Multilistas aplicadas a la implementación de un parseador de XML para la definición de XPDL 2.2 en objective c para iOS

Multilists applied to an XML parser implementation in Objective C for iOS for XPDL 2.2 standard

Daniel Iván Meza Lara
Ingeniero de Sistemas, Universidad Piloto de Colombia
Joven Investigador Grupo InnovaTIC,
Universidad Piloto de Colombia
danielmezal@gmail.com

Leidy Andrea Ruiz Rodríguez
Ingeniero de Sistemas, Universidad Piloto de Colombia
Joven Investigador Grupo InnovaTIC,
Universidad Piloto de Colombia
Leidy.ruiz.r@gmail.com

Óscar Elías Herrera Bedoya
Doctor en Telecomunicaciones,
Universidad Politécnica de Valencia
Docente Tiempo Completo, Investigador Grupo InnovaTIC,
Universidad Piloto de Colombia
oscar-herrera@unipiloto.edu.co

Resumen— En este trabajo se expone la implementación de estructuras de datos en el desarrollo de un parseador que permite la interpretación de archivos XPDL (XML ProcessDefinitionLanguage) en su versión 2.2, mediante multilistas. Junto con el metamodelo propio del XPDL se busca solucionar una problemática en la interpretación del esquema XML, permitiendo un correcto almacenamiento de los elementos bajo el lenguaje Objective C para iOS, con el cual se pretende innovar en el campo de las plataformas móviles que hacen uso del lenguaje estándar BPMN (BusinessProcessModeling Notation) para la representación de procesos de negocio y que generan el XPDL. El objetivo principal de un XPDL es describir la información del flujo de datos del proceso mediante un esquema XML (Extensible MarkupLanguage).

Palabras clave— BPMN, GDataXML, Objective C, Parser, Procesos de Negocio, XPDL.

Abstract— With the creation of the standard language BPMN(Business Process Modeling Notation) used to represent business processes, the XPDL(XML Process Definition Language) is generated, which describes the data flow information of the process using a XML(Extensible Markup Language) schema. This document shows the implementation of data structures on the development of a parser which allows the interpretation of XPDL files in the 2.2 version; using together multi-lists and the XPDL meta-model, the interpretation of the XML schema problematic is pretended to be solved, allowing a correct storage of the elements. As an additional contribution, the development of the functional parser is made under the Objective C language for iOS, which is pretended to innovate in the mobile platform field.

Keywords— BPMN, GDataXML, Objective C, Parser, Business Process, XPDL.

I. INTRODUCCIÓN

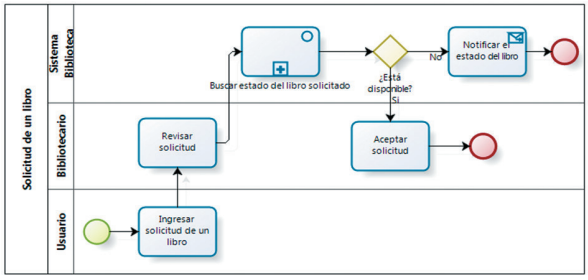
En este artículo, se presenta un modelo para la implementación de estructuras de datos complejas que faciliten la generación de documentos XPDL (*XML ProcessDefinitionLanguage*) [1], los cuales son generados de forma organizada por medio de un esquema conceptual definido por el metamodelo propio.

Con la apropiación del esquema conceptual se componen las jerarquías del XML (*Extensible MarkupLanguage*)[2], que basado en el-estándar XPDL 2.2, contiene la representación de los elementos propios del modelado de procesos de negocio bajo el lenguaje BPMN (*Business Process-ModelingNotation*)[3] cuya representación gráfica del ejemplo de un proceso se muestra en la Fig. 1.

La representación de los procesos de negocio por medio del BPMN dentro de un modelador, muestra la unión entre los objetos y las relaciones entre ellos de forma gráfica, a partir de esto se genera el XPDL en el cual se guarda, por medio

de un esquema XML, todas las características propias del proceso. (Ver Tabla I).

Fig. 1 EJEMPLO DE LA REPRESENTACIÓN GRÁFICA DE UN PROCESO EN NOTACIÓN BPMN



Para la definición de los tipos de datos e inclusión de cada elemento del BPMN dentro del esquema XML, se realiza un modelo preliminar que valide los elementos básicos del BPMN; la abstracción del modelo conceptual define de manera general cada elemento de flujo que pueda presentarse dentro del proceso diagramado; posteriormente se realiza la ampliación y definición de los tipos de objetos creando así un metamodelo.

El metamodelo generado contiene todos los elementos de flujo usados para la representación del BPMN, según cada elemento, se definen sus atributos propios y las relaciones entre objetos [4].

Con la conceptualización del esquema XML y según cada elemento propio del metamodelo [5] se construyen las estructuras de datos denominadas multilistas que, al ser implementadas al esquema, buscan de forma sencilla acceder a los elementos y sus respectivos atributos; las multilistas se encargan de guardar las características de los objetos y sus atributos.

Las relaciones entre los objetos van de la mano con nodos establecidos para cada objeto, los cuales se encargan de enlazar los elementos padres (contenedores), por ejemplo los Pools, y los nodos hijo (contenido), por ejemplo, los Lanes, de esta manera se asegura que el XPDL creado cumpla con el estándar 2.2 y el esquema XML definido previamente; con esto se asegura que cada elemento incluido dentro del estándar BPMN, desde un Pool hasta un elemento *DataStore*, cree un nodo índice propio y se conecte al elemento próximo para construir la jerarquía del XPDL.

