Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Tesis de Maestría
en Ingeniería en Computación

Framework para Acceder a Bases de Datos Relacionales a Través de Internet

Ing. Sandro Moscatelli

Supervisor Ph.D. Raul Ruggia

Setiembre 2002

Resumen

En esta tesis se presenta un `framework` que permite, a aplicaciones cliente, realizar consultas a bases de datos relacionales a través de `internet`, tendiendo a hacer transparente el diseño lógico de la base de datos.

El núcleo del `framework` está formado por la *API ANS* (**A**PI **N**úcleo del **S**istema) que permite el acceso a las bases de datos relacionales, brindando flexibilidad para la realización de consultas mediante el uso de lenguajes de consulta apropiados y utilizando *XML* como formato para el intercambio de información, desde y hacia la aplicación cliente.
El `framework` se basa en el protocolo *SOAP* (**S**imple **O**bject **A**ccess **P**rotocol) para la comunicación, a través de `internet`, entre las aplicaciones cliente y la *API ANS*.

Para lograr una mayor transparencia del diseño lógico de la base de datos se propone que las consultas se realicen sobre vistas *XML* de la base de datos, las que son especificadas mediante *XML Schemas*, utilizando un lenguaje de consulta *XML*, por lo cual se presenta el caso concreto de la utilización del lenguaje de consulta $XPath_{Sub}$, subconjunto del lenguaje *XPath*.

Por último se presenta la definición formal de la semántica de $XPath_{Sub}$ usando semántica denotacional y se especifica un algoritmo que transforma consultas $XPath_{Sub}$, realizadas sobre la vista *XML* de la base de datos a expresiones en álgebra relacional, basado en la semántica de $XPath_{Sub}$.

Palabras clave
SOAP, XML, XML Schema, XPath, vista XML, SQL, álgebra relacional.

Agradecimientos

Esta tesis fue realizada con la ayuda de un grupo de personas a las cuales quiero expresar desde aquí mi más profundo agradecimiento.

Quiero agradecer a Raúl Ruggia, quien me alentó a realizar esta maestría y posteriormente aceptó ser el supervisor de esta tesis.

Agradezco muchísimo la colaboración de Fernando Carpani por su disposición a discutir conmigo algunos aspectos fundamentales de esta tesis y por el interés mostrado en la misma.
De la misma manera quiero agradecer a Alberto Pardo por su colaboración.

Agradezco a todos los integrantes del C.P.A.P. y especialmente a su directora, Nora Szasz, la oportunidad que me brindaron de realizar esta maestría.

Agradezco finalmente a Verónica y a Martín por su apoyo incondicional y por la enorme paciencia mostrada durante el proceso de desarrollo de esta tesis.

Índice

Lista de Figuras

Capítulo I Introducción

1.1 Descripción y objetivos

A lo largo del tiempo las empresas han adoptado en forma creciente el uso de sistemas manejadores de bases de datos, fundamentalmente relacionales, para almacenar sus datos de interés. En el ámbito de la empresa los datos son accedidos por aplicaciones cliente, normalmente, a través de una intranet. Actualmente, debido al auge de internet, muchas empresas desean exponer sus datos, para que sean consultados por aplicaciones cliente remotas, a través de internet.

En esta tesis se analiza el problema de como las aplicaciones cliente pueden consultar bases de datos relacionales a través de internet, tendiendo a hacer transparente el diseño lógico de las bases de datos.

Para resolver el problema anterior se propone un framework con las siguientes características:

- Construido sobre estándares abiertos de internet: *SOAP* y *XML*
- Brindar la posibilidad de abstraer a las aplicaciones cliente del diseño lógico de la base de datos (transparencia del diseño lógico).
- Brindar flexibilidad para la realización de consultas.

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) es un protocolo de comunicación, basado en *XML* (e**X**tensible **M**arkup **L**anguage), que se está transformando rápidamente en un estándar a nivel de internet para comunicar aplicaciones, debido a que es independiente de plataformas y lenguajes de programación lo cual permite la interoperabilidad entre aplicaciones heterogéneas y el desarrollo de sistemas con un bajo acoplamiento, no requiriéndose que en el cliente se encuentren instalados componentes de software dependientes de los componentes de software instalados en el servidor.
Esto significa que las aplicaciones cliente pueden utilizar lenguajes de programación y plataformas distintas a las utilizadas en el ámbito de la empresa donde se encuentra el servidor de bases de datos.

XML puede considerarse actualmente como el estándar a nivel de internet para el intercambio de información. Porque al ser independiente de plataformas y lenguajes de programación es útil en una amplia variedad de áreas, fundamentalmente cuando intervienen sistemas heterogéneos y se requiere interoperabilidad entre los mismos, que son las carácterísticas de internet.

Para brindar una mayor transparencia del diseño lógico de la base de datos se utilizan vistas de la base de datos especificadas mediante *XML Schemas*. Uno de los aspectos relevantes es la utilización de vistas *XML*, que permiten ocultar el diseño lógico relacional, y la realización, por parte de las aplicaciones cliente, de consultas sobre estas vistas usando un lenguaje de consulta *XML*, con lo cual se logra que las aplicaciones cliente sean más homogéneas, en lo relativo a no mezclar, en la misma aplicación, distintos lenguajes: la representación de los datos (*XML*) y *SQL* como lenguaje de consulta (relacional), evitando potenciales complicaciones en su programación.

La Figura 1 muestra la arquitectura, simplificada, del framework. El núcleo del framework lo constituye la *API* (**A**pplication **P**rogram **I**nterface) *ANS* (**A**PI **N**úcleo del **S**istema) que permite el acceso a las bases de datos relacionales. *ANS* provee dos primitivas cuya interface está basada en *XML*, tanto para recibir como para enviar datos. Al utilizar *XML* para el intercambio de información, *ANS* es, también, independiente de plataformas y lenguajes de programación.

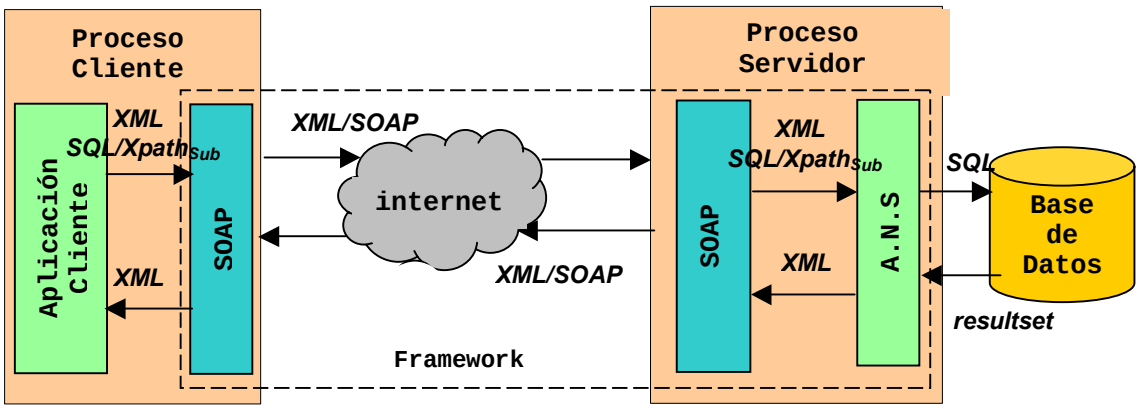


Figura 1 – Arquitectura simplificada del framework

ANS tiene las siguientes características funcionales:

- Permite que las aplicaciones cliente obtengan las especificaciones de las vistas *XML* creadas sobre las bases de datos relacionales o bien la estructura relacional correspondiente a una vista o a la base de datos.
- Utilizar un lenguaje de consulta para que las aplicaciones cliente realicen sus consultas (*SQL* o un lenguaje de consulta *XML*).
- Procesa las consultas del cliente. En caso de no estar escritas en *SQL*, las convierte a *SQL* para su resolución por parte del manejador de la base de datos relacional.
- Convierte los conjuntos de tuplas (`resultset`) obtenidos como resultado de la consulta *SQL* a formato *XML* a los efectos de retornarlos a la aplicación cliente.

El `framework` propuesto en esta tesis representa una solución general que puede ser adaptado a distintas necesidades, en particular en lo relativo al lenguaje de consulta *XML* que utilizan las aplicaciones cliente.
En consecuencia se presenta, como parte del `framework`, un posible lenguaje de consulta *XML*, denominado $Xpath_{Sub}$, que es un subconjunto funcional del lenguaje *XPath*. La elección de *XPath*, como lenguaje de base para realizar las consultas, se debe a varias razones entre las que se destacan:

- *XPath* es una recomendación oficial del W3C (**W**orld **W**ide **W**eb **C**onsortium).
- Las consultas *XPath* son compactas y simples.
- Las consultas *XPath* son fáciles de escribir y de leer.
- Actualmente existen muchas implementaciones de *XPath*, lo cual demuestra que es uno de los lenguajes más difundidos y de más amplio uso por parte usuarios y aplicaciones.

Finalmente se define un algoritmo que establece la correspondencia entre $Xpath_{Sub}$ y álgebra relacional a los efectos de especificar un algoritmo de conversión a *SQL*, dado que en la implementación del `framework` se utiliza *SQL* para acceder a la base de datos.
Las aplicaciones cliente que deseen consultar una base de datos de una empresa, utilizando el lenguaje de consulta $Xpath_{Sub}$ se comunican usando el protocolo *SOAP*, con las primitivas de *ANS*, para obtener, primeramente, el *XML Schema* que especifica la vista *XML* de la base de datos y, posteriormente, para enviar las consultas escritas en $Xpath_{Sub}$.

ANS convierte la consulta $Xpath_{Sub}$ a *SQL*, basándose en el algoritmo definido, obtiene los datos de la base de datos y los transforma a formato *XML*, para su posterior envío a la aplicación cliente.

En resumen, las contribuciones de esta tesis son las siguientes:

- Un `framework` general, incluyendo la especificación y diseño de una *API* (*ANS*), el cual brinda una solución concreta al problema de como las aplicaciones cliente pueden consultar bases de datos relacionales a través de `internet`, tendiendo a hacer transparente el diseño lógico de las bases de datos, integrando tecnologías `open source` sumamente recientes que actualmente son, o que pronto llegarán a ser, estándares.

- Especificación de la semántica formal del lenguaje $Xpath_{Sub}$ y de un algoritmo para transformar consultas $Xpath_{Sub}$ a álgebra relacional. Por lo que está en nuestro conocimiento no existen estudios previos en este sentido (por lo menos que sean públicos), en lo relativo a la especificación de un algoritmo que transforma consultas expresadas en un lenguaje de consulta *XML* a expresiones de álgebra relacional. En cambio existen algoritmos (en su mayoría no públicos) que convierten consultas expresadas en un lenguaje de consulta *XML* a *SQL* directamente, como se indica en la sección 2.7. Sin embargo la generación de expresiones en álgebra relacional tiene claras ventajas frente a la generación directa de *SQL*:

    - Permite una clara separación de la especificación del algoritmo de su implementación, dado que el álgebra relacional es un lenguaje relacional abstracto sin elementos de implementación. Esto permite una asociación más clara entre las expresiones de ambos lenguajes ($Xpath_{Sub}$ y álgebra relacional).
    - Álgebra relacional es el lenguaje estándar de referencia para el modelo relacional.
    - Las expresiones de álgebra relacional pueden descomponerse de manera simple y clara en subexpresiones (lo cual no es tan simple en *SQL*) lo cual facilita el mapeo de expresiones $Xpath_{Sub}$ a subexpresiones de álgebra relacional para la posterior construcción de la expresión completa.
    - La especificación es válida para cualquier versión de *SQL*, de cualquier manejador de base de datos, y no para un *SQL* particular.
    - Se obtiene un control más fino entre la generación y la optimización. Al utilizar álgebra relacional el énfasis está en la generación de la expresión y no en la optimización, que puede ser realizada a posteriori. En cambio si la especificación se realizara directamente en *SQL* se deberían realizar, tanto la generación como la optimización, en un solo paso.

Este informe está organizado de la siguiente manera. En el capítulo 2 se presentan las tecnologías que sirven de base para la presente tesis, en particular el protocolo *SOAP*; *XML*; *XML Schemas* y *XPath*.
En el capítulo 3 se presenta el `framework` en su forma general y se definen las reglas para generar la vista *XML* de la base de datos relacional. En el capítulo 4 se presenta el lenguaje de consulta $Xpath_{Sub}$ (gramática y semántica formal) y el algoritmo de conversión a álgebra relacional.
En el capítulo 5 se presenta la implementación realizada del `framework`. En el capítulo 6 se presentan las conclusiones y, finalmente, en el capítulo 7 los trabajos futuros.

Capítulo II Estado del Arte

2.1 Introducción

En este capítulo se presentan las principales tecnologías que constituyen la base de esta tesis, en particular el protocolo *SOAP*, el lenguaje *XML*, *XML Schemas*, el lenguaje de consulta *XPath* y la *API XML for Analysis*.

SOAP es una tecnología emergente que consiste en un protocolo para comunicar aplicaciones o componentes de software distribuidos heterogéneos, mediante la combinación de tecnologías open source (*XML* y *HTTP*) de amplia aceptación en la industria de software.

XML es un lenguaje de marcas que define un conjunto de reglas que permiten estructurar datos en lo que se denomina un documento *XML*.

Los *XML Schema* son un mecanismo para definir un modelo para una clase de documentos, el cual describe la estructura, contenido y semántica de los documentos de la clase. También pueden verse como un acuerdo sobre un vocabulario común, cuando se realiza un intercambio de documentos, entre aplicaciones, por ejemplo.

XPath es un lenguaje que permite navegar por un documento *XML* y seleccionar partes del mismo.

XML for Analysis es la especificación de una *API* (**A**pplication **P**rogram **I**nterface) para intercambiar datos multidimensionales entre una aplicación cliente y un servidor, independiente de plataformas y lenguajes, basada en *XML* y *SOAP*.

2.2 SOAP

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) es un protocolo que permite comunicar aplicaciones o componentes de software distribuidos heterogéneos permitiendo el desarrollo de sistemas con un bajo acoplamiento como se requiere en el ámbito de internet. Este protocolo se basa en *XML*, que es un estándar basado en texto e independiente de plataformas y lenguajes y en *HTTP* (**H**yper **T**ext **T**ransfer **P**rotocol), entre otros, como transporte.

2.2.1 Historia de SOAP

En el año 1997 Microsoft comenzó a investigar, junto con las empresas DevelopMentor y Userland, la posibilidad de utilizar *XML*, sección 2.3, como base para la computación distribuida, con el objetivo de lograr la comunicación entre aplicaciones mediante invocaciones *RPC* (**R**emote **P**rocedure **C**all) embebidas en *HTTP*.
Esta investigación sufrió ciertos retrasos, y como consecuencia la empresa Userland publicó en el verano de 1998 una versión propia de un protocolo basado en *XML* para realizar invocaciones *RPC*, denominado *XML-RPC* [1].

En el año 1999 la investigación de las empresas mencionadas retomó nuevamente impulso y a partir de la especificación del protocolo *XML-RPC*, fue desarrollada la versión 0.9 de *SOAP* que aparece públicamente en setiembre de 1999 y es enviada a la IETF (**I**nternet **E**ngineering **T**ask **F**orce). La versión 1.0 de *SOAP* fue publicada a continuación, en diciembre del mismo año.

En mayo de 2001 la versión 1.1 de *SOAP* [2] fue enviada al W3C con la empresa IBM como coautora, entre muchas otras de empresas de software que participaron en el desarrollo de dicha versión. En esta versión fueron eliminados aspectos que eventualmente pudiesen estar relacionados con tecnologías propietarias de Microsoft.

El hecho que varias empresas de relevancia se sumaran a la especificación de la versión 1.1 de *SOAP*, implicó que el mismo fuera tomado en consideración rápidamente, por parte de la industria de `software` y comenzaran a aparecer las primeras implementaciones del mismo.

En el año 2000 el W3C anuncia su intención de crear un grupo de trabajo en el área de los protocolos basados en *XML*, lo cual se concreta en setiembre de 2000. Este grupo empezó su trabajo con la versión 1.1 de *SOAP* y produjo una nota con la versión 1.2 en julio de 2001 [3]. Este grupo continúa actualmente su trabajo produciendo nuevas notas de la versión 1.2 de *SOAP*, la último de las cuales es de diciembre de 2001 [4].

2.2.1 Qué es SOAP

SOAP [3] [4] [5] es un protocolo que permite comunicar aplicaciones o componentes de software en un entorno distribuido o descentralizado.
La filosofía de *SOAP*, para lograr dicha comunicación, consiste en combinar tecnologías existentes y de amplia aceptación en la industria de software. En particular, utiliza *XML* para la codificación de los mensajes y un protocolo de transporte, *HTTP* por ejemplo, para transportar dichos mensajes. *SOAP* no mandata el uso de un protocolo de transporte particular, aunque *HTTP* es el más popular.

SOAP es independiente de plataformas o lenguajes de programación, lo cual permite que las aplicaciones que se intenta comunicar puedan estar ejecutándose en cualquier plataforma y estar escritas en cualquier lenguaje de programación, siempre y cuando tengan la capacidad de formular y entender los mensajes *SOAP*. Por este motivo, *SOAP* es sumamente útil para permitir la interoperabilidad de aplicaciones distribuidas, fundamentalmente en `internet` pero también en `intranets`.

