componente**

El tiempo promedio de generación del código del formulario es de 68 milisegundos.

Los resultados de pasar el esquema XML al componente se detallan en la siguiente tabla:

| Cantidad de etiquetas esquema XML Declaración Jurada. | Cantidad de etiquetas analizadas por el componente | Cantidad de campos significativos del esquema XML Declaración Jurada. | Cantidad de campos del formulario HTML. | Tiempo promedio de generación del código HTML del formulario. |
|---|---|---|---|---|
| 42 | 42 | 26 | 26 | 68 milisegundos |

**Tabla 23: Resultados de la prueba al esquema XML Declaración Jurada**

Para ver el formulario generado en esta prueba ir al anexo 5: Formulario Web del esquema XML Declaración Jurada. Advertir que al final del anexo se muestra la cantidad de etiquetas analizadas por el sistema.

Para la segunda prueba fue utilizado el esquema XML que representa una Anulación de Documentos Complementarios. La anulación de documentos complementarios cuenta con un elemento compuesto. Este está constituido por los siguientes elementos no compuestos: *numeroDocumento, anno y codigoTipoDocumento*. Para ver el esquema Anulación de documentos complementarios, ver Anexo 4: Esquema XML Anulación de Documentos Complementarios.

Los tiempos de las peticiones para este esquema XML se muestran a continuación.

| Número de Petición | Tiempo |
|---|---|
| 1 | 24 ms |
| 2 | 33 ms |
| 3 | 31 ms |
| 4 | 37 ms |
| 5 | 22 ms |

**Tabla 24: Tiempo de respuesta del componente**

EL tiempo promedio para este caso es de 25,4 milisegundos.

Los resultados de pasar el esquema XML por el componente se muestran a continuación.

| Cantidad de etiquetas esquema XML Anulación de Documentos Complementarios. | Cantidad de etiquetas analizadas por el componente | Cantidad de campos significativos del esquema XML Anulación de Documentos Complementarios. | Cantidad de campos del formulario HTML. | Tiempo promedio de generación del código HTML del formulario. |
|---|---|---|---|---|
| 13 | 13 | 3 | 3 | 25,4 milisegundos |

**Tabla 25: Resultados de la prueba al esquema XML Anulación de Documentos Complementarios**

Para ver el formulario generado en esta prueba ir al anexo 6: Formulario Web del esquema XML Anulación de Documentos Complementarios. Advertir que al final del anexo se muestra la cantidad de etiquetas analizadas por el sistema.

Mediante el análisis de los resultados de las pruebas descritas anteriormente, se puede observar que el tiempo de generación del código HTML del formulario a partir de la recepción de un esquema XML fue reducido considerablemente, de unos 7 a 10 días que se demora en la Ventanilla Única del Comercio Exterior, al usar el componente de generación de formularios web se redujo a menos de 1 segundo (1000 milisegundos).

## CONCLUSIONES DEL CAPÍTULO

En el presente capítulo se detallaron el diagrama de compontes y la nomenclatura del código utilizada, además de las pruebas realizadas al software, por lo que se pudo concluir:

- El uso de nomenclaturas mejora la comprensión del código fuente, además que indica la manera de nombrar las variables, las clases y las operaciones.
- Mediante las pruebas de caja blanca y de caja negra se pudo constatar que la implementación del componente funciona de manera correcta.
- Se demostró que el componente reduce el tiempo de generación de formularios a partir de un esquema XML.

# CONCLUSIONES GENERALES

En este trabajo se implementó el componente de generación de formularios web a partir de un esquema XML de la Ventanilla Única del Comercio Exterior. Los resultados alcanzados en la investigación permiten concluir lo siguiente:

- Mediante el establecimiento del marco conceptual de la investigación se logró dominio de los contenidos a tratar en la misma, dando cumplimiento al primer objetivo específico.
- Se definió el estándar de generación de formularios web, facilitando el entendimiento de la solución desarrollada.
- Se modeló e implementó la solución a partir del estándar de generación de formularios web.
- Se probó la validez del resultado haciendo uso del método de demostración, obteniendo resultados satisfactorios.

# RECOMENDACIONES

Una vez analizados los resultados de la investigación se recomienda lo siguiente:

- Extender la solución para permitir generar formularios a partir de cualquier esquema XML.
- Agregar validaciones al componente para comprobar que los tipos definidos estén presentes en el esquema XML.