Como la estructura de XPDL se crea a partir del esquema XML, se deduce que en la plataforma *iOS* la interpretación del archivo del XPDL 2.2 no tiene problema al iniciar el *parseo* del proceso, por esta razón, al iniciar la implementación de dicho *parseador* se analiza sobre qué tipos de *parsers* pueden ser implementados en Objective C[6]. Para la creación del *parser* se toma como punto de partida el esquema XML definido para el estándar XPDL 2.2.

El analizador sintáctico que realiza el *parseo*, tanto de escritura como de lectura del documento con extensión XPDL usa el API para Google XML de uso exclusivo para *Objective* C denominado *GDataXML*[7], el cual permite el manejo de archivos de forma dinámica y sin alterar el rendimiento de la memoria en el dispositivo, lo cual es importante al desarrollar aplicaciones para dispositivos móviles de Apple [8].

Para concluir con la implementación del *parseador* se realizan pruebas tanto del rendimiento y consumo de procesos dentro del dispositivo como de la ejecución del *parseador* XPDL.

II. ESQUEMA CONCEPTUAL XPDL

El lenguaje estándar XPDL fue desarrollado por WfMC (*Workflow Management Coallition*) [9] en 2001, cuyo objetivo principal es almacenar y modificar las características del diagrama del proceso, dicho lenguaje permite por un lado leer y editar los procesos y por el otro ejecuta el modelo en un compilador de XPDL en una suite BPM [10].

A. Modelo preliminar BPMN

Tomando como base la diagramación del BPMN y la especificación actual del lenguaje XPDL, definida por WfMC para su nueva versión 2.2, se propone un esquema conceptual básico que especifica qué entidades y relaciones existen al momento de realizar la creación del XPDL.

Como primera instancia se tienen en cuenta todos los elementos que contiene el BPMN como notación (ver Fig. 2), de estos se parte un modelo inicial, dicho modelo abstrae de manera general, los elementos diagramados por medio del esquema BPD (*Business ProcessDiagram*)[11].

Obtenida la abstracción de los elementos usados en el BPMN y establecidas relaciones entre ellos, el paso siguiente es crear el diagrama conceptual (ver Fig. 3), el cual conllevará a la creación del metamodelo propio del XPDL.

ITECKNE Vol. 9 Número 2 • ISSN 1692 - 1798 • Diciembre 2012 • 52 - 61

El modelo conceptual contiene, además de los elementos definidos del BPMN, el tipo de relaciones que se establecen entre los objetos, creando así un paquete llamado BPD, el cual mantiene cada elemento en un contenedor interno (*Package*); dependiendo de los elementos se establece una relación de acuerdo a cuantos elementos podrían existir dentro del contenedor interno, para el caso de los *Swimlanes*, siempre iniciará en uno y no estará limitada la creación de elementos, por otro lado, para el elemento *WorkFlowProcesses*, la creación está limitada a uno, ya que solamente en un proceso diagramado habrá un flujo de eventos.

Fig. 2 MODELO PRELIMINAR BPMN

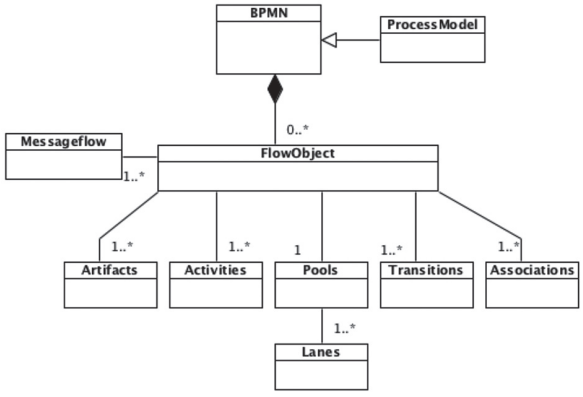
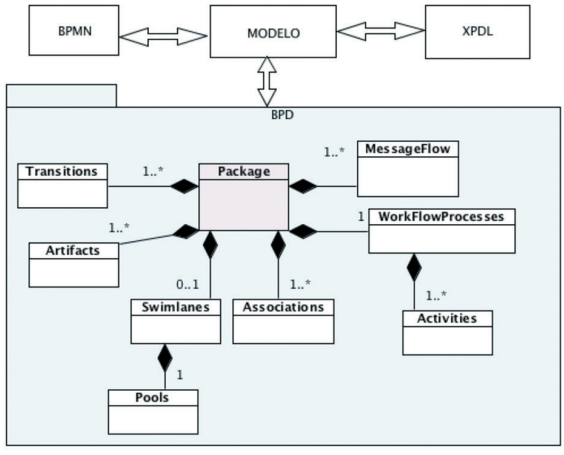