SOAP es un protocolo de comunicación simple y extensible basado en mensajes, que permiten el intercambio de información estructurada y tipeada, entre aplicaciones en un entorno distribuido. Para lo cual provee:

- Un mecanismo para definir la unidad de comunicación. En *SOAP* toda la información se empaqueta en un mensaje *SOAP*. Un mensaje es un documento XML definido mediante la construcción *ENVEL0PE* (sección 2.2.3).

- Un conjunto de reglas para codificar los datos de los mensajes *SOAP* en *XML* (sección 2.2.5).

- Una convención para representar invocaciones *RPC* y sus correspondiente respuestas en mensajes *SOAP*, debido a que *RPC* es la forma más común de interacción entre aplicaciones distribuidas (sección 2.2.6).

- Una convención para el uso de un protocolo de transporte (sección 2.2.4).

2.2.2 Protocolo SOAP

Los mensajes *SOAP* se codifican en *XML*, lo cual tiene las siguientes ventajas:

- *XML* permite representar datos de manera independiente de plataformas y lenguajes.
- *XML* se está transformando rápidamente en un estándar al ser ampliamente aceptado por la industria de `software`.
- Está disponible en todas las plataformas.
- Es adecuado para manipular datos estructurados y tipeados.
- Es fácil de entender y analizar, tanto para las computadoras como para las personas, por ser texto.

En cuanto al transporte de los mensajes *SOAP*, en la versión 1.0 se establecía como obligatorio el uso del protocolo *HTTP*, sin embargo a partir de la versión 1.1 [2] se eliminó la obligatoriedad de utilizar *HTTP*, permitiéndose la utilización de otros protocolos de transporte como ser *FTP* (**F**ile **T**ransfer **P**rotocol), *SMTP* (**S**imple **M**ail **T**ransfer **P**rotocol), etc.
A pesar de que no se mandata el uso de un protocolo de transporte particular, se define una convención de como se realiza el transporte usando *HTTP* (sección 2.2.4).
El uso del protocolo de transporte *HTTP* tiene las siguientes ventajas:

- Es el protocolo estándar para la comunicación en `internet`.
- Está disponible para todas las plataformas.
- Requiere muy poco soporte en tiempo de ejecución para funcionar adecuadamente.
- Es un protocolo que generalmente no es bloqueado por `firewalls` (en las secciones 2.2.4 y 2.2.7 se trata este aspecto con más detalle)

SOAP no define ningún mecanismo de interacción particular entre las aplicaciones participantes, por lo cual pueden implementarse distintos patrones de interacción, desde patrones de una única vía entre un emisor y un receptor, hasta patrones de requerimiento/respuesta. A su vez los mensajes pueden ser sincrónicos o asincrónicos.

Uno de los usos más populares de *SOAP* es para realizar invocaciones *RPC*. Este tipo de interacción se basa en la utilización del patrón requerimiento/respuesta, generalmente mediante mensajes sincrónicos.
El uso del protocolo de transporte *HTTP* facilita este tipo de interacción, dado que los mensajes *SOAP* siguen el modelo propio de *HTTP*, o sea el requerimiento es enviado en un mensaje `request` de *HTTP* y la respuesta es recibida en un mensaje `response` de *HTTP*.
SOAP define una convención para la representación y codificación de los nombres de los métodos y parámetros, así como para los valores de retorno (sección 2.2.6).

Cuando los mensajes no corresponden a invocaciones *RPC*, el protocolo *SOAP* no define ni convenciones ni restricciones, por lo cual cualquier información puede ser incluida en un mensaje *SOAP*.

2.2.3 Estructura de los mensajes SOAP

SOAP utiliza mensajes para la comunicación entre aplicaciones. Un mensaje es un documento *XML*, el cual consiste en el elemento *ENVELOPE* (obligatorio), que contiene toda la información del mensaje.

Dentro del elemento *ENVELOPE* se encuentran los elementos *HEADER* (opcional) y *BODY* (obligatorio). La figura 2 muestra la estructura de un mensaje *SOAP*:

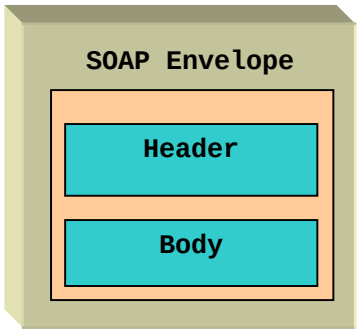


Figura 2 – Estructura de mensajes SOAP

2.2.3.1 SOAP Envelope

El elemento *ENVELOPE* define los mensajes *SOAP* y contiene la información del mismo. Todo mensaje *SOAP* debe contener este elemento y éste debe ser el primer elemento del documento *XML* que representa el mensaje.

En particular el nombre del elemento debe ser *ENVELOPE* y puede contener declaraciones de `namespaces` (sección 2.3), que son globales al mensaje, así como atributos. Si estos atributos están presentes, deben estar calificados por un `namespace`.
Entre los atributos que pueden aparecer, uno de los más importantes es:

- `encodingStyle`: es usado para indicar las reglas de codificación usadas en el mensaje para codificar los datos en *XML*.

En la figura 3 se muestra un ejemplo de un *ENVELOPE SOAP*:

```
<SOAP-ENV:Envelope

  xmlns:SOAP-ENV ="http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV: encodingStyle="http://www.w3.org/2001/12/soap-encoding">

</SOAP-ENV:Envelope>
```

Figura 3 - SOAP Envelope

2.2.3.2 SOAP Header

El elemento *HEADER* es opcional. En caso de estar presente debe ser el primer hijo del elemento *ENVELOPE*.

El nombre del elemento debe ser *HEADER* y puede tener una cantidad arbitraria de elementos hijo, cada uno de los cuales se denomina entrada o bloque. Cada entrada se identifica por un nombre local y por un `namespace` y puede tener atributos. Entre los atributos que pueden aparecer, uno de los más importantes es:

- `mustUnderstand`: es usado para indicar al receptor del mensaje si debe procesar, de manera obligatoria (valor 1) la entrada, o bien si la entrada (valor 0) es opcional. La ausencia de este atributo en una entrada es equivalente a que la entrada es opcional.

La finalidad principal del elemento *HEADER* es proveer un mecanismo genérico de extensibilidad de los mensajes *SOAP* de una manera descentralizada y sin conocimiento previo de las partes involucradas, es decir sin modificar la estructura básica de los mensajes. El elemento *HEADER* puede contener datos de autenticación, autorización, manejo transaccional, trazabilidad o auditoria por ejemplo.

Por ejemplo, en una transferencia de pagos son necesarios los números de cuenta involucrados y el monto de la transferencia, sin embargo el requerimiento de una transferencia, usualmente, contiene mucha más información como ser la identidad de la persona que realiza el requerimiento, la prioridad de la transferencia, etc. Esta información adicional es usualmente manejada por otros servicios que intervienen en la transferencia (servicio de `login`, servicio de coordinación de transacciones, etc.) y no por el propio servicio de transferencia.
Esta información adicional puede ser pasada en el elemento *HEADER*, como se muestra en la figura 4:

```
<SOAP-ENV:Header>

     <t:Transaction
        xmlns:t= “http://example.org/2001/12/tx”
        SOAP-ENV:mustUnderstand="1">
             5
      </t:Transaction>
      <p:Priority
        xmlns:t= “http://example.org/2001/12/tx”>
             1
      </p:Priority>

</SOAP-ENV:Header>
```

Figura 4 – SOAP Header

2.2.3.3 SOAP Body

El elemento *BODY* es obligatorio en los mensajes *SOAP* y debe ser un elemento hijo inmediato del elemento *ENVELOPE* y debe estar a continuación del elemento *HEADER*, si éste está presente.
El nombre del elemento debe ser *BODY* y puede tener una cantidad arbitraria de elementos hijo. Cada elemento hijo se denomina entrada o bloque y se identifica por un nombre local y por un `namespace`.

El elemento *BODY* actúa como contenedor para la información que se envía al receptor del mensaje.
Un uso típico del elemento *BODY* es realizar invocaciones *RPC*, o bien para recibir la respuesta de una invocación *RPC*. Un ejemplo de la utilización del elemento *BODY* para realizar invocaciones *RPC* se muestra en la figura 5:

```
<SOAP-ENV:Body>

  <m1:RemoteMethodName_1
        xmlns:m1="http://example/2001/12/m1_method">
     <arg1>valor1</arg1>
     <arg2>valor2</arg2>
  </m1:RemoteMethodName_1>

  <m2:RemoteMethodName_2
        xmlns:m2=" http://example/2001/12/m2_method ">
     <arg1>valor1</arg1>
   </m2:RemoteMethodName_2>

</SOAP-ENV:Body>
```

Figura 5 – SOAP Body

SOAP define una convención para representar invocaciones *RPC* y sus correspondiente respuestas en mensajes *SOAP* que se detalla en la sección 2.2.6.

SOAP define una entrada particular en el elemento *B0DY*, que es la entrada *FAULT*, usada para reportar errores. Esta entrada tiene cuatro subelementos hijos, de los cuales los mas importantes son:

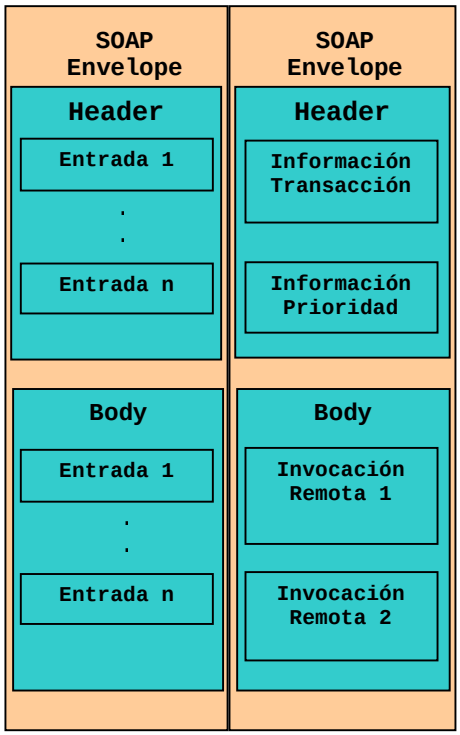
- `faultcode`: este subelemento debe estar presente en el elemento *FAULT* y consiste en un código de error. En la especificación se definen ciertos valores estándar de código de error.

- `faultstring`: este subelemento debe estar presente en el elemento *FAULT* y provee información, en un formato legible para personas, sobre el error generado.

2.2.3.4 Mensajes SOAP

En resumen, un mensaje *SOAP* es un documento *XML* que consta de un elemento de nombre *ENVELOPE*. Este elemento tiene, opcionalmente, un elemento hijo de nombre *HEADER* y obligatoriamente un elemento hijo de nombre *BODY*.

Los elementos *HEADER* y *BODY* pueden tener, cada uno de ellos, una cantidad arbitraria de elementos hijos, que se denominan entradas o bloques. La figura 6 muestra un ejemplo de la estructura de un mensaje *SOAP*:




```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV ="http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV: encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <SOAP-ENV:Header>

      <t:Transaction
          xmlns:t= "http://example.org/2001/12/tx"
          SOAP-ENV:mustUnderstand="1">
               5
       </t:Transaction>
       <p:Priority
            xmlns:t= "http://example.org/2001/12/tx">
              1
       </p:Priority>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>

    <m1:RemoteMethodName_1
        xmlns:m1="http://example/2001/12/m1_method">
     <arg1>valor1</arg1>
     <arg2>valor2</arg2>
    </m1:RemoteMethodName_2>

    <m2:RemoteMethodName_2
        xmlns:m2=" http://example/2001/12/m2_method">
     <arg1>valor1</arg1>
     </m2:RemoteMethodName_2>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 6 – Mensaje SOAP

2.2.4 SOAP y HTTP

SOAP no mandata el uso de un protocolo de transporte particular, por lo cual se puede utilizar cualquier protocolo de transporte. Esto es debido a que los mensajes *SOAP*, en cuanto a su contenido *XML*, son independientes del protocolo que los transporta.
Debido a *HTTP* es el protocolo de transporte más popular en `internet`, *SOAP* describe una convención a usar cuando el transporte se realiza mediante *HTTP*.

Cuando se utiliza *HTTP* como protocolo de transporte, *SOAP* sigue el modelo de mensajes `request` y `response` de *HTTP*. Para realizar el `request` pueden usarse distintos métodos pero el más común es el *POST*.

Un mensaje `request` de *SOAP* incluye un cabezal *HTTP* que se encuentra antes del mensaje *SOAP*, propiamente dicho, este cabezal *HTTP* es similar a un cabezal *HTTP* típico como se detalla a continuación:

En la primer línea se especifica el método de envío, la *URI* (**U**niform **R**esource **I**dentifier) requerida y la versión del protocolo usada:

```
POST /ProductCatalog HTTP/1.1
```

En la siguiente línea se específica el destino:

```
HOST: www.mywebsite.com
```

Las siguientes 3 líneas son usadas para definir el formato *MIME* para desplegar el mensaje, la codificación *HTTP* y el largo del mensaje:

```
Content-Type: text/xml;
charset="utf-8"
Content-Length: nn
```

A continuación se encuentran los cabezales *SOAPACTION*, esta parte del cabezal *HTTP* es la que difiere de otros cabezales típicos *HTTP*, que indican cual es la intención del mensaje. La sintaxis de este cabezal es la siguiente:

```
soapaction = "SOAPACTION"  ":"  [ <"> URI_reference <">]
```

donde `URI_reference` es una referencia a una *URI* que se encuentra definida en la *RFC 2396* [6]

Como casos particulares de la `URI_reference` son:

- el `string` vacío (""), que indica que se intenta acceder a la *URI* especificada en el campo *HOST*, por ejemplo:

```
SOAPAction:""
```

- sin valor, que indica que no se tiene información sobre cual es la intención del mensaje, por ejemplo:

```
SOAPAction:
```

El cabezal *SOAPACTION* puede ser utilizado por los `firewalls` o servidores para filtrar los mensajes que llegan, o bien realizar procesamientos condicionales, basados en el contenido de este cabezal.

En cuanto al mensaje `response`, *SOAP* sigue la semántica de los códigos de status de *HTTP*. Por ejemplo un código de status *2nn* indica que el `request` del cliente fue recibido de manera exitosa.

En la figura 7 se presenta el mensaje `request` *SOAP* completo, correspondiente a los ejemplos que se han mostrado en las secciones anteriores:

```
POST /ProductCatalog HTTP/1.1
HOST: www.mywebsite.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nn
SOAPAction: "http://example/2001/12/m1_method"
SOAPAction: "http://example/2001/12/m2_method"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV ="http://www.w3.org/2001/12/soap-envelope"
 SOAP-ENV: encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <SOAP-ENV:Header>
    <t:Transaction
       xmlns:t= "http://example.org/2001/12/tx"
       SOAP-ENV:mustUnderstand="1">
            5
    </t:Transaction>
    <p:Priority
        xmlns:t= "http://example.org/2001/12/tx">
            1
    </p:Priority>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
  <m1:RemoteMethodName_1
      xmlns:m1="http://example/2001/12/m1_method">
   <arg1>valor1</arg1>
   <arg2>valor2</arg2>
  </m1:RemoteMethodName_1>
  <m2:RemoteMethodName_2
      xmlns:m2="http://example/2001/12/m2_method">
   <arg1>valor1</arg1>
  </m2:RemoteMethodName_2>
</SOAP-ENV:Body>
```

Figura 7 – SOAP request usando HTTP

el mensaje `response` correspondiente se muestra en la figura 8:

```
HTTP/1.1 200 OK
HOST: www.mywebsite.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nn

<SOAP-ENV:Envelope
  ...
</SOAP-ENV:Envelope>
```

Figura 8 – SOAP response usando HTTP

2.2.5 Reglas codificación (Encoding Rules)

Las reglas de codificación son usadas para determinar como se realiza la codificación de los datos de los mensajes *SOAP* en *XML*.
Para especificar las reglas de codificación a usar en un mensaje se utiliza el atributo `encodingStyle`. Este atributo puede ser global a todo el mensaje (atributo del elemento *ENVELOPE*) o bien puede especificarse una codificación particular para cada elemento del mensaje. De esta manera cualquier elemento en un mensaje *SOAP* puede tener definidas sus propias reglas de codificación.

SOAP define sus propias reglas de codificación, identificadas por el `namespace` `http://www.w3.org/2001/12/soap-encoding`, pero no mandata el uso de las mismas, sino que las distintas implementaciones del protocolo *SOAP* pueden definir sus propias reglas de codificación.

La codificación definida por SOAP se basa en un sistema simple de tipos de datos que es la generalización de las características comunes de los sistemas de tipos de datos, tanto de los lenguajes de programación, como de las bases de datos y datos semiestructurados

2.2.6 SOAP RPC

Uno de los aspectos fundamentales de *SOAP* es definir una metodología para encapsular y realizar invocaciones *RPC* usando la extensibilidad y flexibilidad que proporciona *XML*.
En este caso el comportamiento es similar al de una arquitectura cliente servidor, donde el proceso es iniciado por el cliente y el servidor responde al requerimiento del mismo.
Este tipo de comunicación se basa en un sistema de mensajes SOAP sincrónicos, codificados en XML.

En la figura 9 se observa la arquitectura básica de las invocaciones RPC encapsuladas en mensajes SOAP:

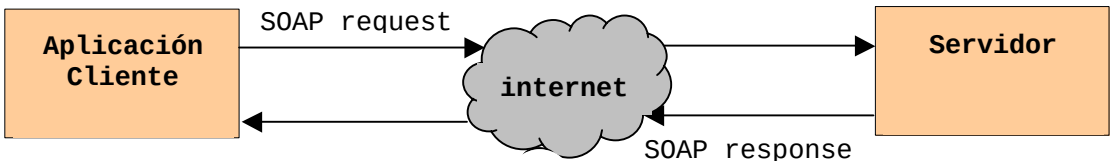


Figura 9 – Arquitectura básica de invocaciones RPC en SOAP

El uso de *SOAP* para realizar *RPC* es ortogonal al protocolo usado para el transporte del mensaje *SOAP*. En el caso de usar HTTP, la invocación *RPC* se mapea naturalmente al `request` de *HTTP* y la respuesta *RPC* se mapea al `response` de *HTTP*.

Para realizar una invocación *RPC* es necesaria la siguiente información:

- ✓ La *URI* del receptor del mensaje
- ✓ El nombre del procedimiento o método
- ✓ Los parámetros del procedimiento o método

Las invocaciones y respuestas *RPC* son transportadas en el elemento *BODY* del mensaje *SOAP*, de acuerdo a la siguiente convención:

- La invocación *RPC* se modela como un registro. Este registro constituye una entrada o bloque del elemento *BODY*.
- El nombre asociado a la `tag` de la entrada, que representa el registro, es el mismo que el nombre del procedimiento o método y contiene un subelemento para cada parámetro de entrada o entrada / salida.
- El nombre de cada subelemento (parámetro), ya sea de entrada o entrada / salida, es idéntico al nombre del parámetro *RPC*. Los parámetros aparecen en el mismo orden que la signatura del procedimiento o método.
- La respuesta *RPC* se modela como un registro. Este registro constituye una entrada o bloque del elemento *BODY*.
- El nombre asociado a la `tag` de la entrada, que representa el registro, es el mismo que el nombre del procedimiento o método, con el agregado al final de la cadena "`Response`", y contiene un subelemento para el valor de retorno y para cada parámetro de salida o entrada / salida.
- El primer subelemento corresponde al valor de retorno y los siguientes subelementos corresponden a los restantes parámetros. El nombre de cada subelemento (parámetro) es idéntico al nombre del parámetro *RPC*. Los parámetros aparecen en el mismo orden que la signatura del procedimiento o método.

En la figura 10 se muestra un ejemplo de una invocación *RPC* encapsulada en un mensaje *SOAP* y en la figura 11 la respuesta.