# BIBLIOGRAFÍA

1. **Society, Internet.** Breve historia de internet. [En línea] [Citado el: 8 de Diciembre de 2012.] http://www.internetsociety.org/es/breve-historia-de-internet?gclid=CMrWktiEibQCFcyf4AoduRwAgg#Origins.

2. **W3C.** Sobre el W3C - W3C España. [En línea] [Citado el: 6 de Diciembre de 2012.] http://www.w3c.es/Consorcio/.

3. **David C. Fallside, Priscilla Walmsley.** XML Schema Part 0: Primer Second Edition. [En línea] [Citado el: 22 de Enero de 2013.] http://www.w3.org/TR/xmlschema-0/.

4. **W3C.** XML Essentials. [En línea] [Citado el: 06 de Diciembre de 2012.] http://www.w3.org/standards/xml/core.

5. **República, Aduana General de la.** [En línea] [Citado el: 9 de Diciembre de 2012.] http://www.aduana.co.cu/.

6. **W3C.** Extensible Markup Language (XML). [En línea] [Citado el: 22 de Enero de 2013.] http://www.w3.org/XML/.

7. **Tim Bray, Jean Paoli, C. M. Sperberg-McQuee, Eve Maler, François Yergeau.** Extensible Markup Language (XML) 1.0 (Fifth Edition). [En línea] [Citado el: 22 de Enero de 2013.] http://www.w3.org/TR/REC-xml/.

8. **Autores, Conjunto de.** What characterizes a (software) component? [En línea] [Citado el: 5 de Enero de 2013.] http://www.softwareresearch.net/fileadmin/src/docs/publications/J010.pdf.

9. **Reader, ECOOP'96 Workshop.** WCOP'96 Summary. dpunkt Verlag : s.n., 1997. ISBN 3-920993-67-5.

10. **Sommerville, Ian.** *Ingeniería del Software Septima Edición.* Madrid : Pearson Educación, 2005. 84-7829-074-5.

11. **W3Schools.** DTD Tutorial. [En línea] [Citado el: 25 de Enero de 2013.] http://www.w3schools.com/dtd/default.asp.

12. **Lapuente, María Jesús Lamarca.** DTDs y XML Esquema. [En línea] [Citado el: 2 de Diciembre de 2012.] http://www.hipertexto.info/documentos/dtds.htm.

13. **Dongwon Lee, Wesley W. Chu.** Comparative Analysis of Six XML Schema Languages. [En línea] [Citado el: 22 de Enero de 2013.] http://www.cobase.cs.ucla.edu/tech-docs/dongwon/sigmod-record-00.html.

14. **Microsoft.** XML-Data Reduced Schema. [En línea] [Citado el: 15 de Diciembre de 2013.] http://msdn.microsoft.com/en-us/library/dd961025(v=office.12).aspx.

15. **Entidades, Centro de Informatización de la Gestión de.** *Modelo de Desarrollo de Software.* 2012.

16. **Corporation, Oracle.** NetBeans IDE. [En línea] [Citado el: 23 de Enero de 2013.] http://netbeans.org/features/index.html.

17. **Carmina Lizeth Torres Flores, Germán Harvey Alférez Salinas Navojoa.** *Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa usando OpenUP.* 2008.

18. **Rumbaugh, James, Ivar.** *El Lenguaje Unificado de Modelado.* 2000.

19. PHP: Hypertext Preprocessor. [En línea] [Citado el: 14 de Noviembre de 2012.] http://php.net/.

20. **Hinostroza, Raul Rodas.** *Características de PHP.*

21. **Villalobos.** Ciberaula. [En línea] 2006. [Citado el: 19 de Enero de 2013.] http://linux.ciberaula.com/articulo/linux_apache_intro/.

22. **Pressman, Roger S.** *Ingeniería del Software Un enfoque práctico.*

23. **Mendoza Navarro.** Diseño del Sistema de Tarjeta de Crédito con UML. [En línea] [Citado el: 6 de 04 de 2013.] http://sisbib.unmsm.edu.pe/bibvirtualdata/tesis/basic/mendoza_nj/cap5.pdf.

24. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley.** *Design Patterns. Elements of Reusable Object-Oriented Software.*

25. **tvdi.det.uvigo.es.** Servicios para la Sociedad de la Información. [En línea] [Citado el: 21 de 04 de 2013.] http://tvdi.det.uvigo.es/~avilas/UML/node49.html.

26. **Vaishnavi, Vijay K. and Jr, William Kuechler.** *Design Science Research Methods and Patterns.* 2008.