Fig. 3 DEFINICIÓN DEL MODELO CONCEPTUAL PARA XPDL



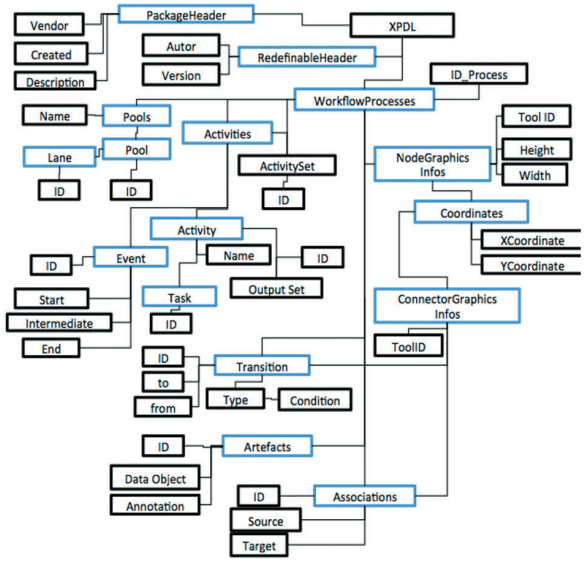
La definición del paquete permite la especificación de un número de atributos de definición de elementos, los cuales serán aplicados en definición de procesos individuales contenidos dentro del paquete.

A partir de la anterior definición del modelo conceptual del XPDL y mediante una abstracción profunda sobre cada elemento, se establece el metamodelo, en el cual se definen de manera individual las relaciones entre cada tipo de objetos.

B. Metamodelo XPDL

El metamodelo [12] identifica las entidades y atributos para el intercambio, o almacenamiento del modelo de procesos, establece una serie de reglas de herencia para asociar una delimitación del proceso individual con definiciones de entidades por especificación de participantes, los cuales se establecen en el nivel de paquetes en vez del nivel de definición individual de procesos (ver Fig. 4).

Fig. 4 METAMODELO XPDL



La estructura jerárquica del metamodelo permite generar de manera estructurada la creación del *parseador*; posterior al anterior análisis se hace una representación gráfica y comparación entre dos lenguajes BPMN y XPDL.

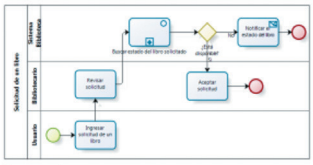
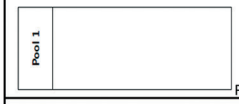
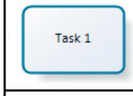
III. IMPLEMENTACIÓN DEL *PARSER*

El lenguaje XPDL y BPMN son muy similares, se organizan en forma de organigrama [13], una forma simple de ver sus similitudes es mostrar gráficamente (ver Tabla I) elementos específicos en el código XPDL que representa un objeto gráfico en el proceso.

Tomando como ejemplo el XPDL generado por *BizAgiModeler®*, se muestra en la Tabla 1.

TABLA I

REPRESENTACIÓN DE UN BPMN EN LENGUAJE XPDL. GENERADO POR BIZAGI.

BPMN	XPDL
 Proceso interno	*<WorkflowProcess></WorkflowProcess>*
 Pool	<Pool> </Pool>
 Evento inicio	<Activity><StartEventTrigger="None" /></Activity>
 Tarea 1	<Activities><Activity Id="16ef8901" Name="Task1"></Activities>
 Compuerta 1	<Activity Id="c3190c0c" Name="Compuerta 1"><Description /><Route />

Conocido el modo de interpretación que le da el lenguaje Objective C al esquema XML y con un amplio conocimiento en la creación de métodos que abstraen la información presentada en los archivos XML, se debe programar de manera lógica la búsqueda de elementos internos y sus atributos, esto se logra mediante la implementación de un *parser* o analizador sintáctico que actúe de manera dual, ya que el *parser* debe interpretar y crear un archivo XPDL según el estándar en su versión 2.2.

El proceso para iniciar el *parser* XML, requiere de la entrada de un archivo, el cual contiene el valor de cada elemento individual definido.

A. Definición de Parser

En *iOS* no existe propiamente un marco referente a las definiciones XML como XML Documents para aplicaciones ejecutadas en OSX [14], por tal razón es primordial crear un *parseador* propio para la definición del XPDL; dado que existen diferentes librerías que permiten realizar el *parseo*, se elabora una comparación entre estas, de igual manera se evalúa su rendimiento una vez estén implementadas sobre el lenguaje Objective C(ver Fig. 9).

Un *parser* o analizador sintáctico lee el documento XML y comprueba que esté bien escrito, adicionalmente puede ser usado para verificar su estructura[15].

Este *parser* puede ser de tipo SAX(*Simple API for XML*) o de tipo DOM(*DocumentObjectModel*), tal como se determina en [16]:

- SAX: *Parser* en el cual el código es notificado conforme se avanza dentro del árbol XML, y el programador se encarga de mantener en mente el estado y la construcción de cualquier objeto que se quiera conservar mientras el *parser* avanza dentro del archivo.
- DOM: *Parser* que lee todo el documento y construye en memoria una representación en la que se puede hacer un *query* para saber la información dentro de distintos elementos.