```
POST /StockQuote HTTP/1.1
Host: www.example.org
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://example.org/2001/06/quotes"

<SOAP-ENV:Envelope
  xmlns:env="http://www.w3.org/2001/12/soap-envelope" >
   < SOAP-ENV:Header>
       <t:Transaction
           xmlns:t="http://example.org/2001/06/tx"
           env:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
           env:mustUnderstand="1" >
               5
       </t:Transaction>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body >
       <m:GetLastTradePrice
             env:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
             xmlns:m="http://example.org/2001/06/quotes" >
           <m:symbol>DEF</m:symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope >
```

Figura 10 – Invocación SOAP RPC

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:env="http://www.w3.org/2001/12/soap-envelope" >
   <SOAP-ENV:Body>
     <m:GetLastTradePriceResponse
       env:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
       xmlns:m="http://example.org/2001/06/quotes" >
        <Price>34.5</Price>
     </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 11 – Respuesta SOAP RPC

2.2.7 Características de SOAP

1- estándar abierto
SOAP es una especificación abierta (open source), construido sobre tecnologías también abiertas como *XML* y *HTTP*. Por estas razones está siendo aceptado uniformemente por la industria de software, lo cual incrementa sus posibilidades de transformarse en un estándar [7].

2- Simplicidad y extensibilidad
La especificación de *SOAP* está diseñada teniendo como objetivos principales la simplicidad y la extensibilidad. Para lograr estos objetivos se omiten, en la especificación, características que comúnmente se encuentran en los sistemas distribuidos tales como: garbage collection, manejo de estado de objetos, seguridad, objetos por referencia, etc. [4] [7].

3- Independiente de plataformas y lenguajes
Al estar construido sobre *XML* y *HTTP*, *SOAP* es completamente independiente de plataformas, sistemas operativos o lenguajes de programación.

4- Interoperabilidad [8]
El mundo de la computación distribuida se encuentra dividido en grandes grupos, dependiendo de la tecnología usada, cada uno de los cuales adhiere a su propio protocolo para realizar invocaciones *RPC*. Como resultado de esta división, la interacción entre sistemas basados en diferentes tecnologías no es algo simple de lograr.
Frente a este hecho, surge la pregunta de porque, si los navegadores existentes pueden acceder a los servidores Web sin tener en cuenta la plataforma del mismo, los programadores no pueden hacer lo mismo, o sea invocar servicios remotos de manera independiente de la plataforma.

Uno de los objetivos de *SOAP* es eliminar las dificultades que separan las plataformas de la programación distribuida. Para esto SOAP se basa en propiedades de otros protocolos exitosos en internet: simplicidad, flexibilidad, independencia de plataforma y basado en texto (*XML*). Bajo este enfoque, *SOAP* no es una nueva tecnología sino la utilización de tecnologías existentes, en internet, para estandarizar la comunicación entre aplicaciones a través de internet.
En el nivel más bajo *SOAP* es una manera estándar de codificar la información necesaria para invocar servicios remotos en un formato que puede ser transportado a través de la red e interpretado por el servicio remoto independientemente de su plataforma.

Otras tecnologías de procesamiento distribuido como ser *DCOM* (**D**istributed **C**omponent **O**bject **M**odel) de Microsoft o *CORBA*[1] (**C**ommon **O**bject **R**equest **B**roker **P**rotocol), para mencionar solamente algunas de las más populares, utilizan formatos propietarios de codificación de la información.
DCOM usa un formato binario llamado *NDR* (**N**etwork **D**ata **R**epresentation) y *CORBA* utiliza un formato binario, similar pero incompatible, llamado *CDR* (**C**ommon **D**ata **R**epresentation).
Por este motivo no es posible hacer invocaciones entre un sistema *DCOM* y un sistema *CORBA* en forma nativa y solamente es posible si se dispone de un software específico (bridge), el cual agrega complejidad y disminuye la performance.

SOAP utiliza *XML*, como reemplazo de los formatos propietarios de codificación de otras tecnologías, lo cual le permite asumir una posición neutral entre las distintas plataformas.

Por lo tanto en el área de la interoperabilidad *SOAP* tiene ventajas con respecto a otros protocolos, como ser *IIOP* de *CORBA*, *DCOM* o *Java RMI*, fundamentalmente en el

[1] De aquí en adelante cuando se menciona *CORBA* se está haciendo referencia al *IIOP* (**I**nternet **I**nter-**O**rb **P**rotocol) , para comunicar objetos remotos.

ámbito de internet donde las aplicaciones existentes son sumamente heterogéneas (escritas en distintos lenguajes y funcionando en distintas plataformas).

5- Firewalls [9]

Generalmente los desarrolladores se deben esforzar mucho para hacer que sus aplicaciones distribuidas funcionen en internet por la presencia de firewalls dado que los mismos bloquean comúnmente casi todos los puertos excepto algunos, entre los cuales se encuentra el puerto 80 (*HTTP*).

Las tecnologías de procesamiento distribuido, como ser *DCOM* y *CORBA*, se basan en la asignación dinámica de puertos para realizar invocaciones remotas. Esto implica que los administradores del lugar donde reside el servidor, desbloqueen los puertos necesarios en el firewall, como forma de resolver el problema.
Sin embargo, las solución anterior no es completa, dado que si los clientes de las aplicaciones distribuidas están detrás de otro firewall corporativo, tienen el mismo problema, es decir que si no configuran su firewall para abrir el mismo puerto no pueden hacer uso de la aplicación. Obviamente esta reconfiguración del firewall, del cliente, para acomodarlo a la aplicación que se intenta usar no es algo práctico.

Como *SOAP* utiliza *HTTP* como mecanismo de transporte y la mayoría de los firewalls no bloquean el puerto *HTTP*, no existen problemas de invocaciones de ambos lados de los firewall.

Es de destacar que *SOAP* permite que los administradores configuren los firewalls para bloquear selectivamente los mensajes *SOAP* usando cabezales específicos. En particular el cabezal *SOAPACTION* (sección 2.2.4) puede ser usado por los firewalls para conocer el requerimiento que llega al mismo.

6- Bajo acoplamiento

SOAP [10] permite que los sistemas tengan un bajo acoplamiento por el hecho que no define la infraestructura de software a usar.
Cuando se implementa un componente, para ser usado mediante *SOAP*, el desarrollador del componente no indica a los clientes que tecnología deben utilizar para acceder al mismo. Esto contrasta con otras tecnologías distribuidas (*CORBA*, *DCOM*, etc.) donde existe un alto acoplamiento [11] con la arquitectura subyacente de las aplicaciones lo que implica que tanto el servidor (componente) como el cliente deben usar el mismo sistema o por lo menos sistemas compatibles.

7- Escalabilidad

Al usar el protocolo *HTTP* (stateless) como transporte, los sistemas distribuidos construidos sobre *SOAP* son sumamente escalables [7] [12] .

8- Performance

Dado que los mensajes *SOAP* están codificados en *XML* (texto), el mismo es, generalmente, más grande que el mensaje equivalente en *DCOM* o *CORBA*, que utilizan un formato binario. Por lo tanto *SOAP* no es una forma de comunicación compacta.

Además la performance es baja [7] [12] por el hecho de tener que extraer el *ENVELOPE* del paquete *HTTP* y analizar (parsing) el contenido *XML*, para posteriormente realizar la invocación al método requerido.

9- Manejo del estado

SOAP no define, por sí mismo, ningún mecanismo para el manejo de estado de los objetos. En todo caso si se usa *HTTP* como protocolo de transporte el manejo de estado es stateless [7] [12].

10- Garbage collector

SOAP no define ningún mecanismo para manejar objetos que no están siendo referenciados o para soportar `garbage collector` en forma remota [12]. La propia especificación de *SOAP* explícitamente deja de lado este aspecto, de forma de lograr un protocolo simple y liviano [4].

11- Seguridad

SOAP no define ningún mecanismo de seguridad. En particular dado que los mensajes *SOAP* están en formato texto, cualquiera podría leerlos o alterarlos [11].

De todas maneras la seguridad en *SOAP* queda determinada, actualmente, por el protocolo de transporte, por ejemplo *HTTPS* usando *SSL* (**S**ecure **S**ocket **L**ayer) [7], como forma de proteger el mensaje.

2.3 XML

XML (E**X**tensible **M**arkup **L**anguage) [13] es una especificación desarrollada por W3C basada en *SGML* (**S**tandard **G**eneralized **M**arkup **L**anguage) [14]. Los diseñadores de *XML* tomaron lo mejor de *SGML* y desarrollaron un lenguaje similar pero mucho más simple de usar [15].
El desarrollo de *XML* comenzó en 1996 y es, desde febrero de 1998, una recomendación oficial del W3C.

XML es un lenguaje de marcas, similar a *HTML* (**H**yper **T**ext **M**arkup **L**anguage), que define un conjunto de reglas que permiten estructurar datos en lo que se denomina un documento *XML*.
La similitud entre *XML* y *HTML* proviene del hecho que, al igual que *HTML*, *XML* utiliza `tags` y atributos. Sin embargo , mientras que *HTML* especifica lo que significa cada `tag` y atributo, *XML* utiliza las `tags` simplemente como delimitadores de datos y deja la interpretación de las mismas a la aplicación que utiliza el documento.
Dicho de otra manera, las `tags` y la estructura de un documento *HTML* son predefinidas mientras que en *XML*, el autor de un documento debe definir sus propias `tags` y la estructura del documento [16]. El hecho que las `tags` no estén predefinidas hace de *XML* un lenguaje extensible.

Los documentos *XML* son archivos de texto, lo cual tiene la ventaja, frente a los formatos binarios, que permite a las personas entender el documentos, si es necesario, sin la necesidad de una aplicación específica, utilizando herramientas tradicionales como ser un editor de textos. Al ser los documentos *XML* un archivo de texto el tamaño de los mismos es generalmente más grande que su correspondiente en formato binario. La utilización de texto fue una decisión explícita de los diseñadores de *XML* teniendo en cuenta las ventajas que proporciona este formato [15].

XML no requiere licencias (es libre), es independiente de plataformas y aplicaciones y no está ligado a un vendedor de `software` particular, lo cual implica que se dispone de un soporte amplio y creciente para el mismo [15].
Al ser independiente de plataformas y aplicaciones es sumamente útil para el intercambio de información en una gran variedad de áreas, fundamentalmente cuando intervienen sistemas heterogéneos (interoperabilidad).

XML permite definir un nuevo documento combinando y reusando otros documentos existentes [15]. Al combinar documentos independientes pueden surgir colisiones de nombres, en el caso que en los documentos a combinar existan elementos o atributos con el mismo nombre, generándose dudas sobre el significado de un elemento o atributo. Para eliminar estas colisiones *XML* utiliza el mecanismo de `namespace`.
Un `namespace` [17] es una manera de crear nombres. Consiste en una colección de nombres identificados por una *URI* (**U**niform **R**esource **I**dentifier) [6]. Al usar `namespaces` en los documentos XML es posible identificar los elementos o atributos de manera única y evitar las colisiones de nombres.

Cada `tag` de un documento *XML* define un elemento. Un elemento puede contener otros elementos (denominado `element content`) o contener otros elementos y texto (`mixed content`). Generándose de esta manera una anidación jerárquica que es característica de los documentos *XML*.
Eventualmente un elemento puede contener solo texto (`simple content` ó `text content`) o no tener contenido (`empty content`). Un elemento puede ser declarado con contenido *ANY* para indicar que la estructura y contenido del elemento es arbitraria.
Los elementos pueden tener atributos, que son parejas formadas por un nombre y un valor. Los atributos aparecen en la `tag` de comienzo del elemento.

Las reglas que definen lo que constituye un documento *XML* [13] son estrictas y simples, exigiendo que un documento *XML* sea bien formado y que sea válido.
Un documento *XML* es bien formado si su sintaxis es conforme a la definida en la especificación de *XML*. Básicamente la sintaxis de un documento *XML* está definida por las siguientes condiciones:

- Un documento *XML* debe tener un único elemento, denominado elemento documento, que contiene a los demás elementos.
- Un documento *XML* debe comenzar con una declaración que especifica la versión de *XML* en uso
- Cada elemento del documento tiene una `tag` de comienzo.
- Cada elemento del documento tiene una `tag` de finalización.
- Los elementos deben estar correctamente anidados, no pudiendo existir solapamientos.
- Un atributo no puede tener múltiples valores. El valor de cada atributo está delimitado por comillas (simples o dobles).
- Los nombres de atributos y elementos deben seguir las convenciones establecidas en la especificación.

Un documento *XML* es válido si es conforme con un *XML Schema* (sección 2.4) el cual describe su estructura.

2.4 XML Schema

XML Schema [18] es una recomendación oficial del W3C. Es un mecanismo para definir un modelo, para una clase de documentos, el cual describe la estructura, contenido y semántica de los documentos de la clase [19].
Desde otro punto de vista un *XML Schema* es un acuerdo sobre un vocabulario común cuando se realiza un intercambio de documentos [20].

En los *XML Schemas* los modelos son descriptos en términos de restricciones [20] las que definen que puede suceder en un determinado contexto. Básicamente hay dos tipos de restricciones que se pueden imponer en el modelo: la restricción de contenido, que describe el orden y anidamiento de los elementos del documento *XML*, y la restricción de tipos, que describe los tipos de datos válidos para la información contenida en el documento *XML*.

Otra forma para determinar la validez de un documento *XML* es el uso de *DTD* (**D**ocument **T**ype **D**efinition) [21]. *DTD* es un mecanismo propio de *SGML* que fue heredado por *XML*.
Un *DTD* puede ser usado para definir el modelo de contenido y, de manera limitada, para definir los tipos de datos de los atributos. Los *DTD* tienen varias limitaciones:

- No están escritos en *XML*.
- No soportan `namespace`.
- El soporte para tipos de datos es sumamente limitado.
- El mecanismo de extensión que utilizan es complejo e insuficiente.

Los *XML Schemas* tienen una expresividad superior a los *DTD* y carecen de sus limitaciones. En este sentido los *XML Schemas* pueden ser vistos como los sucesores naturales de los *DTD*. Entre sus características más relevantes se encuentran las siguientes:

- Proveen un sistema de tipos de datos predefinido amplio y más expresivo que el sistema de tipos de los *DTD*. El sistema de tipos permite la definición de nuevos tipos de datos a partir de tipos de datos predefinidos o ya definidos.
- Proveen soporte para `namespace`
- Están escritos en *XML*, por lo cual no es necesario conocer otro lenguaje como sucede en el caso de los *DTD*. Además pueden usarse las mismas herramientas, tanto para edición y procesamiento, que se utilizan para procesar los documentos *XML*.
- Brindan mecanismos de extensibilidad, de manera análoga a *XML*, por estar escritos en *XML*.

El propósito de un *XML Schema*, como fue mencionado, es permitir la validación de un documento *XML*. Cualquier documento *XML* que está escrito conforme a las restricciones del modelo definido en un *XML Schema* es, por definición, válido de acuerdo a dicho *XML Schema*. Dicho de otra manera un documento *XML* es válido si pertenece a la clase de documentos descriptos por el *XML Schema*.

La posibilidad de testear la validez de documentos *XML*, utilizando programas de `software`, es un aspecto sumamente importante en las aplicaciones en el ámbito de `internet`, que reciben y envían información desde y hacia varias fuentes, de manera de reducir los errores que se pueden producir. Por ejemplo:

- En el contexto del comercio electrónico, cuando se recibe una orden de compra es deseable saber, antes de su procesamiento, que en la misma no falta nada, que no hay información que no se esperaba y que la información incluida es del tipo correcto (las cantidades son números positivos, los precios son valores decimales, etc.

- En el contexto de intercambio de información entre bases de datos corporativas de empresas, que no son idénticas, cuando se recibe un registro de una de las bases de datos en formato *XML*, es necesario saber si el mismo es válido previamente a su inserción en la base de datos.
  Los datos erróneos deben ser rechazados para que no exista la posibilidad de agregar datos incorrectos a la base de datos.

Al exigir que un documento *XML* sea bien formado (sección 2.3) y que sea válido, no se eliminan todos los errores que puedan existir en el documento, pero se reducen de manera significativa.
Utilizando ambos conceptos, cualquier documento puede clasificarse en una de las siguientes categorías:

- Si no está bien formado no es *XML*.
- Si un documento no identifica el *XML Schema*, al cual dice conformar, es simplemente bien formado.
- Si un documento tiene asociado un *XML Schema*, pero no es conforme a las restricciones descriptas en el mismo, es bien formado pero no es válido.
- Si un documento tiene asociado un *XML Schema* y el documento es conforme al mismo, el documento es bien formado y válido.

2.5 Lenguajes de Consulta de XML

Muchos lenguajes de consulta para documentos XML han sido propuestos, entre los que se destacan *XSL* [22], *XQL* [23], *XML-QL* [24], *Lorel* [25], *Quilt* [26], *XPath* [27] y *XQuery* [28]. Las características que se mencionan a continuación, para cada uno de los lenguajes, son extraídas de [29] y [30].

XSL (E**x**tensible **S**tylesheet **L**anguage) [22] es un lenguaje propuesto por el W3C cuya finalidad principal es transformar documentos *XML*, pero tiene características que pueden servir de base para realizar consultas, en particular utiliza el lenguaje de expresiones definido por *XPath* [27] para acceder a partes del documento *XML*.

XQL [23] es un lenguaje propuesto por Microsoft como una extensión del lenguaje *XSL* [22]. Es un lenguaje diseñado con el objetivo de proveer una sintaxis simple y compacta para realizar consultas.

XML-QL [24] es un lenguaje propuesto por AT&T Labs que extiende el lenguaje *SQL* con cláusulas que permiten construir documentos a partir de los resultados de una consulta. Permite expresar consultas y también transformaciones.

Lorel [25] es un lenguaje propuesto por la Universidad de Stanford. Originalmente fue diseñado para consultar datos semiestructurados y posteriormente fue extendido para soportar datos *XML*. Es un lenguaje amigable con un estilo similar al de *SQL* y *OQL*.

Quilt [26] es un lenguaje propuesto por el centro de investigación Almaden de IBM. Es un lenguaje de consulta que integra características de varios lenguajes, incluyendo *XQL* [23], *XPath* [27], *XML-QL* [24], *Lorel* [25], *SQL* y *OQL*.

XPath [27] es un lenguaje propuesto por el W3C que permite seleccionar partes de un documento *XML*.

XQuery [28] es un lenguaje propuesto por el W3C que surge a partir de *Quilt* [26] y que actualmente se encuentra en la etapa de especificación.

A pesar que varios de los lenguajes mencionados tienen características que los hacen sumamente poderosos como lenguajes de consulta, el único que hasta el momento es una recomendación oficial del W3C es el lenguaje *XPath*, el cual se detalla en la siguiente sección.

2.5.1 XML Path Language (XPath)

XPath es un lenguaje cuyo propósito principal es seleccionar partes de un documento *XML* [27]. El nombre *XPath* se debe a la utilización de expresiones que involucran caminos, las cuales permiten la navegación por el documento *XML*.
La sintaxis, definida por *XPath*, es compacta y no está basada en *XML*. En particular es similar a la sintaxis usada para navegar por los sistemas de archivos tradicionales.

XPath opera sobre la estructura lógica de los documentos *XML* y no directamente sobre el documento *XML* en sí. Para esto *XPath* define un modelo de datos que representa el documento *XML* como un árbol de nodos, que refleja la anidación jerárquica del documento *XML*. El modelo define los distintos tipos de nodos que puede contener el árbol. Existen siete tipos de nodos (`root`, elemento, texto, atributo, `namespace`, `processing instruction` y comentario), de los cuales los más relevantes son:

- Nodo `root` (`/`): es un nodo virtual y es la raíz del árbol. Es el padre del nodo correspondiente al elemento documento del documento *XML*.
- Nodos elemento: cada elemento del documento *XML* tiene un nodo elemento correspondiente en el árbol. Los nodos elemento pueden tener hijos que corresponden a elementos o texto del documento *XML*.
- Nodos atributo: cada nodo elemento puede tener asociado un conjunto de nodos atributos, que se corresponden con los atributos del elemento en el documento *XML*. Los nodos elementos no comparten nodos atributos entre sí, es decir que para dos nodos elementos distintos, ninguno de los nodos atributos de uno de los elementos es el mismo nodo que los correspondientes nodos atributos del otro elemento.
  Los nodos atributo no tienen hijos. Por definición, los nodos atributos de un nodo elemento no son considerados hijos del nodo elemento, aunque el nodo elemento es el padre.
- Nodos texto: corresponden al texto asociado a los elementos en el documento *XML*. Los nodos texto no tienen hijos.
- Nodos comentario: corresponden a los comentarios del documento *XML*.

Por ejemplo, dado el documento *XML*:

```
<?xml version='1.0' encoding='UTF-8' ?>
<personas>
   <persona fnac = “1912” >
    <datos>
      <nombre>Alan</nombre>
      <apellido>Turing</apellido>
    </datos>
    <profesion>matematico</profesion>
    <profesion>criptografo></profesion>
   </persona>
   <persona fnac = “1918” >
    <datos>
      <nombre>Richard</nombre>
      <apellido>Feynman</apellido>
    </datos>
    <profesion>fisico</profesion>
   </persona>
</personas>
```

Figura 12 – Ejemplo de documento XML

el árbol de nodos que representa el documento anterior es el que se muestra en la figura 13.

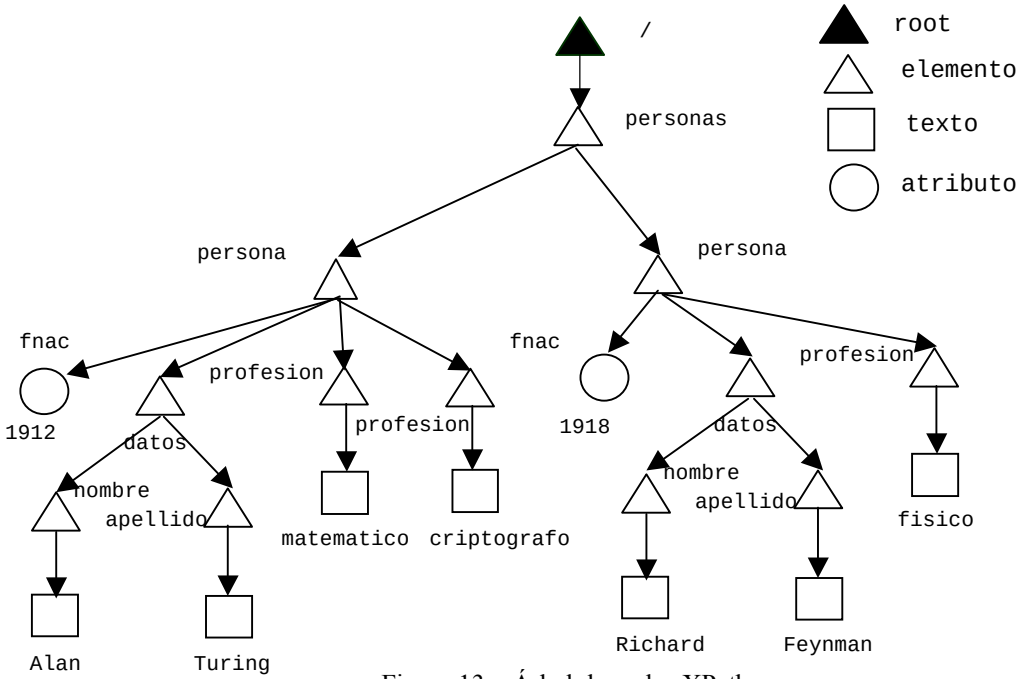


Figura 13 – Árbol de nodos XPath

En el árbol existe un orden definido sobre todos los nodos del mismo. El nodo `root` es, por definición, el primer nodo. Los nodos elemento se encuentran ordenados según el orden de ocurrencia en el documento *XML* y aparecen en el árbol antes que sus hijos. Los nodos atributo de un elemento, aparecen antes que los hijos del elemento.
Los nodos no comparten hijos, es decir que para dos nodos distintos, ninguno de los nodos hijo, de uno de los nodos, es el mismo nodo que los correspondientes nodos hijos del otro nodo.
Cada nodo tiene un único padre, excepto el nodo `root` que no tiene padre. Los descendientes de un nodo son los hijos del nodo y los descendientes de los hijos del nodo.

Como se mencionó anteriormente, el propósito de *XPath* es seleccionar partes de un documento *XML*, lo cual se realiza usando caminos de localización (caminos).
Los caminos permiten navegar por el árbol, que representa el documento *XML*, y seleccionar conjuntos de nodos relativos a un nodo contexto.
El concepto de nodo contexto es fundamental para comprender como se seleccionan los conjuntos de nodos. Un camino no se evalúa automáticamente sobre la totalidad del documento, sino a partir de un punto de partida que es el nodo contexto. El nodo contexto puede ser cualquier nodo del árbol, inclusive el nodo `root` en cuyo caso la consulta involucra la totalidad del documento.

Hay dos tipos de caminos: caminos relativos y caminos absolutos.
Un camino relativo consiste en una secuencia de uno o más pasos de localización (pasos) separados por /:

$$paso_1/.../paso_m$$

Los pasos en un camino relativo se componen de izquierda a derecha. Cada paso selecciona un conjunto de nodos relativos a un nodo contexto. Una secuencia inicial de pasos se compone con el siguiente paso de localización de la siguiente forma:

> la secuencia inicial de pasos selecciona un conjunto de nodos relativos a un nodo contexto. Cada nodo de este conjunto se usa como nodo contexto para el siguiente paso. Los distintos conjuntos de nodos seleccionados por el paso se unen. El conjunto de nodos seleccionado por la composición de pasos es dicha unión.

Por ejemplo:

```
child::A/child::B
```

> selecciona los elementos `B`, hijos de los elementos `A` hijos del nodo contexto, o, en otras palabras, los elementos `B` nietos del nodo contexto, que tengan como padres elementos `A`.

Un camino absoluto consiste en / seguido, opcionalmente, por un camino relativo:

$$/paso_1/.../paso_m$$

La / inicial, selecciona el nodo `root`. Si es seguida por un camino relativo, el camino absoluto selecciona el conjunto de nodos que sería seleccionado por el camino relativo teniendo como nodo contexto al nodo `root`.

Cada paso ($paso_i$) consiste de tres partes: un eje (`axis`), un `node test` y una lista de predicados (que eventualmente puede ser vacía). La sintaxis de un paso es la siguiente:

```
axis::node test [predicado]*
```

El eje (`axis`) especifica un conjunto de nodos, relativo al nodo contexto, candidatos a estar en el conjunto de nodos resultado del paso. El conjunto de nodos candidatos se especifica en términos de relaciones en el árbol que representa el documento *XML*. *XPath* define varios ejes, entre los que se encuentran:

- `child`: selecciona todos los nodos hijos del nodo contexto.
- `parent`: selecciona el nodo padre del nodo contexto.
- `self`: selecciona el nodo contexto
- `descendant`: selecciona los nodos descendientes del nodo contexto
- `attribute`: selecciona los atributos del nodo contexto

El `node test` permite seleccionar los nodos que son de un determinado tipo (por ejemplo nodos de tipo texto, nodos de tipo comentario, etc) o bien nodos que tienen un determinado nombre (por ejemplo persona, profesión, etc.). El nombre de un nodo es el mismo que el nombre del elemento o atributo del documento XML al cual el nodo corresponde.

Un paso puede tener cero o más predicados. Los predicados son expresiones[2] booleanas que permiten filtrar un conjunto de nodos.
Para que un nodo forme parte del conjunto de nodos resultado de un paso, debe satisfacer cada uno de los predicados del mismo.
El conjunto de nodos inicial del paso (formado por los nodos determinados por el eje y que satisfacen el `node test`) se filtra por el primer predicado para generar un nuevo conjunto de nodos; este nuevo conjunto de nodos es filtrado usando el segundo predicado, y así sucesivamente hasta el último predicado.

Para determinar si un nodo satisface un predicado, se evalúa la expresión del predicado considerando el nodo como nodo contexto. Si la expresión evalúa[3] a verdadero para el nodo contexto, el mismo es incluido en un nuevo conjunto de nodos; si la expresión evalúa a falso el nodo contexto es descartado.

Además de la sintaxis mostrada anteriormente para los pasos, *XPath* permite el uso de una notación abreviada con el objetivo de que las consultas sean más compactas. Las abreviaturas son referidas a algunos de los ejes, por ejemplo `attribute` puede ser abreviado por @, `parent` puede ser abreviado por .., `self` puede ser abreviado por ., etc.
La abreviatura más importante es la referida al eje `child`, el cual puede ser omitido porque es considerado el eje por defecto, de esta manera la consulta:

```
child::A/child::B
```

puede escribirse en notación abreviada como:

```
A/B
```

Los siguientes son algunos ejemplos de consultas *XPath* sobre el documento *XML* mostrado en la figura 12:

1. `/personas/persona/profesion` Devuelve las profesiones de las personas.

```
<profesion>matematico</profesion>
<profesion>criptografo></profesion>
<profesion>fisico</profesion>
```

2. `/personas/persona/datos/nombre` Devuelve los nombres de las personas.

```
<nombre>Alan</nombre>
<nombre>Richard</nombre>
```

[2] Las expresiones que forman los predicados pueden incluir caminos.
[3] La expresión de un predicado es evaluada y el resultado de la misma es convertido a un valor de tipo `boolean`, según determinadas reglas definidas en la especificación de *XPath*.

3. `/personas/persona[profesion = “fisico”]` Devuelve las personas que tengan como profesión físico.

```
<persona fnac = “1918”>
 <datos>
  <nombre>Richard</nombre>
  <apellido>Feynman</apellido>
 </datos>
 <profesion>fisico</profesion>
</persona>
```

4. `/personas/persona[@fnac < 1915]` Devuelve las personas que hayan nacido antes de 1915.

```
<persona fnac = “1912” >
 <datos>
  <nombre>Alan</nombre>
  <apellido>Turing</apellido>
 </datos>
 <profesion>matematico</profesion>
 <profesion>criptografo></profesion>
</persona>
```

5. `/personas/persona[@fnac < 1920]/datos[nombre = “Alan”]` Devuelve las personas que hayan nacido antes de 1920 y cuyo nombre sea Alan.

```
<persona fnac = “1912” >
 <datos>
  <nombre>Alan</nombre>
  <apellido>Turing</apellido>
 </datos>
 <profesion>matematico</profesion>
 <profesion>criptografo></profesion>
</persona>
```

6. `/personas/persona[@fnac = 1912][profesion=”criptografo”]` Devuelve las personas que hayan nacido en 1912 y tengan como profesión criptografo.