27. **Shaw, M.** *What makes good research in software engineering.* SPRINGER BERLIN / HEIDELBERG : FOR TECHNOLOGY TRANSFER (STTT), 2002.

28. **W3C.** Schema - W3C. [En línea] [Citado el: 23 de Enero de 2013.] http://www.w3.org/standards/xml/schema.

29. **Fabien Potencier, François Zaninotto.** *Symfony La Guia Definitiva.* CA, USA : Apress Berkely, 2007. ISBN:1590597869.

30. **Eguiluz, Javier.** *Desarrollo Ágil con Symfony2.* 2012.

31. **Corporation, Oracle.** NetBeans IDE. [En línea] [Citado el: 23 de Enero de 2013.] http://netbeans.org/community/releases/72/.

32. **Vlist, Eric van der.** *XML Schema.* s.l. : O'Reilly Media, Inc, 2002.

33. **Foundation, Eclipse.** About the Eclipse Foundation. [En línea] [Citado el: 22 de Enero de 2013.] http://www.eclipse.org/org/.

34. **JetBrains.** PHP IDE :: JetBrains PhpStorm. [En línea] [Citado el: 21 de Enero de 2013.] http://www.jetbrains.com/phpstorm/.

35. —. PHPStorm Features. [En línea] [Citado el: 22 de Enero de 2013.] http://www.jetbrains.com/phpstorm/features/index.html#PHP_UML.

36. **IBM.** IBM Software - Rational Rose. [En línea] [Citado el: 24 de Enero de 2013.] http://www-01.ibm.com/software/awdtools/developer/rose/.

37. **Lic. Francisco Javier Díaz, Lic. Claudia Mariana Banchoff Tzancoff.** *PHP: una solución "open source" para el desarrollo de páginas Web dinámicas.* La Plata : LINTI. Facultad de Informática.

38. **RUMBAUGH, I.J.G.B.J.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.

39. **Electrónicos, Instituto de Ingenieros Eléctricos y.** *IEEE Computer Dictionary. Software Engineering Terms.* s.l. : IEEE 610-1990, 1990.

40. **Sommerville, Ian.** *Ingeniería de Software. 6ta. Edición.* s.l. : Prentice-Hall, 2002. ISBN 970-26-0206-8.

# ANEXOS

Anexo #1: Ejemplo de documento XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
   <Mensaje>
     <Remitente>
        <Nombre>Nombre del remitente</Nombre>
        <Mail> Correo del remitente </Mail>
     </Remitente>
     <Destinatario>
        <Nombre>Nombre del destinatario</Nombre>
        <Mail>Correo del destinatario</Mail>
     </Destinatario>
     <Texto>
        <Asunto>
          Este es mi documento con una estructura muy sencilla
          no contiene atributos ni entidades...
        </Asunto>
        <Parrafo>
          Este es mi documento con una estructura muy sencilla
          no contiene atributos ni entidades...
        </Parrafo>
     </Texto>
   </Mensaje>
</root>
```

Anexo #2: Esquema XML utilizado para el ejemplo de las pruebas.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="DeclaracionJurada">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="codigoImportador" minOccurs="1" maxOccurs="1"
type="codigoImportador"/>
    <xs:element name="codigoDeclarante" minOccurs="1" maxOccurs="1"
type="codigoDeclarante"/>
    <xs:element name="fechaEmision" type="xs:date" minOccurs="1" maxOccurs="1"/>
    <xs:element name="numeroDeclaracion" minOccurs="1" maxOccurs="1"
type="xs:string"/>
    <xs:element name="codigoMonedaDM" minOccurs="1" maxOccurs="1" type="monedaDC"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