Los *parsers* que analizamos en este trabajo y que se definen [16] son los siguientes:

- *NSXMLParser*: Es un *parser* SAX incluido por defecto con el *iPhone* SDK.
- *libxml2*: Es una librería de código abierto que se incluye por defecto con el *iPhone SDK*. Las librerías soportan el procesamiento de *parsers* SAX y DOM. Es capaz de hacer *parser* a los datos a medida que son leídos.
- *TouchXML*: Es un *parser* DOM de estilo *NSXML*. Solo se pueden realizar lecturas de archivos.
- *TBXML*: Es un *parser* DOM ligero diseñado para ser tan rápido como sea posible mientras se consume lo mínimo de recursos de memoria. Salva tiempo porque no realiza validaciones, no soporta XPath y solo se puede realizar lectura de archivos XML.
- *GDataXML*: *Parser* DOM de estilo *NSXML*, desarrollado por *Google,* que permite la manipulación sencilla de objetos dentro del XML.

Como la estructura del XPDL se crea a partir de un modelo XML, se deduce que sus nodos jerárquicos son inalterables, así se hace más fácil la representación de cada elemento como objeto en el lenguaje de programación. De ante mano se sabe que cualquier lenguaje debe soportar la estructura del lenguaje XML, por ello inicialmente la interpretación del documento XPDL con la ayuda

de cualquiera de las librerías anteriores no fue un problema en el caso de la plataforma *iOS*, ya que *iOS* internamente maneja un archivo *PList*[17], que contiene una estructura propia XML, el cual representa cada elemento de la aplicación como un objeto.

B. Selección del parser.

Para la selección del *parser* se generaron una serie de pruebas, primero se decide que debe hacer el *parser* en una parte específica del documento, por ejemplo, una compuerta, como un objeto en el XPDL.

Dentro de las pruebas realizadas se observa que entre más complejos son los archivos, los *parser* SAX no pueden ser utilizados, ya que se necesita leer el documento completamente para sólo buscar un elemento específico.

Ya que el *parser* de tipo SAX, no permite realizar búsquedas internas en el documento, se decide optar por un *parser* de tipo DOM, el cual es más ágil en la búsqueda de elementos y consume menos recursos, además accede a un nodo especifico del árbol en el XPDL.

De los *parsers* de tipo DOM se descartan *TouchXML* y *TBXML*, ya que estos no permiten la escritura de archivos XML y no son útiles para la implementación del diagramador.

Al implementar *GDataXML,* se observa que este *parser* incorpora *libxml2* por defecto y permite realizar tanto la lectura como la escritura de archivos XPDL, así como la búsqueda de elementos que se necesiten dentro del archivo.

Consideradas estas características y las Tablas en las que se hicieron las pruebas de velocidad (ver Fig. 8), se toma la decisión de usar el *parser GDataXML* de tipo DOM, ya que su tiempo de proceso es mejor que la mayoría y además permite la escritura y lectura de archivos XML.

C. Parser XPDL con GDataXML.

Dentro de las clases del *parser GDataXML*, se encuentran implementados métodos que retornan elementos o atributos abstraídos del XPDL.

GdataXML provee dos interfaces, una de ellas es *GDataXMLNode* que contiene métodos privados, los cuales obtienen los nodos del árbol que se genera al leer el archivo XPDL. Otra interface importante es *GDataXMLElement* que no sólo permite crear una instancia que tiene la capacidad de navegar dentro de un *tag* (padre) o un grupo de *tags* permitiendo así la abstracción de atributos propios del nodo padre, sino que también permite crear y modificar un nuevo elemento dentro del XPDL.

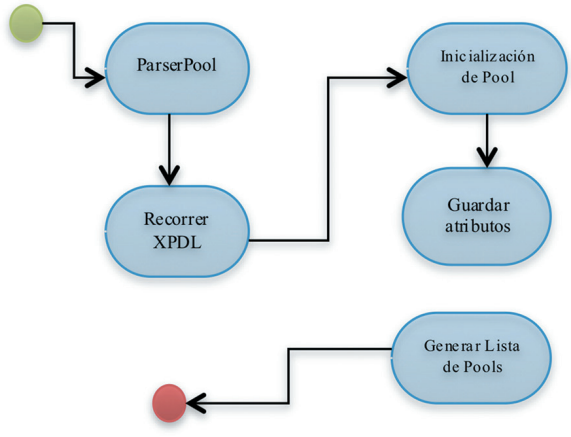
Previo al *parser*, se crean las clases con atributos que representan cada elemento del BPMN, (ver Tabla 1), para esto fue necesario entender el metamodelo del XPDL (ver Fig. 4).

La primera labor del parseador es el manejo de la lectura e interpretación de un documento XPDL, para esto, se crea una clase que contiene el *parser* y se implementa el método que carga el archivo XPDL.

Para estar más seguros de los elementos que maneja el lenguaje XPDL, se crea un diccionario de datos, el cual contiene la definición de los objetos del lenguaje XPDL 2.2; posteriormente se define el XPath [18], este es el encargado de recorrer el árbol jerárquico del documento XPDL, que permite el despliegue de la ruta del objeto a *parsear* y accede a la información de los nodos atributos definidos en dicho documento.