```
<persona fnac = “1912” >
 <datos>
  <nombre>Alan</nombre>
  <apellido>Turing</apellido>
 </datos>
 <profesion>matematico</profesion>
 <profesion>criptografo></profesion>
</persona>
```

2.6 XML for Analysis

XML for Analysis [31] es la especificación de una *API* (**A**pplication **P**rogram **I**nterface) que soporta el intercambio de datos multidimensionales entre una aplicación cliente y un servidor, funcionando sobre cualquier plataforma y cualquier lenguaje.

Cuando se usa *OLE DB*, para acceder a una fuente de datos, es necesario instalar en la máquina cliente componentes de `software` que permitan, a las aplicaciones cliente, acceder a los datos de un proveedor de datos particular.
Esta situación genera arquitecturas altamente acopladas que crean dependencias relativas a plataformas de `hardware`, sistemas operativos, lenguajes de programación y versiones de las componente instaladas en el cliente y el servidor.
XML for Analysis extiende los conceptos de *OLE DB*, proporcionando un acceso estándar y universal a fuentes de datos a través de `internet`, sin la necesidad de tener que instalar en la máquina cliente componentes de `software` altamente acoplados con el proveedor de datos que se intenta acceder.

La especificación de *XML for Analysis* tiene los siguientes objetivos de diseño:

- Proveer una *API* para el acceso a proveedores de datos multidimensionales remotos, que pueda ser usada universalmente en `internet` y también en `intranets`.
- Optimizar una arquitectura `stateless`, que no requiera componentes en el cliente y que minimice el envío, de ida y vuelta, de información entre el servidor y la aplicación cliente.
- Soportar implementaciones tecnológicamente independientes usando cualquier herramienta, lenguaje de programación y plataforma de `hardware`.
- Utilizar estándares abiertos de `internet` como ser *SOAP*, *XML* y *HTTP*.
- Reutilizar conceptos de diseño de *OLE D*B, de forma que las aplicaciones *OLE DB* para *OLAP* (**O**n **L**ine **A**nalytical **P**rocessing) y los proveedores *OLE DB* puedan ser fácilmente adaptados a la especificación de *XML for Analysis*.

XML for Analysis está basada en *XML* y define dos métodos: `Discover` y `Execute`. Ambos métodos reciben y envían datos en formato *XML*, a través de sus parámetros, lo cual permite una arquitectura con bajo acoplamiento entre el cliente y el servidor,

La comunicación entre *XML for Analysis* y las aplicaciones cliente se realiza usando el protocolo *SOAP*. Las aplicaciones cliente invocan los métodos `Discover` y `Execute` usando las convenciones definidas en el protocolo *SOAP* (sección 2.2.6). Por su parte, *XML for Analysis*, al recibir la invocación, obtiene los datos requeridos y los envía a la aplicación cliente usando *SOAP*.

El método `Discover` es usado por la aplicación cliente para obtener información. Esta información puede incluir una lista de fuentes de datos disponibles y datos sobre el proveedor para una fuente de datos particular.
Teniendo en cuenta que la aplicación cliente puede llegar a necesitar distintas clases de información, el método `Discover` ofrece una interfase genérica y extensible que permite especificar cual es la información requerida por la aplicación cliente, mediante los valores de los parámetros de entrada con que se invoca al método.
Los parámetros de entrada permiten a la aplicación cliente:

- determinar que tipo de información es retornada por el método a la aplicación cliente (por ejemplo lista de fuentes de datos disponibles, lista de palabras reservadas del proveedor de datos, etc.).

- definir restricciones, que habilitan a la aplicación cliente a restringir la información retornada en el parámetro de salida, filtrándola mediante las restricciones con que se invoca al método. Cada proveedor que implemente la *API* puede agregar nuevas restricciones a las predefinidas. El método contempla la posibilidad de que la aplicación

cliente obtenga la lista de todas las restricciones disponibles, invocando al propio método e indicando que el tipo de la información a retornar es la lista de restricciones.

- definir propiedades, que habilitan a la aplicación cliente a controlar algún aspecto del método, como ser el formato del resultado, información sobre la conexión, tiempo máximo de espera para establecer la conexión, usuario, `password`, etc. Cada proveedor que implemente la *API* puede agregar nuevas propiedades a las predefinidas. El método contempla la posibilidad de que la aplicación cliente obtenga la lista de todas las propiedades disponibles, invocando al propio método e indicando que el tipo de la información a retornar es la lista de propiedades.

El contenido y estructura de la información retornada por el método, a la aplicación cliente, es determinado teniendo en cuenta el tipo de información requerido y los valores de las restricciones.

El método `Execute` es usado para ejecutar expresiones multidimensionales sobre una fuente de datos. Los parámetros de entrada permiten a la aplicación cliente:

- especificar la sentencia multidimensional a ser ejecutada

- definir propiedades, que permiten controlar algún aspecto del método, de manera análoga al método `Discover`.

El contenido de la información retornada por el método, a la aplicación cliente, es el resultado de la ejecución de la sentencia multidimensional y la estructura es determinada por el valor de la propiedad `format`, con que fue invocado el método*. XML for Analysis* define la estructura de los dos posibles formatos: tabular y multidimensional.

Las expresiones multidimensionales que pueden ejecutarse corresponden, actualmente, al lenguaje *MDX* definido en la especificación de *OLE DB* para *OLAP*. En futuras versiones está previsto el uso del lenguaje *mdXML*, el cual es una versión de *MDX* encapsulada en *XML*.

XML for Analysis es `stateless` por defecto (o sea que el servidor no mantiene la identidad o contexto de un cliente después de finalizada la invocación a un método), pero brinda soporte para manejo de estado `stateful` (o sea que el servidor preserva la identidad y el contexto del cliente entre las invocaciones a métodos) a nivel de sesión.
XML for Analysis define una manera para soportar sesiones en el ámbito de `internet` usando, entre otros, la extensibilidad del protocolo *SOAP*. En particular se utilizan *SOAP HEADERS* para iniciar, mantener y cerrar una sesión.

2.7 Otros trabajos relacionados

En general los principales sistemas comerciales de bases de datos incorporan, actualmente, distintas técnicas relacionadas con *XML* en sus productos. Estas técnicas son en todos los casos propietarias de cada manejador.

DB2 XML Extender [32] incorpora extensiones para almacenar, componer y realizar consultas de documentos *XML* en la base de datos relacional y generar documentos *XML* a partir de datos relacionales mediante consultas *SQL* o *stored procedures* predefinidos.

Las extensiones de Oracle [33] permiten el almacenamiento de documentos *XML* en la base de datos relacional y la realización de consultas sobre los documentos *XML* almacenados, para lo cual dispone de distintas opciones como ser el uso de un subconjunto muy reducido de *XPath*.
Además provee el utilitario *XSU* (**X**ML **S**QL **U**tility), el cual consiste en un conjunto de clases *Java* mediante que brindan funcionalidades para la transformación de datos relacionales en documentos *XML* a partir de consultas *SQL* y para la utilización de documentos *XML* para actualizar, eliminar e insertar datos en la base de datos, basándose en un mapeo entre el documento *XML* y la base de datos que debe ser construido manualmente.

Microsoft SQL Server 2000 [34], a su vez, dispone de varias extensiones para soportar *XML*. Proporciona la extensión de la sentencia `select` de *SQL* para que el resultado de la consulta *SQL* sea devuelto en formato *XML*.
Otra extensión tiene con ver con el almacenamiento de documentos *XML* a través de *OpenXML*.
Por último brinda la posibilidad de generar vistas *XML* de la base de datos [35], que se describen mediante documentos *XML* denominados esquemas *XDR* (**X**ML **D**ata **R**educed) a los cuales se agregan anotaciones para establecer el mapeo con los elementos de la base de datos. Estos esquemas con anotaciones son usados para la realización de consultas, para lo cual se dispone de un subconjunto del lenguaje *XPath*. Para la realización de consultas pueden usarse las extensiones provistas en *OLEDB* y *ADO* que actualmente soportan *XML* y *XPath* [36].

Por otro lado existen proyectos que permiten la publicación de datos relacionales como documentos *XML*, entre los que se destacan Silkroute [37] y XPERANTO [38] [39].

SilkRoute [37] es un sistema desarrollado en un proyecto conjunto de AT&T y la Universidad de Pennsylvania, que actúa como una capa intermedia entre las aplicaciones y los sistemas manejadores de bases de datos. SilkRoute utiliza un lenguaje declarativo, llamado *RXL* (**R**elational to **X**ML transformation **L**anguage) para definir vistas *XML* de la base de datos. Las vistas generadas son consultadas, posteriormente, utilizando el lenguaje de consulta *XML-QL* [24].
SilkRoute realiza la composición de la consultas expresada en *XML-QL* con la consulta en *RXL* que genera la vista, generándose una nueva consulta *RXL*. Posteriormente se traslada la consulta *RXL* resultante a *SQL*, la cual es resuelta por la base de datos. El algoritmo que traslada la consulta *RXL* a *SQL* se explica muy brevemente y no se dan mayores detalles sobre el mismo.

XPERANTO [38] [39] es un proyecto de IBM Almaden Research Center que propone una capa intermedia entre las aplicaciones y los sistemas manejadores de bases de datos relacionales. Esta capa crea inicialmente una vista *XML* de la base de datos relacional y mediante el lenguaje *XQuery* [28] permite que los usuarios definan nuevas vistas a partir de la vista inicial o bien, realicen consultas sobre las vistas generadas. Posteriormente la consulta realizada es trasladada a *SQL* a los efectos de consultar la base de datos relacional. El algoritmo que traslada la consulta *XQuery* a *SQL* se explica únicamente mediante ejemplos, no brindándose mayores detalles.

Otros trabajos utilizan bases de datos relacionales para almacenar documentos *XML*. El enfoque principal de estos trabajos es a definir un esquema relacional, que permita almacenar eficientemente los documentos *XML*.

Si bien existen muchos trabajos al respecto, muy pocos especifican algoritmos que convierten consultas, expresadas en algún lenguaje de consulta *XML*, a sentencias *SQL* sobre el esquema definido.
Entre estos trabajos se destaca XRel [40] que se basa en el modelo de datos del lenguaje *XPath* (sección 2.5.1) para definir el esquema para almacenar los documentos *XML*. El esquema relacional consta de cuatro tablas:

- tablas para almacenar cada uno de los siguientes tipos de nodos: elemento, atributo y texto
- una tabla para almacenar los caminos a cada elemento o atributo *XML* del documento.

XRel especifica un algoritmo para convertir consultas, expresadas en un subconjunto del lenguaje *XPath*, a sentencias *SQL*. Sin embargo el algoritmo especificado no es un algoritmo general sino que está fuertemente influenciado por el esquema relacional, particular, definido para almacenar el documento *XML*.

2.8 Conclusiones

En este capítulo se presentaron las tecnologías dentro de las cuales se enmarca la tesis, así como los trabajos relacionados con el mismo.

Como conclusión puede decirse que si bien existen algunos mecanismos para acceder a bases de datos relacionales utilizando tecnologías relacionadas con *XML*, estos mecanismos no están estandarizados.
En general los mecanismos existentes representan soluciones particulares de algunos manejadores de bases de datos comerciales que no interactúan entre sí, o sea son mecanismos propietarios, o bien son prototipos propuestos por laboratorios de investigación o universidades, que no tienen una difusión a nivel comercial.

En cuanto al acceso a bases de datos relacionales a través de `internet`, tampoco existen mecanismos estandarizados.
Existe la propuesta de un mecanismo para realizar consultas multidimensionales, como es el caso de la *API XML for Analysis*. Sin embargo este mecanismo implica conocer la estructura lógica de la bases de datos multidimensional, por lo cual puede ser considerado simplemente como un mecanismo de transporte de consultas.

En consecuencia en esta tesis se propone integrar varias tecnologías `open source` consolidadas como recomendaciones oficiales del W3C (*XML*, *XML Schemas* y *XPath*), que pueden ser consideradas estándares, junto con tecnologías `open source` sumamente recientes o emergentes (*SOAP*), que aunque actualmente no es un estándar se estima que lo será en un plazo breve, de manera de definir un `framework` que permita la realización de consultas a bases de datos relacionales a través de `internet` haciendo transparente el diseño lógico de las mismas.

Capítulo III Arquitectura del Framework

3.1 Introducción

Durante las décadas pasadas, los sistemas de bases de datos han surgido como las herramientas tradicionales para el manejo de los datos. Luego de muchos años de desarrollo, la tecnología de bases de datos relacionales, así como los sistemas manejadores de bases de datos (*DBMS*), se encuentra en un estado sumamente maduro y estable. Los productos comerciales de bases de datos relacionales proporcionan almacenamiento, consulta y actualización de los datos, así como técnicas que permiten resolver distintos problemas relacionados con el almacenamientos de los datos y su acceso, como ser el procesamiento transaccional y control de concurrencia, entre otros.
En consecuencia las empresas han ido adoptando en forma creciente el uso de bases de datos y sus correspondientes manejadores, a los efectos de disponer de repositorios de almacenamiento para sus datos.

Generalmente los datos almacenados en las bases de datos, son accedidos, ya sea para ser actualizados o para ser consultados, por aplicaciones cliente existentes dentro del ámbito de la empresa, a través de una `intranet`.

Debido al auge de `internet` en los últimos tiempos, actualmente es muy frecuente que las empresas deseen exponer subconjuntos de los datos que disponen, para que sean consultados por aplicaciones cliente que se ejecutan fuera del ámbito de la empresa.
Para lograrlo se necesita una tecnología apropiada, que permita que las bases de datos relacionales de las empresas sean accedidas, para consultas, por aplicaciones cliente a través de `internet`.

A pesar de la aparente similitud que existe con el acceso a bases de datos relacionales por aplicaciones cliente que acceden, a los datos, a través de una `intranet`, existen varios aspectos que deben ser examinados cuando se traslada el escenario de una `intranet` a `internet`:

1. La tecnología de base a utilizar debe permitir la comunicación de las aplicaciones cliente a través de `internet` con el servidor que brinda el servicio de acceso a las bases de datos relacionales. En particular es necesario analizar soluciones a posibles problemas que se presenten en el ámbito de `internet`, entre los que se pueden mencionar:

    - Interoperabilidad: dado que las aplicaciones cliente pueden ser escritas en lenguajes de programación y ejecutarse en plataformas diferentes que los lenguajes y plataformas en que se ejecutan las aplicaciones cliente de la empresa que acceden a la base de datos.

    - Acoplamiento: esto punto es, en gran medida, consecuencia de lo anterior. No debe ser necesario tener instalados en la máquina cliente componentes de `software` que generen una arquitectura altamente acoplada con el servidor de base de datos, dado que esta situación genera dependencias que afectan la interoperabilidad.

    - Seguridad: es necesario reducir al mínimo posible los problemas que eventualmente pueden surgir a causa de los mecanismos de seguridad existentes en las empresas, como es el caso de los `firewalls`, cuando se intenta acceder a través de `internet`.

2. Acceso a las bases de datos por parte de las aplicaciones cliente. En particular es necesario analizar soluciones a los siguientes problemas:

- A partir de que información las aplicaciones cliente realizan las consultas. Esto significa determinar cual es el conocimiento del cual deben disponer las aplicaciones cliente a los efectos de poder realizar las consultas sobre la base de datos.

- Cual es el formato a utilizar para el intercambio de información.

En esta tesis se propone un `framework` para que aplicaciones cliente remotas puedan acceder a datos almacenados en bases de datos relacionales a través de `internet`, que brinda soluciones concretas a los aspectos mencionados y que tiene las siguientes características:

- Construido sobre estándares abiertos de `internet`:

  - *SOAP* como protocolo de comunicación.
  - *XML* como formato de intercambio de datos.

  de forma que sea tecnológicamente independiente de lenguajes de programación o plataformas y que exista un bajo acoplamiento con el servidor de base de datos.

- Brindar la posibilidad de abstraer a las aplicaciones cliente del diseño lógico de la base de datos (transparencia del diseño lógico), de manera que el conocimiento de como se relacionan las tablas, cuales son las claves primarias, etc. no sea determinante para la realización de consultas

- Brindar flexibilidad para la realización de consultas, utilizando un lenguaje de consulta.

3.2 Comunicación a través de Internet

Las aplicaciones cliente que consultan datos de las bases de datos, accediendo a los mismos a través de una `intranet` utilizan, en el caso de plataformas *Windows*, habitualmente tecnologías de acceso a datos basadas en *ODBC* (**O**pen **D**ata **B**ase **C**onnectivity) o alguna de sus variantes (por ejemplo *OLE DB*).

Si se traslada el uso de dichas tecnologías al ámbito de `internet` se presentan problemas críticos [31] porque dichas tecnologías requieren que en la máquina cliente, donde se ejecutan las aplicaciones, se encuentren instalados componentes de `software` altamente acoplados a los componentes de `software` instalados en los manejadores de base de datos, de forma que las aplicaciones sea capaces de acceder los datos almacenados en dichas bases de datos.

Este alto acoplamiento de las componentes de `software` genera arquitecturas donde existen dependencias entre las plataformas, sistemas operativos, lenguajes de programación y aún entre las versiones de las propias componentes, tanto las instaladas en la máquina cliente como las instaladas en la máquina donde se encuentra el manejador de la base de datos.

Las arquitecturas altamente acopladas no son apropiadas para funcionar en el entorno de `internet` donde es necesario un bajo acoplamiento que permita la independencia de plataformas y lenguajes de programación, dada la heterogeneidad, en estos aspectos, que la caracteriza.

Por otro lado durante la década pasada han ido surgiendo con mucha fuerza los sistemas distribuidos en respuesta a la creciente descentralización de los recursos de computación. Entre las tecnologías más relevantes de procesamiento distribuido se encuentran *CORBA*, y *DCOM*. Estas tecnologías utilizan protocolos de comunicación basados en *RPC* como mecanismo para lograr la comunicación entre aplicaciones distribuidas. Este mecanismo implica la capacidad de realizar invocaciones a funciones que, eventualmente, se ejecutan en

máquinas remotas, pasándoles los argumentos requeridos por las mismas y obteniendo como retorno, de las invocaciones, un determinado resultado.

Las tecnologías de procesamiento distribuido mencionadas, funcionan adecuadamente en el ámbito de las empresas [41] donde las plataformas, lenguajes de programación y protocolos de transporte son usualmente homogéneos y estandarizados por la propia empresa y la heterogeneidad no existe o bien es mantenida al mínimo.
En el caso de extender el uso de estas tecnologías al ámbito de `internet`, surgen problemas por las limitaciones que las mismas presentan [11] :

- *son dependientes de la plataforma y de fabricantes*. *DCOM* está directamente relacionado con *Windows NT* y *CORBA*, a pesar de ser una arquitectura abierta, está controlado por determinados vendedores. Por lo tanto lograr que los sistemas distribuidos, construidos usando *DCOM* o *CORBA*, sean accesibles a otros sistemas heterogéneos (escritos o funcionando en otra plataforma) es un problema, a menos que ambas partes tengan la misma plataforma y sistemas similares, porque implica que se generen arquitecturas altamente acopladas, lo cual no es adecuado en el ámbito de `internet`.

- *presentan problemas con los firewalls*. Tanto *DCOM* como *CORBA* asignan puertos en forma dinámica (sección 2.2.7), lo cual dificulta su uso en `internet`, donde los `firewalls` corporativos bloquean normalmente el acceso, a no ser a través de puertos específicos (como puede ser el caso del puerto *HTTP*, por ejemplo).

- *Ambas tecnologías utilizan formatos propietarios de representación de los datos*. Tanto *DCOM* como *CORBA* utilizan formatos binarios y altamente dependientes de la arquitectura de los modelos de componentes respectivos. *DCOM* utiliza el formato *NDR* (**N**etwork **D**ata **R**epresentation) y *CORBA* utiliza el formato, similar pero incompatible, *CDR* (**C**ommon **D**ata **R**epresentation). Esto hace necesario el uso de `software` específico que actúe como un `bridge` a los efectos de lograr la comunicación, lo cual agrega complejidad a las aplicaciones.

En definitiva en el ámbito de `internet`, las tecnologías de procesamiento distribuido existentes presentan serias limitaciones debido a la heterogeneidad del entorno, la presencia de `firewalls` y al uso de formatos propietarios para representar datos.

Recientemente ha surgido el protocolo *SOAP* (sección 2.2) que apunta a resolver algunos de los problemas que presentan las tecnologías de acceso a datos y de procesamiento distribuido, fundamentalmente el referido a la interoperabilidad entre aplicaciones y plataformas heterogéneas, permitiendo el desarrollo de sistemas con un bajo acoplamiento, tal como se requiere en el ámbito de `internet`.

SOAP es una alternativa sumamente útil para comunicar aplicaciones heterogéneas, fundamentalmente por el uso de *XML* que es un estándar basado en texto e independiente de plataformas y lenguajes y por el uso de *HTTP* como transporte, el cual normalmente no es bloqueado por los `firewalls`, dado que no bloquean el puerto *HTTP*.

Para lograr la comunicación a través de `internet` el `framework` utiliza el protocolo *SOAP*. En particular se utilizarán invocaciones *RPC*, encapsuladas en mensajes *SOAP* transportados por *HTTP*. De esta manera se implementa un `pattern` de requerimiento/respuesta mediante un sistema de mensajes *SOAP* sincrónicos. Siendo el comportamiento similar al de una arquitectura cliente servidor.

En la siguiente sección se explica las principales componentes del `framework` relacionadas con *SOAP*, la interacción entre las componentes y el intercambio de mensajes *SOAP* que encapsulan invocaciones *RPC*, tal como se describe en la sección 2.2.6.

3.2.1 Componentes SOAP del `framework`

Actualmente existen muchas implementaciones del protocolo *SOAP*, varias de las cuales son distribuciones sin costo, por lo cual es posible elegir cualquiera de las mismas para su utilización en el `framework`. En general todas las implementaciones proveen una librería para ser usadas por parte de la aplicación cliente (*API SOAP Cli*ente) y el servidor *SOAP*, propiamente dicho, de forma de lograr la comunicación entre aplicaciones a través de `internet`. La figura 14 muestra la arquitectura del `framework` en lo que respecta al uso del protocolo *SOAP* para realizar invocaciones *RPC*, sin los detalles de las componentes *SOAP* y *HTTP*, tanto del lado del cliente como del lado del servidor:

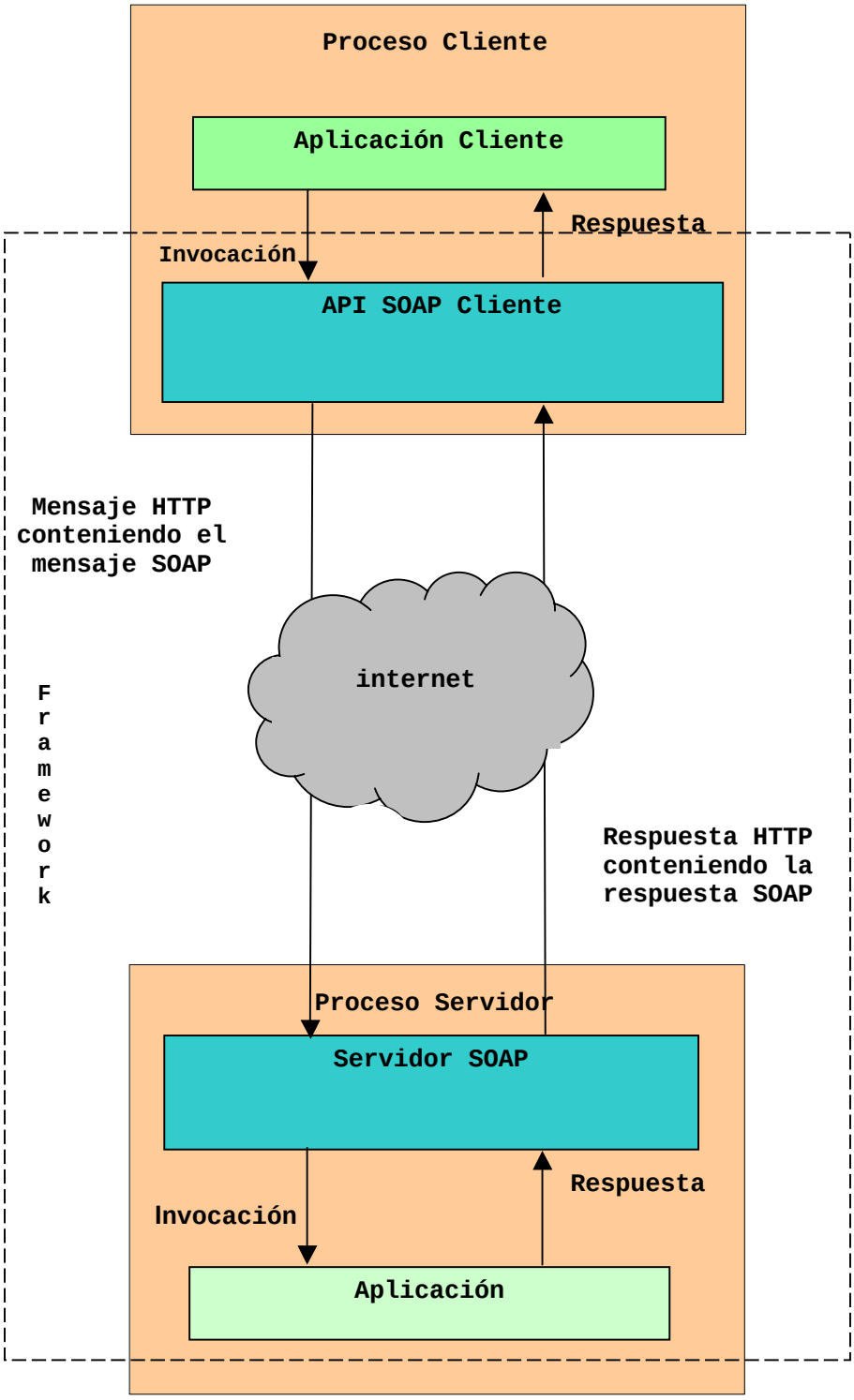


Figura 14 – Arquitectura SOAP del Framework

3.2.1.1 Componentes SOAP en el lado del cliente

La *API SOAP Cliente* es una librería que es usada por las aplicaciones cliente para realizar, por ejemplo, invocaciones *RPC* a métodos remotos, especificando la *URL* (**U**niform **R**esource **L**ocator) donde se encuentra el método remoto y los parámetros necesarios. En este caso, lo primero que realiza la librería es codificar la invocación en un mensaje *SOAP*, luego lo codifica en un mensaje *HTTP* y por último lo envía a la *URL* especificada.
De la misma manera, una vez que se recibe la respuesta del servidor *SOAP*, se utiliza la misma librería para realizar el camino opuesto al anterior, o sea primero se decodifica la respuesta *HTTP*, luego se extrae el mensaje *SOAP* y se lo decodifica para obtener la respuesta a la invocación del método. Por último, esta respuesta es pasada a la aplicación cliente.
La figura 15 muestra en forma gráfica el proceso anterior:

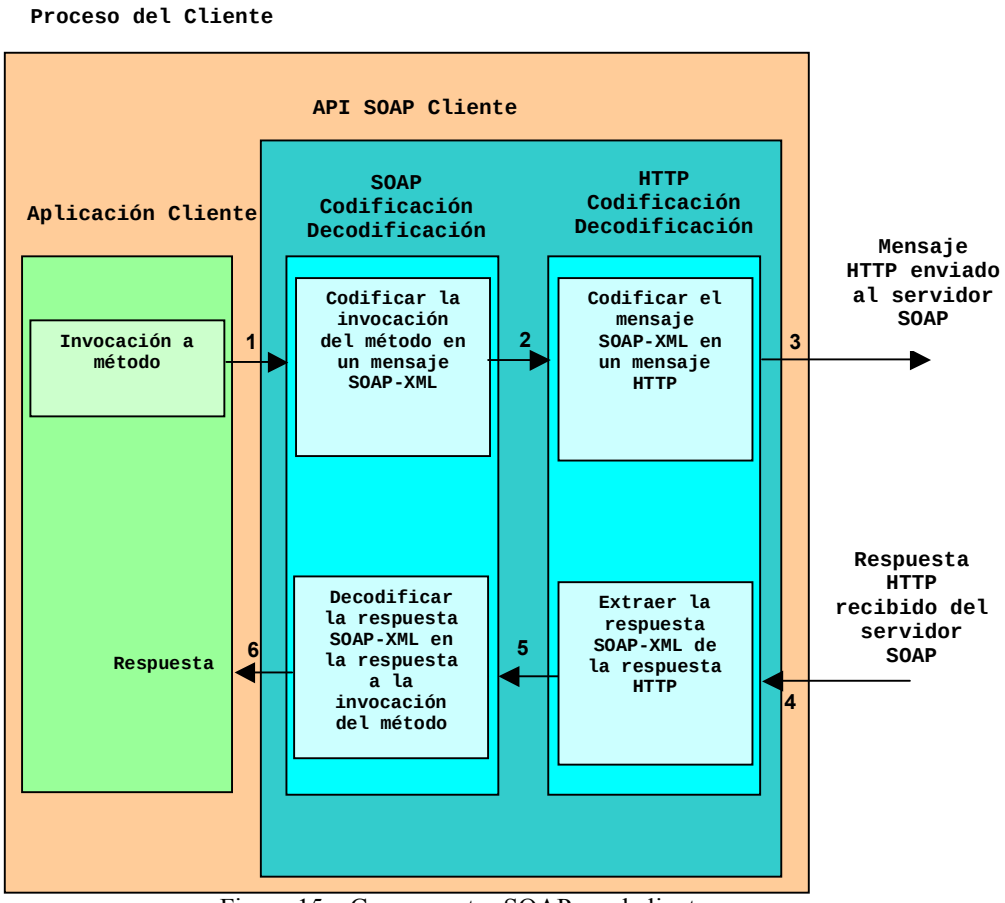


Figura 15 – Componentes SOAP en el cliente

1. La aplicación cliente realiza la invocación al método remoto, utilizando las funcionalidades de la librería *API SOAP Cliente*.

2. La *API SOAP Cliente* codifica la invocación en un mensaje *SOAP* y lo envía codificador *HTTP*.

3. El codificador *HTTP*, codifica el mensaje *SOAP* en un mensaje *HTTP* y lo envía al servidor *SOAP*.

4. La respuesta es recibida por el modulo de codificación y decodificación *HTTP*. Este módulo realiza la decodificación de *HTTP* para extraer la respuesta *SOAP*.

5. Se decodifica la respuesta *SOAP* para extraer la respuesta a la invocación del método.

6. La respuesta es pasada a la aplicación cliente

3.2.1.2 Componentes SOAP en el lado del servidor

El servidor *SOAP* se compone de un proceso *Listener* (*SOAP Listener*) y un módulo para codificar y/o decodificar mensajes *HTTP* y *SOAP*.

Cuando el servidor *SOAP* recibe un mensaje *HTTP*, extrae del mismo el mensaje *SOAP*, lo decodifica para encontrar el nombre del método y sus parámetros y posteriormente invoca al método.
El resultado de la invocación es codificado en un mensaje *SOAP*, que a su vez es codificado en un mensaje *HTTP* y enviado al cliente. La figura 16 muestra en forma gráfica el proceso anterior:

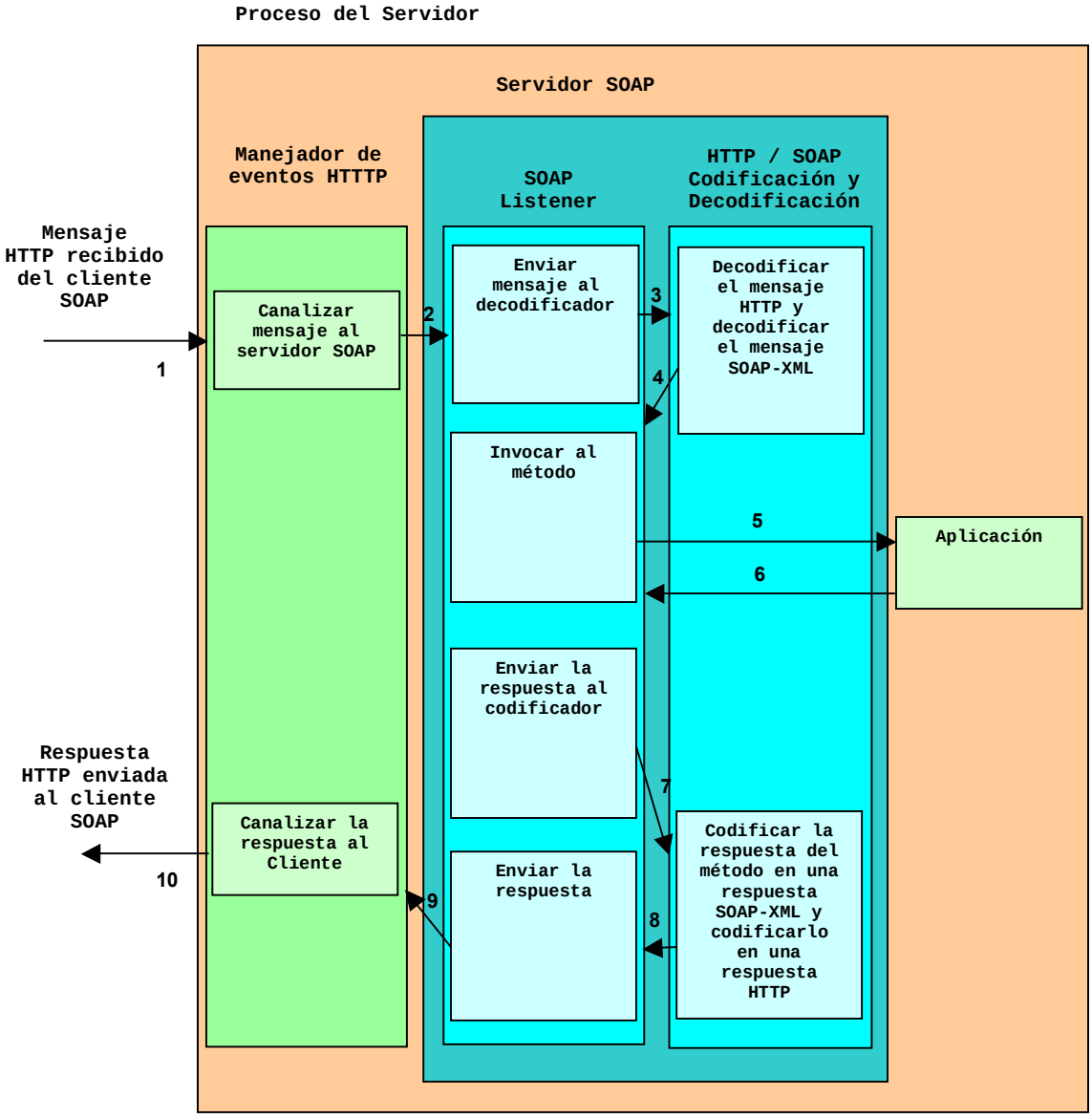


Figura 16 – Componentes SOAP en el servidor

1. El manejador de eventos *HTTP* recibe el mensaje *HTTP* enviado por la aplicación cliente.

2. El mensaje *HTTP* es pasado al servidor *SOAP*.

3 y 4. Se decodifica el mensaje *HTTP* para extraer el mensaje *SOAP*. El mensaje *SOAP* es decodificado para extraer los detalles de la invocación realizada, o sea el nombre del método y los parámetros.

5 y 6. El Servidor *SOAP* realiza la invocación al método y obtiene el resultado de la invocación.

7 y 8. Se codifica la respuesta en un mensaje *SOAP*, que a su vez, es codificado en un mensaje *HTTP*.

9. El mensaje *HTTP* es pasado al manejador de eventos *HTTP*.

10. El mensaje *HTTP* es enviado a la *API SOAP Cliente*.

3.2.1.3 Diagrama de Eventos

El diagrama de la figura 17 explica en detalle la arquitectura de la figura 14 y resume lo detallado en las secciones 3.2.1.1 y 3.2.1.2

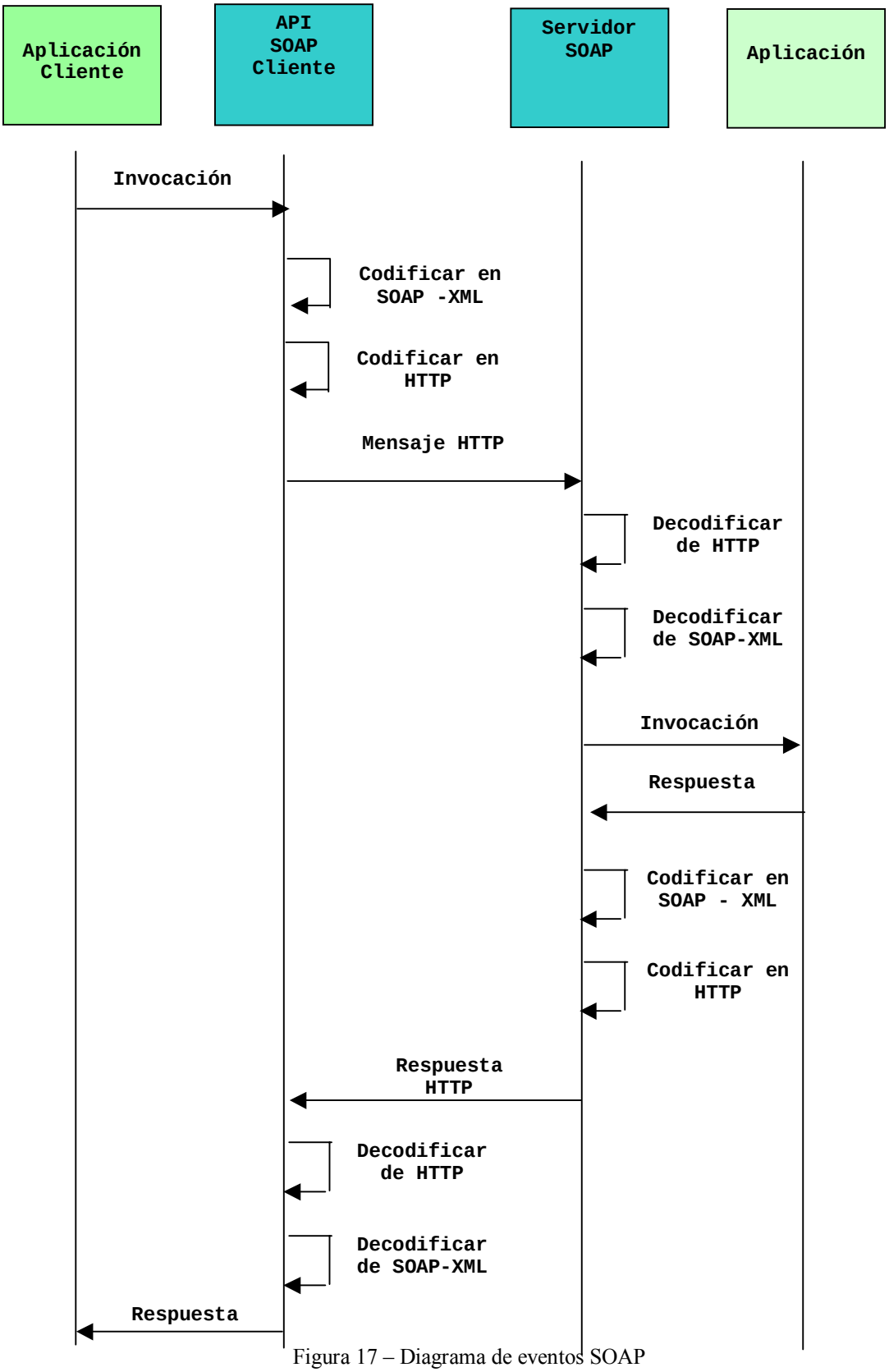


Figura 17 – Diagrama de eventos SOAP

1. La aplicación cliente realiza la invocación al método remoto, utilizando las funcionalidades de la librería *API SOAP Cliente*.

2. La *API SOAP Cliente* codifica la invocación en un mensaje *SOAP* y lo envía codificador *HTTP*.

3. El codificador *HTTP*, codifica el mensaje *SOAP* en un mensaje *HTTP* y lo envía al servidor *SOAP*.

4. El manejador de eventos *HTTP* recibe el mensaje *HTTP* enviado por la aplicación cliente.

5. El mensaje *HTTP* es pasado al servidor *SOAP*.

6. Se decodifica el mensaje *HTTP* para extraer el mensaje *SOAP*. El mensaje *SOAP* es decodificado para extraer los detalles de la invocación realizada, o sea el nombre del método y los parámetros.

7. El Servidor *SOAP* realiza la invocación al método y obtiene el resultado de la invocación.

8. Se codifica la respuesta en un mensaje *SOAP*, que a su vez, es codificado en un mensaje *HTTP*.

9. El mensaje *HTTP* es pasado al manejador de eventos *HTTP*.

10. El mensaje *HTTP* es enviado a la *API SOAP Cliente*.

11. La respuesta es recibida por el modulo de codificación y decodificación *HTTP*. Este módulo realiza la decodificación de *HTTP* para extraer la respuesta *SOAP*.

12. Se decodifica la respuesta *SOAP* para extraer la respuesta a la invocación del método.

13. La respuesta es pasada a la aplicación cliente

Es importante recordar que la elección del lenguaje de programación, de la plataforma y de la *API SOAP Cliente* en el lado del cliente es completamente independiente con respecto a cual es el lenguaje, plataforma y servidor *SOAP* utilizados en el lado del servidor.

3.3 Acceso a las bases de datos

La arquitectura del `framework` descripta en la sección 3.2.1 implica la existencia de una aplicación en el lado del servidor que brinda un servicio, actuando como intermediaria entre la aplicación cliente y los manejadores de bases de datos. Desde el punto de vista de la aplicación cliente esta aplicación actúa como una *API* (**A**pplication **P**rogram **I**nterface), constituyendo el núcleo del `framework` y brindando determinadas funcionalidades a la misma. Por este motivo de aquí en adelante se hablará en términos de *ANS* (**A***PI* **N**úcleo del **S**istema) en vez de aplicación. La descripción de *ANS* se encuentra en la sección 3.3.4.

ANS es responsable de definir como las aplicaciones cliente acceden a las bases de datos relacionales para la realización de consultas. Existen por lo menos dos soluciones relativamente simples para este problema, entre las que se encuentran usar el lenguaje de consulta *SQL* directamente o bien programar determinadas consultas a los efectos que sean usadas por *ANS* como primitivas.