```
</xs:schema>
```

Anexo #3: Esquema XML Declaración Jurada.

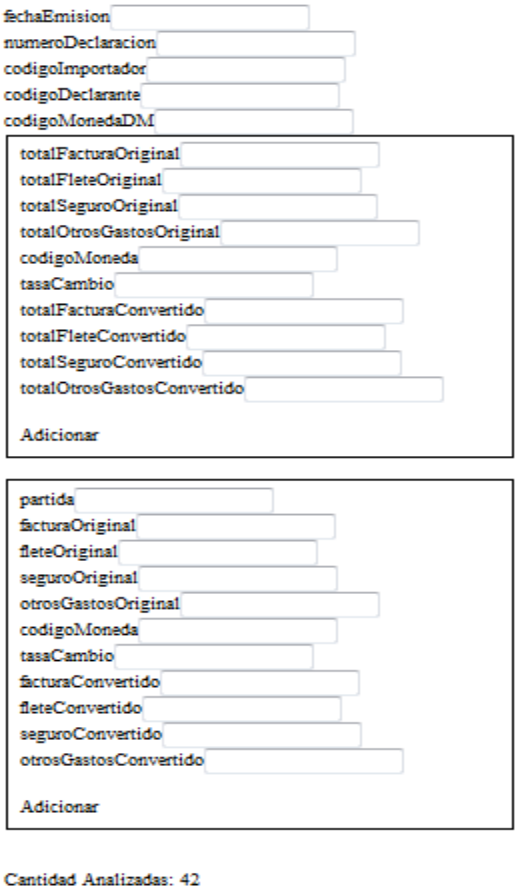
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="DeclaracionJurada">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="codigoImportador" minOccurs="1" maxOccurs="1"
type="codigoImportador"/>
                <xs:element name="codigoDeclarante" minOccurs="1" maxOccurs="1"
type="codigoDeclarante"/>
                <xs:element name="fechaEmision" type="xs:date" minOccurs="1"
maxOccurs="1"/>
                <xs:element name="numeroDeclaracion" minOccurs="1" maxOccurs="1"
type="xs:string"/>
                <xs:element name="codigoMonedaDM" minOccurs="1" maxOccurs="1"
type="monedaDC"/>
                <xs:element name="totales" type="detalleTotales" minOccurs="1"
maxOccurs="1"/>
                <xs:element name="partidas" type="detallePartidas" minOccurs="1"
maxOccurs="1"/>
            </xs:sequence></xs:complexType></xs:element>
    <xs:complexType name="detalleTotales">
        <xs:sequence>
            <xs:element name="total" type="detalleTotal" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence></xs:complexType>
    <xs:complexType name="detalleTotal">
        <xs:sequence>
            <xs:element name="totalFacturaOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="totalFleteOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="totalSeguroOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="totalOtrosGastosOriginal" type="decimalCuatro"
minOccurs="1" maxOccurs="1"/>
            <xs:element name="codigoMoneda" type="monedaDC" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="tasaCambio" type="xs:string" minOccurs="1"
maxOccurs="1"/>
```

```
            <xs:element name="totalFacturaConvertido" type="decimalCuatro"
minOccurs="1" maxOccurs="1"/>
            <xs:element name="totalFleteConvertido" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="totalSeguroConvertido" type="decimalCuatro"
minOccurs="1" maxOccurs="1"/>
            <xs:element name="totalOtrosGastosConvertido" type="decimalCuatro"
minOccurs="1" maxOccurs="1"/>
        </xs:sequence></xs:complexType
    <xs:complexType name="detallePartidas">
        <xs:sequence>
            <xs:element name="partidas" type="detallePartida" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="detallePartida">
        <xs:sequence>
            <xs:element name="partida" type="codigoSACLAP" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="facturaOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="fleteOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="seguroOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="otrosGastosOriginal" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="codigoMoneda" type="monedaDC" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="tasaCambio" type="xs:string" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="facturaConvertido" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="fleteConvertido" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="seguroConvertido" type="decimalCuatro" minOccurs="1"
maxOccurs="1"/>
            <xs:element name="otrosGastosConvertido" type="decimalCuatro"
minOccurs="1" maxOccurs="1"/>
        </xs:sequence></xs:complexType></xs:schema>
```

Anexo #4: Esquema XML Anulación de Documento Complementario.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="AnulacionDocumentosComplementarios">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="listaDocumentos" type="detalleListaDocumentos"
minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element><!-- Lista Documentos -->
    <xs:complexType name="detalleListaDocumentos">
        <xs:sequence>
            <xs:element name="documento" type="detalleListaDocumento" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="detalleListaDocumento">
        <xs:sequence>
            <xs:element name="numeroDocumento" type="xs:nonNegativeInteger"
minOccurs="1" maxOccurs="1"/>
            <xs:element name="anno" type="anno" minOccurs="1" maxOccurs="1"/>
            <xs:element name="codigoTipoDocumento" type="xs:string" minOccurs="1"
maxOccurs="1"/><!-- verificar -->
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Anexo 5: Formulario Web del esquema XML Declaración Jurada.




Anexo 6: Formulario Web del esquema XML Anulación de Documentos Complementarios.