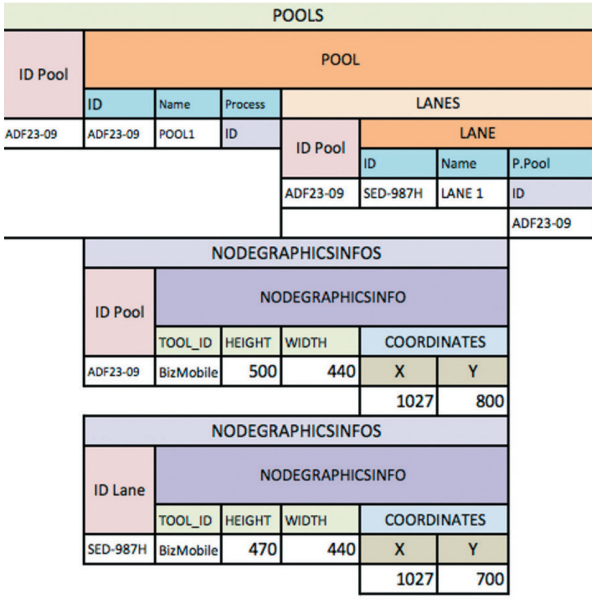
Para realizar el recorrido de cada objeto dentro del documento, se crea un elemento propio del *parser*, el cual por medio de ciclos, abstrae el valor de los atributos tanto del padre como de los hijos en cada nivel del XPDL. Dentro de esta clase se instancian los objetos (color azul) y sus atributos (color negro), que se usan en la representación gráfica del proceso. La siguiente figura describe el proceso de lectura que realiza el *parser* para la abstracción de atributos, de acuerdo al metamodelo del elemento Pool. (Ver Fig. 5).

Fig. 5. PROCESO DE LECTURA DEL PARSER PARA LA GENERACIÓN DEL POOL.



Posterior a la abstracción de los objetos y sus atributos, estos se almacenan dentro de un arreglo que guarda al elemento hijo, este elemento se introduce en otra lista que representa al nodo padre, creando así una lista doblemente enlazada. (Ver Fig. 6).

Fig. 6 ESTRUCTURA LISTA DOBLEMENTE ENLAZADA PARA EL NODO POOLS.



IV. IMPLEMENTACIÓN DE LAS ESTRUCTURAS MULTILISTAS

De acuerdo con la jerarquía de nodos del XPDL y sus respectivos atributos, se realiza un análisis sobre qué tipo de estructura se adaptaría mejor al momento de *parsear* el documento XPDL.

Como primera instancia se opta por implementar árboles de datos, dado que la jerarquía del XML está desarrollada por esta estructura y esto permitiría una fácil abstracción y creación de los objetos.

Con la implementación de los árboles y mediante métodos de conservación para los nodos, en la lectura del documento XPDL no hubo problema alguno. Mientras que en la creación de un nuevo documento, el esquema se creaba con nodos vacíos, es decir, si en el proceso se establecía la creación de un Pool y ningún Lane, el método de generación que recorría el árbol de jerarquías le cargaba al elemento Pool un contenedor Lanes con su respectivo nodo Lane sin que dicho elemento hubiera sido creado; dicho de otro modo, si el nodo es o no un espacio vacío, se toma como nodo significativo, de tal manera que no se cumplía con el estándar del XPDL en su versión 2.2.

Observado lo anterior se decidió por optar por otro tipo de estructura, que concluya con la implementación de listas dobles o multilistas enlazadas, que conectadas entre sí por medio del atributo *id* de cada elemento, permiten una fácil interpretación de los objetos y una apropiada ejecución de los procesos.

Con la adopción de multilistas, el método de búsqueda permite acceder a la información de manera ordenada a través de campos claves, en este caso los nodos *ID*. Las multilistas permiten llegar a un registro por diferentes caminos. El camino lo determina el campo clave sobre el cual se realice la búsqueda [19].

A. Creación de Multilistas

En la Fig. 6 se muestra de manera general, una matriz de objetos Pools, la cual representa la lista doblemente enlazada, esta tiene el objeto *POOLS* como padre, que contiene a *POOL* como hijo, que, a su vez, contiene a *LANES*, al igual que *POOLS*, *LANES* tiene el objeto *LANE* como hijo; que genera así una jerarquía de objetos. Cada hijo está encabezado por un *ID*, el cual es heredado por el padre, que permite la fácil búsqueda, el acceso a dicho objeto y sus respectivos atributos.

Para lograr el *parseo* en la multilista, el primer elemento que se crea es el Pool, dado que es el contenedor de flujo del proceso; la multilista denominada *POOLS* tiene un nodo principal que la identifica llamado *IDPool,* este será el nodo conector de los elementos que están contenidos dentro del Pool.

De igual manera para el caso de los *LANES,* el nodo principal está dado por el *IDPool,* pero para la identificación del objeto se crea un nodo secundario denominado *IDLane,* el cual permite tener acceso a los atributos de este elemento.

Cada elemento, además, está conectado a otra multilista denominada *NODEGRAPHICSINFOS,* que contiene información referente al tamaño, la posición de los elementos y se referencia con el *ID* de cada elemento creado.

Para lograr la creación del *parser,* inicialmente se evalúa la estructura propia del lenguaje estándar XPDL con niveles propios del lenguaje en su

versión 2.2, la cual, agrupa objetos de manera individual con sus atributos, coordenadas y demás especificaciones.

B. Estructuración del XPDL

Tomado como referencia y observada la creación del archivo XPDL generado por *BizAgiModeler®* [20], cabe resaltar que la estructura generada no contiene un orden legible para el programador, ya que ordena los elementos por su posición, es decir, captura el valor de la coordenada X y genera los elementos de manera descendente.

Se realizaron documentos ejemplo de XPDL con cada tipo de elemento de BPMN, en los cuales se observaba la jerarquía y la validez del documento XPDL creados por el modelador *BizAgiModeler*; como resultado se obtuvo el árbol de jerarquías del proceso con sus elementos y atributos (ver Tabla II).

En el siguiente ejemplo se toma el elemento de evento inicio, que bajo el estándar gráfico BPMN se representa por un círculo y su estructura en XPDL (ver Tabla II).

Tabla II

Estructura XPDL evento inicio

```
<WorkflowProcesses>
   <WorkflowProcess Id="66"Name="Proceso principal">
        <Description />
        <ActivitySets />
        <Activities>
             <Activity Id="3f" Name="">
                  <Event>
                       <StartEvent Trigger="None" />
                  </Event>
                  <NodeGraphicsInfos>
                       <NodeGraphicsInfo
                       ToolId="BizMobile"Height="30"
                       Width="30">
                            <Coordinates XCoordinate="55"
                            YCoordinate="130" />
                       </NodeGraphicsInfo>
                  </NodeGraphicsInfos>
             </Activity>
        </Activities>
   </WorkflowProcess>
</WorkflowProcesses>
```

Según los aspectos funcionales anteriormente descritos, el paso siguiente es la codificación y creación del *parseador* XPDL.

V. CONFIGURACIÓN DEL *PARSER* EN *OBJECTIVE C*

El proyecto el cual contiene el *parseador* del XPDL, se crea en el entorno de desarrollo (IDE) de *Objective C*, llamado *Xcode* [21] en su versión 4.3.3.

En la configuración general del proyecto se realizan ciertas modificaciones para que el *parseador* funcione correctamente, en este punto entra Libxml2, una librería que se importa en la clase .h del *parseador* y llama dependencias propias del XML que permite la manipulación de los árboles, nodos y validación de los demás elementos del documento de XML.

Para la validación del documento utilizamos *XPath* [22], que por medio de los *array*, ayuda a fragmentar el documento XPDL dependiendo de la división de los *tag*s (ver Fig. 7).

Los nodos que son abstraídos por el *XPath*, se pueden manipular como elementos individuales según la posición del *Array*; también es posible modificar el archivo XML y mantener las versiones de modificación.

Fig. 7 DECLARACIÓN DEL *XPATH* PARA EL ELEMENTO POOL.

```
NSArray *elementosPools = [doc
nodesForXPath:@"//fb:Pools/fb:Pool"namespaces:ns
error:nil];
```

VI. RESULTADOS OBTENIDOS

Previo a la implementación del *parseador* y su ejecución, se realiza una interpretación completa de un archivo XML para seleccionar de acuerdo a las velocidades, qué tipo de *parser* se va a utilizar para la implementación con el lenguaje estándar XPDL 2.2.

A continuación se muestran las pruebas que se realizaron para la selección del *parser:*

A. Pruebas para la selección del parser

Dado que es primordial medir la velocidad con que el dispositivo móvil ejecuta los procesos en una aplicación, para este caso la primera prueba de selección se basa en la medición de la velocidad de los diferentes *parser*s.

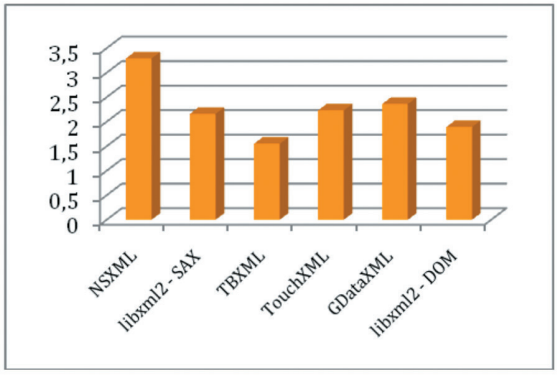
Las figuras que se muestran a continuación dan a conocer cómo interactúan los distintos *parser*s dentro del dispositivo y el tiempo (tomado en segundos) que les toma incluir la información de un archivo XML de 900KB de prueba, que contiene las mejores canciones en *iTune*s[19].

La evaluación se realiza con 10 ejecuciones de las cuales se observan los resultados para tomar

los datos y así generar las tablas de comparación; en la Fig. 8 se muestran los datos de los *parsers*.

Como primera instancia se observa la velocidad del *parser* (ver Fig. 8), de la cual se concluye que el mejor *parser* es *TBXML*, dado que tardó menos tiempo en *parsear* el archivo.

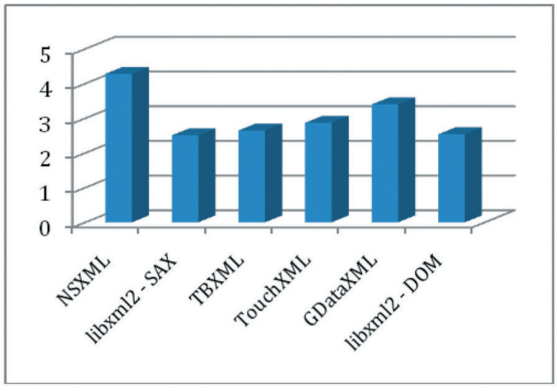
Fig. 8 VELOCIDAD EN SEGUNDOS DEL *PARSER*. PARA *IPOD TOUCH 4G*



En la segunda prueba se realizó una medición del uso de memoria por cada tipo de *parser* (ver Fig. 9), en esta se concluye que el *parser* que menos recursos de memoria consumió fue *Libxml2* de tipo SAX.

Tomados como punto de partida estos datos, se realiza un análisis sobre qué tipo de *parser* implementar; en este punto se tuvo en cuenta la estabilidad del *parser* GDataXML y se evaluó la importancia que este *parser* le da a la lectura y creación de un archivo XPDL.

Fig. 9 USO DE MEMORIA EN MB PARA *IPOD TOUCH 4G*

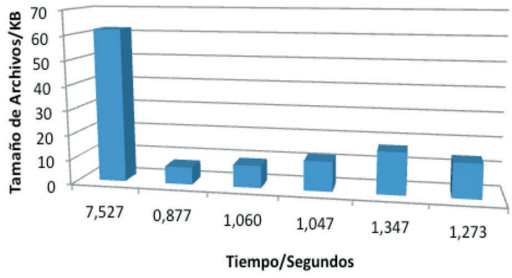


A partir de esto, se genera una serie de pruebas en las que inicialmente se establece qué operaciones debe hacer el *parser,* por ejemplo, abstraer información de una parte específica del documento, como es el caso de un elemento *Activity* dentro del XPDL.

B. Pruebas de la ejecución del parser dentro de Objective C.

En estas pruebas se evaluaron diferentes archivos XPDL variables en su tamaño; observado su comportamiento se obtuvo la siguiente Gráfica (ver Fig. 10), en donde se muestra el tiempo que tardó cada archivo en realizar el *parseo*, recorrer la multilistas y guardar los objetos dentro documento del XPDL.

FIG. 10 PRUEBA DE ARCHIVOS CON EL PARSER XPDL.

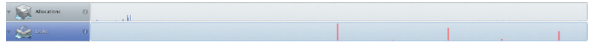


Se concluye así, que sin importar el tipo de XPDL el *parseador* se ejecutará correctamente; la variabilidad del tiempo de ejecución depende del tamaño del archivo y de la cantidad de elementos BPMN contenidos en *él*.

C. Pruebas físicas de fugas

Estas pruebas se encargan de hacer un recorrido general en los métodos para buscar variables que permanezcan en ejecución y muestran su ubicación. En la Fig. 11 se muestran los Bytes vs. Tiempo de ejecución, en la cual las barras representan las variables que quedan en ejecución después de cerrar una escena.

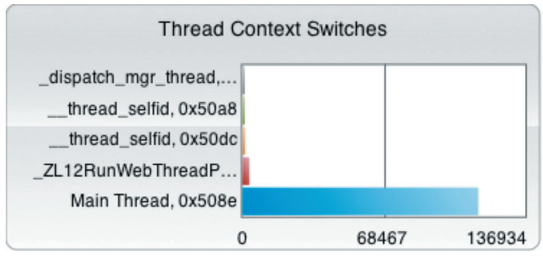
Fig. 11 PRUEBAS DE FUGAS



D. Pruebas físicas de hilos de ejecución.

En estas pruebas se verifica la cantidad de hilos que se ejecutan durante la ejecución normal del programa. (Ver Fig. 12).

Fig. 12 PRUEBAS DE HILOS DE EJECUCIÓN



VII. CONCLUSIÓN Y TRABAJO FUTURO

En esta investigación se plantea una solución para la representación de elementos del BPMN 2.0 [24] recopilados dentro de un archivo creado bajo el lenguaje estándar XPDL en su versión 2.2, que por medio de la implementación de estructuras de datos llamadas multilistas, permite almacenar de manera sencilla y eficiente los valores de cada elemento contenidos dentro del documento XPDL.

Dado que estas estructuras son únicas y complejas, se percibe la necesidad de incorporar un

parser que haga de moderador entre los objetos del BPMN y el lenguaje XPDL.

El procedimiento más relevante de la investigación fue la creación del *parser,* ya que, dentro de *Objective C*, no se ha implementado la generación de este tipo de estructuras. Con *GDataXML*, se soluciona este inconveniente y facilita la lectura y escritura del archivo de diagramación del XPDL.

Con la realización de esta investigación se abordan temáticas poco desarrolladas tanto en el ámbito organizacional como en la parte de implementación y desarrollo para dispositivos móviles.

Para comprobar la eficiencia del *parseador*, se realizaron una serie de pruebas, en donde se compararon archivos de distinto tamaño, los cuales contenían diferente tipo de información. Como resultados obtenidos, se observó la eficiencia y el rendimiento del dispositivo realizando el parseo de los archivos (ver Fig. 10); se concluye que el *parseador* se ejecutará de forma exitosa siempre y cuando el XPDL esté estructurado de manera correcta y sin importar el tamaño del documento. Por otro lado, para probar la resistencia del dispositivo en el momento de la ejecución del *parseador*, se realizaron pruebas físicas, donde se puede comprobar la cantidad de datos que son procesados (ver Fig. 12) y la cantidad de memoria utilizada por la aplicación desde el inicio del *parseo* (ver Fig. 11).

Como resultado satisfactorio, se realizó la inserción y validación del *parseador* dentro del proyecto de grado denominado *Aplicación en entornos móvil para el modelamiento de procesos de negocio* [25], en el cual se implementó el *parseador* de XPDL, permitiendo así, mediante una interfaz amigable, la creación, modificación y modelamiento de procesos de negocio dentro del dispositivo móvil de forma exitosa, innovando en el campo del desarrollo de aplicaciones móviles para la gestión de procesos.

Lo anterior permite apoyar la búsqueda de nuevas tecnologías y soluciones a problemáticas propias de los negocios, e incorporar mejoras en la realización y gestión de procesos desde su planteamiento que pasa por su distribución a cada uno de los miembros del grupo de trabajo y finaliza con la ejecución del proceso, beneficia a las personas participes de este, incluido, por supuesto, el usuario final.

De igual forma, y como trabajo futuro, se pueden realizar implementaciones que mejoren el desempeño de la presente implementación en el ámbito del desarrollo de *software* y optimización de código y la capacidad de auto generar código para el *parseador* basado en el esquema y las reglas propias de una estructura XSD.

REFERENCIAS

[1] XPDL, Welcome to XPDL.org. Noviembre 2011. [Online]. Disponible en: http://www. xpdl.org/

[2] XML, Extensible Markup Language (XML). Julio 2012. [Online]. Disponible en: http://www. w3.org/XML/.

[3] BPMN, Documents Associated with Business Process Model and Notation (BPMN) Version 2.0. Julio 2012. [Online]. Disponible en: http://www.bpmn.org/.

[4] BPMN Elements and Attributes V4. Julio 2012. [Online]. Disponible: http://www.omg.org/bpmn/ Documents/BPMN_Elements_and_Attributes.pdf

[5] Process Definition Interface- XML Process Definition Language, Meta Model. Workflow Management Coalition (WfMC). Julio 2012. [Online]. Disponible en :http://www.wfmc.org/.

[6] R. Wenderlich. How To Choose The Best XML Parser for Your iPhone Project. Julio 2012. [Online]. Disponible en: http://www.raywenderlich.com/553/how-to-chose-the-best-xml-parser-for-your-iphone-project.

[7] Gdata-objectivec-client. Google Data APIs Objective C Client Library. Julio 2012. [Online].Disponible en: http://code.google.com/ p/gdata-objectivec-client/.

[8] Apple. Julio 2012. [Online]. Disponible en: http://www.crunchbase.com/company/apple.

[9] Workflow Management Coalition (WfMC). Noviembre 23 2011. [Online]. Disponible en: http://www.wfmc.org/ .

[10] Jon Puke. XPDL – The silent Workhouse of BPM P1.Noviembre 23 2011. [Online]. Disponible en: http://www.wfmc.org/Published-Research/Article/View-category.html.

[11] BPD, Catalog of Business Modeling and Management Specifications. Noviembre 23 2011. [Online]. Disponible en: http://www.omg. org/technology/documents/br_pm_spec_catalog.htm

[12] Process Definition Interface- XML Process Definition Language, Package Meta Model. Workflow Management Coalition (WfMC). P 14.Julio 2012. [Online]. Disponible en :http:// www.wfmc.org/.

[13] Appian, Appian BPM Suite: Mobiles. Noviembre 23 2011. [Online]. Disponible en: http://www.appian.com/bpm-software/bpm-components/mobile-bpm.jsp.

[14] S. A. White, XPDL and BPMN, Future Strategies Inc. WFMC, P 222. Octubre 26 2011. [Online].Disponible en: http://www.bpmn.org/ Documents/XPDL_BPMN.pdf.

[15] R. Wenderlich. How To Choose The Best XML Parser for Your iPhone Project. Disponible en: http://www.raywenderlich.com/553/how-to-chose-the-best-xml-parser-for-your-iphone-project. Noviembre 25 2011.

[16] Apple Inc. Plist Mac OS X. Noviembre 30 2011. [Online]. Disponible en: http://developer. apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man5/plist.5.html.

[17] Apple Inc. Plist Mac OS X Noviembre 30 2011. [Online]. Disponible en: http://developer. apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man5/plist.5.html.

[18] W3School. XPath Tutorial. Agosto 2011. [Online].Disponible en: http://www.w3schools. com/xpath/

[19] F. Roberto. Algoritmos, estructuras de datos. Programación orientada a objetos. Ecoe ediciones 2005. Bogotá p. 273.

[20] BizAgi, BizAgiModeler. Noviembre 25 2011. [Online]. Disponible en: http://www.bizagi.com/

[21] Xcode. Developer tolos. Julio 2012 [Online]. Disponible en: https://developer.apple.com/ technologies/tools/.

[22] R. Wenderlich. How To Choose The Best XML Parser for Your iPhone Project. Noviembre 2011. [Online]. Disponible en: http://www. raywenderlich.com/553/how-to-chose-the-best-xml-parser-for-your-iphone-project.

[23] Apple, iTunes. Diciembre 2 2011. [Online]. Disponible en: http://www.apple.com/es/itunes/.

[24] BPMN. OMG, Object Managment Group. Business Process Modeling Notation. Agosto 2011. [Online].http://www.bpmn.org/

[25] D. Meza, L. Ruiz. Aplicación en entornos móvil para el modelamiento de procesos de negocio. Universidad Piloto de Colombia. Julio 2012.