Si las funciones de *ANS* utilizan directamente el lenguaje de consulta *SQL* se logra una gran flexibilidad dado que las aplicaciones cliente pueden realizar sus propias consultas. Sin embargo no abstrae a las aplicaciones cliente del diseño lógico de la base de datos, dado que requiere que las mismas conozcan como se relacionan las tablas de la base de datos entre sí, o sea que deben conocer y entender como es el diseño lógico de la base de datos. En esta situación existe un alto acoplamiento entre las aplicaciones cliente y el diseño lógico de la base de datos, a pesar de lo cual esta solución no se descarta.

Si las funciones de *ANS* son consultas programadas, ya sea porque se dispone de funciones o `stored procedures` previamente definidos, se logra abstraer a las aplicaciones cliente del diseño lógico de la base de datos, dado que las mismas estarían accediendo a la base de datos a través de consultas predefinidas y no tienen porque conocer el diseño de la base de datos. Sin embargo se trata de una solución que no es flexible, dado que solamente permitiría realizar determinadas consultas a las aplicaciones cliente, lo cual no está de acuerdo con lo planteado en la sección 3.1.

Por otro lado, actualmente los principales vendedores de bases de datos, como ser IBM, Oracle y Microsoft SQL Server (sección 2.7), incorporan soluciones en sus manejadores para que las aplicaciones cliente utilicen lenguajes de consulta propios de *XML* y obtengan los resultados de dichas consultas en formato *XML*, sin embargo se trata, en todos los casos, de soluciones propietarias y no generales por lo cual se presentan problemas de independencia de plataformas y lenguajes de programación mencionados en secciones anteriores, lo cual descarta estas soluciones.

Por lo tanto para lograr lo expuesto la sección 3.1, se definirá la *API ANS* (sección 3.3.4), para cuyo diseño y especificación se analizarán en las siguientes secciones soluciones concretas.

3.3.1 XML para intercambio de información

XML (sección 2.3) se está transformando rápidamente en el formato estándar para intercambio de información en el ámbito de `internet`. Por lo tanto es lógico utilizar *XML* para que las funciones de *ANS* reciban y devuelvan datos.
Utilizar *XML* como formato para el intercambio de información tiene, además, varias ventajas entre las que se destacan:

- *XML* permite representar datos de manera independiente de plataformas y lenguajes.
- Está disponible en todas las plataformas.
- Es adecuado para manipular datos estructurados y tipeados.

3.3.2 Abstracción del diseño lógico de la base de datos

Una solución general y flexible, provista por el modelo relacional, para lograr una mayor transparencia del diseño lógico, es definir vistas de la base de datos. Dado que el `framework` está pensado fundamentalmente para su utilización en el ámbito de `internet` donde *XML* tiene un papel preponderante, por las razones ya expuestas, es razonable que la aplicación cliente vea las estructuras de datos (vistas) sobre las cuales va a realizar consultas en *XML*.

Una vez que la vista ha sido creada, se debe definir como las aplicaciones cliente usan dicha vista. Una solución simple es materializar la vista a requerimiento de las aplicaciones y retornar un documento *XML* conteniendo los datos. Este enfoque presenta dos problemas, por un lado la actualización de los datos de las vistas que debe ser realizada cada vez que cambia la base de datos y por otro lado que, en muchos casos, las aplicaciones no requieren la totalidad de la vista sino partes de la misma.
Una solución mejor es definir una vista "virtual" de la base de datos relacional, mediante un documento *XML* que especifica cual es la estructura de la vista pero que no contiene datos, sino que los datos siguen residiendo en la base de datos relacional. Para la especificación de la vista se utiliza un *XML Schema* (sección 2.4).

Esta solución tiene el inconveniente que las aplicaciones cliente están obligadas a manejar distintos lenguajes en un mismo programa, por un lado *XML* para la representación de los datos y por otro lado *SQL* para la realización de consultas. Además, si se adopta esta solución como única se restringe la utilización del `framework` a aquellos usuarios que conozcan el lenguaje *SQL* y el modelo relacional.

Para hacer que las aplicaciones cliente sean más homogéneas en lo relativo a los lenguajes que deben soportar, lo cual evita potenciales problemas en su programación y asimismo para ampliar el abanico de los posibles usuarios, se propone utilizar, también, vistas *XML* de la base de datos relacional que permiten ocultar el diseño lógico (relacional) y soportar consultas sobre estas vistas usando un lenguaje adecuado (sección 3.3.3).

No forma parte de la presente tesis como crear las vistas *XML* de la base de datos, sin embargo más adelante se brindarán determinadas reglas para la creación de estas vistas (sección 3.5).

ANS permite a las aplicaciones cliente obtener los *XML Schema* que especifican las vistas de la base de datos, ya sean relacionales o *XML*.

El hecho que las aplicaciones cliente obtengan las especificaciones de las vistas *XML*, permite abstraer, a las mismas, del diseño lógico de la base de datos, dado que no requiere que conozcan cuales son las tablas, como se relacionan entre sí, cuales son las claves primarias, etc.

3.3.3 Especificación de las consultas

Para lograr flexibilidad para realizar consultas es necesario disponer de un lenguaje de consultas. Como ya se mencionó no se descarta el uso del lenguaje *SQL*.
Además dado que *ANS* permite a las aplicaciones cliente obtener las especificaciones de vistas *XML* de la base de datos, se puede lograr una gran flexibilidad soportando consultas sobre dichas vistas, utilizando lenguajes de consulta propios de *XML* (sección 2.5) a los efectos que las aplicaciones cliente realicen sus consultas y sean más homogéneas en lo relativo a los lenguajes que deben soportar.

Una observación importante es que cualquiera sea el lenguaje de consulta *XML* elegido, *ANS* deberá convertir las consultas escritas en dicho lenguaje al lenguaje *SQL* a los efectos de acceder a la base de datos relacional.

De la misma manera *ANS* debe transformar los conjuntos de tuplas que se obtienen como resultado de la consulta *SQL* a la base de datos en documentos *XML*.

3.3.4 *API* Núcleo del Sistema (ANS)

En esta sección se presenta la *API ANS*, la cual es una adaptación de la *API XML for Analysis* [31], explicada en la sección 2.6. La especificación completa de *ANS* se encuentra en el Anexo 1.

Esta *API* define dos métodos, **Descubrir** y **Ejecutar**, los cuales reciben y devuelven datos en formato *XML*, permitiendo el descubrimiento y la consulta de datos, funcionando en un régimen `stateless`, o sea que no se mantiene la identidad o el contexto de la aplicación cliente después que la invocación a un método finaliza.

Ambos métodos, pero fundamentalmente el método **Descubrir**, presentan una `interface` genérica basada en la estructura de los parámetros para determinar su comportamiento. Esto permite la extensibilidad de ambos métodos (aprovechando la extensibilidad de *XML*) si se requiere modificar algún parámetro, manteniendo la misma `interface`.

La especificación de *ANS* se basa en el estándar *XML* y es independiente de un lenguaje de programación o plataforma particular.

Los objetivos de diseño de *ANS* son los siguientes:

- Proveer una *API* que permita la interacción en el acceso remoto a datos entre una aplicación cliente y bases de datos relacionales en el ámbito de `internet`, de acuerdo con lo planteado en la sección 3.1.
- Funcionar en régimen `stateless`.
- Ser tecnológicamente independiente de lenguajes de programación y plataformas.
- Utilizar el estándar *XML* para el intercambio de información.

ANS tiene las siguientes características funcionales:

1. Permitir que las aplicaciones cliente obtengan las especificaciones de las vistas *XML* creadas sobre las bases de datos relacionales (*XML Schemas*) o bien la estructura relacional (mediante un *XML Schema* adecuado) correspondiente a una vista o a la base de datos.
2. Utilizar un lenguaje de consulta para que las aplicaciones cliente realicen sus consultas.
3. Procesar las consultas del cliente. En caso de no estar escritas en *SQL*, convirtiéndolas a *SQL* para su resolución por parte del manejador de la base de datos relacional.
4. Convertir los conjuntos de tuplas (`resultset`) obtenidos como resultado de la consulta *SQL* a formato *XML* a los efectos de retornarlos a la aplicación cliente.

La figura 18 muestra el funcionamiento de *ANS*:

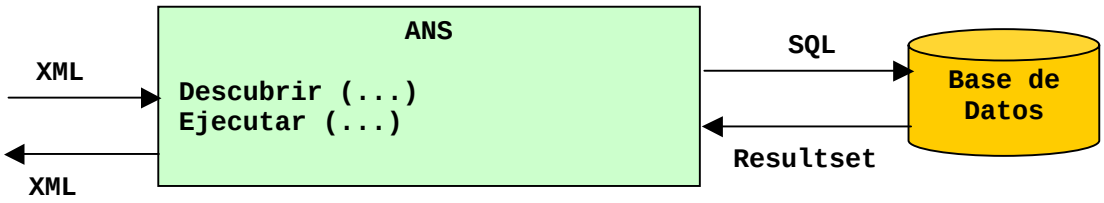


Figura 18 – ANS (API Núcleo del Sistema)

El método **Descubrir** es usado por la aplicación cliente para obtener información. La información que se obtiene puede incluir una lista de las bases de datos disponibles, una lista de las vistas (*XML* y/o relacionales) creadas para cada base de datos, los *XML Schemas* que especifican estas vistas e información específica para acceder a los datos, por ejemplo `password` y usuario. La información obtenida por la aplicación cliente depende de los valores de los parámetros con que se invoca al método.

El método **Ejecutar** es usado por la aplicación cliente para realizar consultas, utilizando un lenguaje de consulta (*SQL* o *XML*).

3.4 Resumen del framework

El framework propuesto representa una solución al problema de permitir el acceso a bases de datos a aplicaciones cliente a través de internet, basado en las siguientes características:

- Construido sobre estándares abiertos de internet:
  - *SOAP* como protocolo de comunicación.
  - *XML* como formato de intercambio de datos.

  de manera que el framework sea tecnológicamente independiente de lenguajes de programación y plataformas y que exista un bajo acoplamiento con el servidor de bases de datos.
- Brindar la posibilidad de abstraer a las aplicaciones cliente del diseño lógico de la base de datos (transparencia del diseño lógico).
- Brindar flexibilidad para la realización de consultas, utilizando *SQL* o bien lenguajes de consulta de *XML*.

Para lograr los objetivos anteriores se realiza la especificación de la *API ANS* que actúa como intermediaria entre las aplicaciones cliente y los manejadores de bases de datos, constituyendo el núcleo del sistema. Las aplicaciones cliente se comunican con *ANS* utilizando el protocolo *SOAP*. *ANS* tiene las siguientes características funcionales:

- Permitir que las aplicaciones cliente obtengan las especificaciones de las vistas *XML* creadas sobre las bases de datos relacionales o bien la estructura relacional correspondiente a una vista o a la base de datos.
- Utilizar un lenguaje de consulta de *XML* para que las aplicaciones cliente realicen sus consultas sobre la especificación de las vistas *XML* de la base de datos, o bien utilicen el lenguaje *SQL*.
- Procesar las consultas del cliente. En caso de no estar escritas en *SQL*, convirtiéndolas a *SQL* para su resolución por parte del manejador de la base de datos relacional.
- Convertir los conjuntos de tuplas obtenidos como resultado de la consulta *SQL* a formato *XML* a los efectos de retornarlos a la aplicación cliente.

Además el framework propuesto presenta una solución general y abierta al problema de permitir el acceso a bases de datos a aplicaciones cliente dado que:

- Provee una *API* de acceso a datos que puede ser usada universalmente en internet.
- Es tecnológicamente independiente de lenguajes de programación y plataformas por estar construido sobre estándares abiertos, como ser *SOAP*, *XML*.
- Es posible utilizar distintos lenguajes de consulta, dado que la especificación de *ANS* no mandata el uso de un lenguaje en particular.
- El framework es independiente de una implementación particular del protocolo *SOAP*, tanto en el lado del cliente como en el lado del servidor.
- El framework es independiente de la implementación en un lenguaje particular de *ANS* o a su funcionamiento en una plataforma particular.

A pesar que el framework está diseñado para su utilización en el ámbito de internet, también puede ser usado en intranets. Los siguientes son ejemplos de aplicaciones que pueden beneficiarse del framework propuesto, en intranets:

- Aplicaciones cliente servidor que requieren flexibilidad tecnológica en los clientes y en el servidor.
- Aplicaciones cliente servidor que utilizan múltiples plataformas.

La figura 19 muestra la arquitectura del `framework`, basada en la utilización de *SOAP* y de la *API ANS*. En el siguiente capítulo se presenta un posible lenguaje de consulta *XML*, entre los muchos posibles, para ser utilizado por las aplicaciones cliente en las invocaciones al método **Ejecutar**.

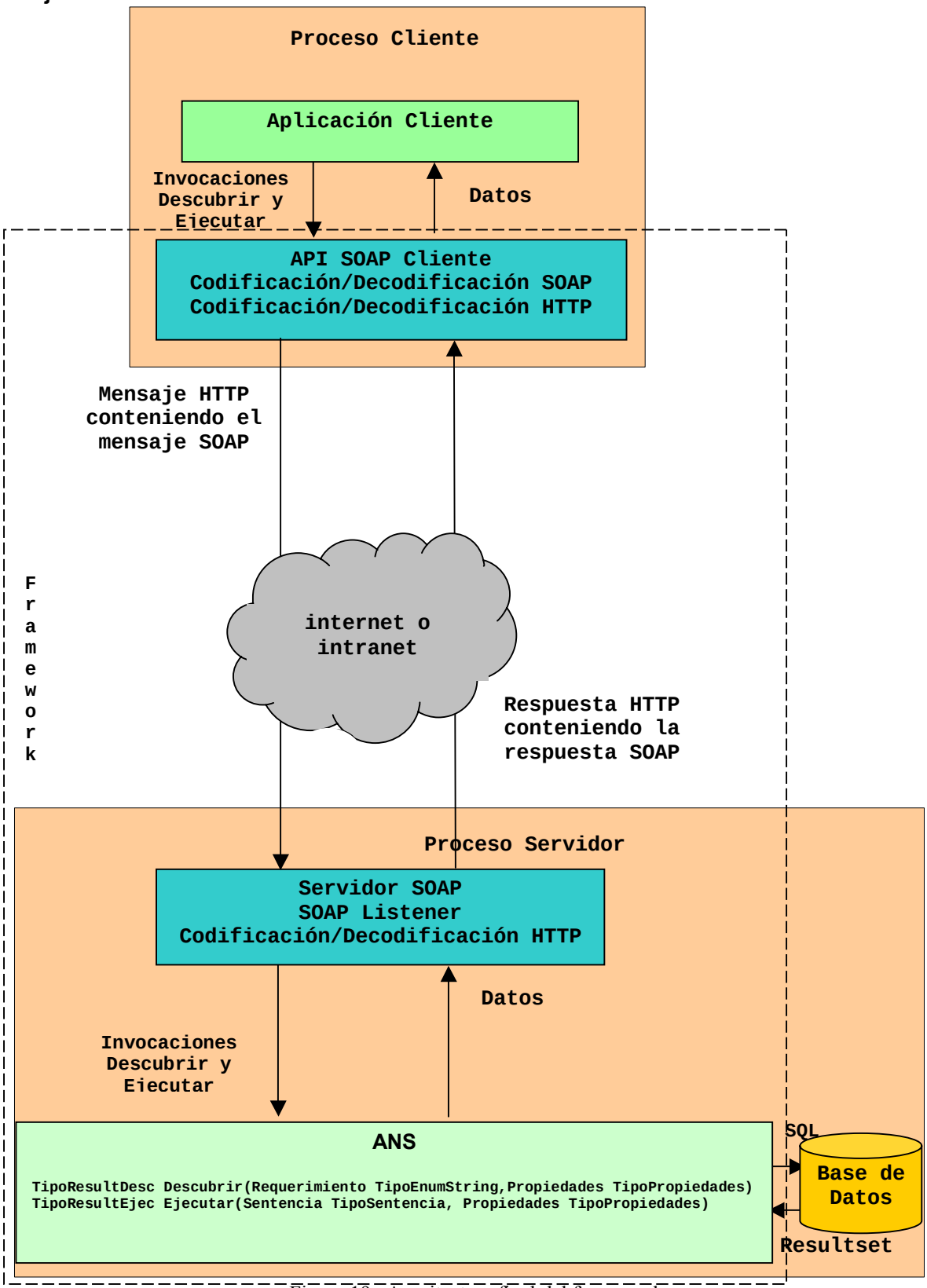


Figura 19 –Arquitectura final del framework

3.5 Reglas para definir la vista XML de la base de datos

Las vistas *XML* de la base de datos relacional son especificadas mediante un *XML Schema* (sección 2.4). La estructura del *XML Schema* determina como las aplicaciones cliente realizan las consultas, utilizando un lenguaje de consulta *XML* e influye en la conversión de las consultas a *SQL*.
En esta sección se definirán un conjunto de reglas que deberán ser tenidas en cuenta al definir el *XML Schema* que especifica la vista.

Antes de definir las reglas mencionadas, existen varios problemas técnicos involucrados en la generación de documentos *XML* a partir de base de datos relacionales, debido a las diferencias de los modelos en que se basan.

En primer lugar, en el modelo relacional una base de datos es una colección de tablas. Cada tabla consiste en un conjunto de tuplas y cada tupla tiene una cantidad fija de columnas. En cambio, en *XML*, los documentos son árboles de profundidad arbitraria.
Por lo tanto, desde un punto de vista estructural, un documento *XML* puede ser usado para representar una base de datos relacional.

En los documentos *XML*, todos los elementos *XML* se definen en el mismo alcance, por lo cual todos deben tener nombres diferentes. Los elementos *XML* pueden tener atributos y diferentes elementos *XML* pueden tener atributos con el mismo nombre.
Esto último recuerda las restricciones del modelo relacional: el nombre de las tablas deber ser diferente y tablas diferentes pueden tener columnas con el mismo nombre. Esto sugiere que es posible establecer una correspondencia entre tablas y elementos *XML* y una correspondencia entre atributos *XML* y columnas de tablas [42].
Basado en este enfoque de correspondencias, que puede denominarse basado en atributos, cada tabla se corresponde con un elemento *XML* vacío, es decir sin contenido, y las columnas de la tabla se corresponden con los atributos del elemento *XML*.

Por ejemplo, a partir de la tabla relacional ***Fabricantes*** *(numero_fabricante, direccion, nombre)* y aplicando el enfoque basado en atributos para establecer las correspondencias, se genera el siguiente fragmento de *XML Schema*:

```
<xs:element name = “fabricantes” minOccurs = “0” maxOccurs = “unbounded”>
  <xs:complexType contenent=”elementOnly”>
    <xs:attribute name=”numero_fabricante” type=”xs:positiveinteger”/>
    <xs:attribute name=”direccion” type=”xs:string”/>
    <xs:attribute name=”nombre” type=”xs:string”/>
  </xs:complextype>
</xs:element>
```

Figura 20 – Fragmento de XML Schema para una tabla

Se observa que el enfoque basado en atributos tiene como ventaja principal que genera un *XML Schema* sumamente compacto y simple [43].

El enfoque basado en atributos es una posible solución para resolver las diferencias entre el modelo relacional y el modelo *XML*. Sin embargo otro problema a resolver es que, generalmente, en una vista de la base de datos interviene más de una tabla. Por lo tanto, en general, es necesario definir una forma de agregación jerárquica de dichas tablas, que determine un orden de anidamiento de las tablas, a los efectos de representar la vista en *XML*.
Para resolver este problema, se puede utilizar la relación de clave foránea y clave primaria existente entre tablas en el modelo relacional.

Una base de datos relacional puede modelarse como un grafo, en el cual los vértices representan las tablas y las aristas son las relaciones de clave foránea y clave primaria entre tablas.
El grafo es no dirigido porque las relaciones de clave foránea y clave primaria pueden ser navegadas en ambos sentidos.

Basado en el grafo, es posible definir una vista *XML* de la base de datos relacional [44] eligiendo, en primer lugar, el subconjunto de vértices que integran la vista, luego se asignan direcciones a las aristas, entre los vértices elegidos, de manera de generar un subgrafo dirigido y acíclico, en el cual el grado de entrada de cada vértice sea a lo sumo 1. Por último se selecciona el vértice inicial que se desee, o bien se elige como vértice inicial un vértice con grado de entrada 0 [45] o se utiliza alguno de los otros criterios indicados en [45] para seleccionar el vértice inicial.
Al recorrer el subgrafo, a partir del vértice inicial seleccionado y siguiendo las direcciones de las aristas, se obtiene una agregación jerárquica de tablas que corresponde a la vista *XML* de la base de datos.
Esta recorrida define el *XML Schema* que especifica la vista *XML* de la base de datos. Por las razones que se explican en la sección 4.4.3 no deberán existir nombres de elementos y atributos *XML* repetidos.

Por restricciones de *XML*, los documentos *XML* deben tener un único elemento[4] que contenga a todos los demás, por lo cual se deberá crear un elemento *XML*, que tenga como subelementos los elementos correspondientes a las tablas. Este elemento ficticio a agregar consiste en la etiqueta <*Vista*> y es un elemento vacío.

El siguiente ejemplo muestra el proceso descripto:

1. Dada la base de datos relacional
   Fabricantes *(numero_fabricante, dirección, nombre)*
   Productos *(numero_producto, descripción)*
   Ventas *(numero_fabricante, numero_producto, precio, fecha_venta, fecha_envio)*

   Donde la tabla ***Ventas*** tiene una relación de clave foránea con la tabla Fabricantes (columna *numero_fabricante*) y con la tabla ***Productos*** (columna *numero_producto*).

2. Se modela mediante el siguiente grafo:

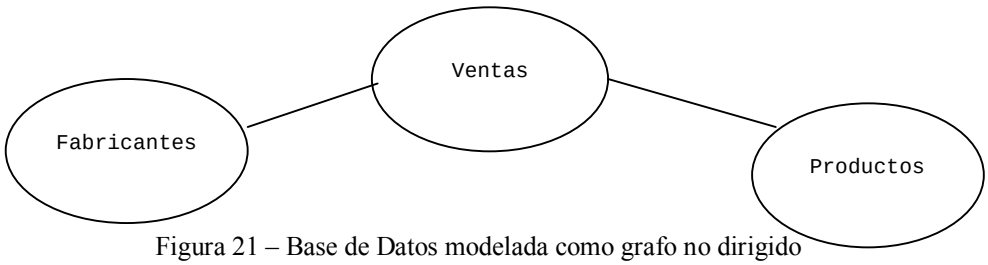


Figura 21 – Base de Datos modelada como grafo no dirigido

3. Se eligen las tablas que integran la vista (en este caso todas) y se asignan direcciones a las aristas, generándose el siguiente grafo dirigido:

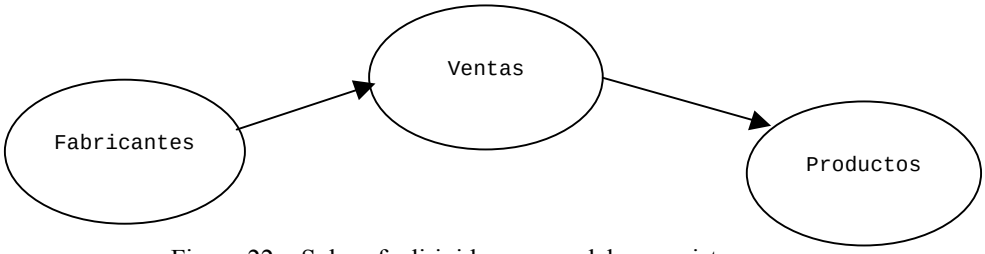


Figura 22 – Subgrafo dirigido que modela una vista

4. Se elige como vértice inicial el vértice que corresponde a la tabla ***Fabricantes***.

[4] Este elemento representa la totalidad del documento *XML* y se denomina elemento documento.

5. Se genera el *XML Schema* que especifica la vista *XML* de la base de datos, recorriendo los vértices a partir del vértice inicial y siguiendo las direcciones de las aristas, agregando como elemento documento un elemento con la etiqueta <*Vista*>.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name = "Vista">
  <xs:complexType contenent="elementOnly">
   <xs:element name="fabricante" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType contenent="elementOnly">
     <xs:element name="venta" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType contenent="elementOnly">
       <xs:element name="producto" minOccurs="1" maxOccurs="1">
        <xs:complexType contenent="elementOnly">
         <xs:attribute name="numero_producto" type= "xs:string"/>
         <xs:attribute name="descripcion" type ="xs:string"/>
        </xs:complexType>
       </xs:element>
       <xs:attribute name="numero_fabricante" type= "xs:positiveinteger"/>
       <xs:attribute name="numero_producto" type= "xs:string"/>
       <xs:attribute name="precio" type= "xs:decimal"/>
       <xs:attribute name="fecha_venta" type= "xs:date"/>
       <xs:attribute name="fecha_envio" type= "xs:date"/>
      </xs:complexType>
     </xs:element>
     <xs:attribute name="numero_fabricante" type= "xs:positiveinteger"/>
     <xs:attribute name="direccion" type= "xs:string"/>
     <xs:attribute name="nombre" type= "xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 23 – XML Schema correspondiente a una vista XML (primera versión)

Se puede observar que en el *XML Schema* del ejemplo existen atributos *XML* que son redundantes, porque aparecen en varios elementos *XML*. Los atributos *XML* redundantes deben ser eliminados. En el *XML Schema* del ejemplo, se puede eliminar el atributo *XML numero_fabricante* del elemento *fabricante* o del elemento *venta*, de la misma manera el atributo XML *numero_producto*, puede ser eliminado del elemento *venta* o del elemento *producto*.
Si se elimina el atributo *numero_fabricante* del elemento *venta* y el atributo *numero_producto* del elemento *producto*, el *XML Schema* del ejemplo queda de la siguiente manera:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name = "Vista">
  <xs:complexType contenent="elementOnly">
   <xs:element name="fabricante" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType contenent="elementOnly">
     <xs:element name="venta" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType contenent="elementOnly">
       <xs:element name="producto" minOccurs="1" maxOccurs="1">
        <xs:complexType contenent="elementOnly">
         <xs:attribute name="descripcion" type ="xs:string"/>
        </xs:complexType>
       </xs:element>
       <xs:attribute name="numero_producto" type= "xs:string"/>
       <xs:attribute name="precio" type= "xs:decimal"/>
       <xs:attribute name="fecha_venta" type= "xs:date"/>
       <xs:attribute name="fecha_envio" type= "xs:date"/>
      </xs:complexType>
     </xs:element>
     <xs:attribute name="numero_fabricante" type= "xs:positiveinteger"/>
     <xs:attribute name="direccion" type= "xs:string"/>
     <xs:attribute name="nombre" type= "xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 24 – XML Schema correspondiente a una vista XML

En resumen, para definir el *XML Schema* que especifica la vista *XML* de la base de datos relacional, se deberán tener en cuenta las siguientes reglas:

- Utilizar el enfoque basado en atributos para representar cada tabla.
- Utilizar las relaciones de clave foránea y clave primaria entre tablas para generar una agregación jerárquica.
- Verificar que no existan nombres de elementos y atributos *XML* repetidos.
- Eliminar atributos *XML* redundantes
- Definir el *XML Schema* que especifica la vista *XML* de la base de datos relacional, agregando un elemento documento (que contenga los otros elementos) de etiqueta <*Vista*>

En el *XML Schema* pueden usarse nombres para los elementos y atributos *XML* diferentes que los nombres de las tablas y columnas correspondientes de la base de datos. Tampoco es necesario incluir todas las columnas de una tabla como atributos *XML* del elemento *XML* correspondiente a la tabla, si se desea.

El *XML Schema* que especifica la vista *XML* de la base de datos relacional, generado siguiendo las reglas indicadas, tiene las siguientes características:

- Existen dos tipos de elementos *XML*: los elementos *XML* vacíos (`empty content`, sección 2.3) que corresponden a las tablas y los atributos *XML* que corresponden a las columnas de las tablas.
- No existen nombres de elementos y atributos *XML* repetidos.
- No existen atributos *XML* redundantes.
- La relación padre hijo entre elementos *XML*, que define la agregación jerárquica, está dada por la relación de clave foránea y clave primaria entre las tablas que corresponden a los elementos.
- Existe un elemento de etiqueta <*Vista*> que es el elemento documento.

La generación del *XML Schema* que especifica la vista *XML* de la base de datos no forma parte de esta tesis y por lo tanto debe ser realizada por una persona, siguiendo las reglas indicadas. Una vez generadas las vista las mismas son obtenidas por las aplicaciones cliente mediante las funcionalidades que brinda la *API ANS* (sección 3.3.4).

3.6 Ejemplo de uso del `Framework`

Para ilustrar como una aplicación cliente utiliza el `framework` para realizar consultas a bases de datos relacionales, se muestra un ejemplo completo de uso del mismo, asumiendo que se utilizan vistas *XML*.

La aplicación cliente busca las bases de datos disponibles, para lo cual invoca al método **Descubrir** con el valor DESCUBRIR_BASES en el parámetro *Requerimiento* y el valor nulo en el parámetro *Propiedades*.
El siguiente es el código Java de la aplicacion cliente utilizando Apache *SOAP* [46], versión 2.2, que es la implementación de Apache Software Foundation [47] de *SOAP*:

```
String opcion = new String("DESCUBRIR_BASES");
Element ListaPropiedad = doc.createElement("ListaPropiedad");
URL url;
url = new URL(http://host/apache_soap22/servlet/rpcrouter);
Call call = new Call();
call.setTargetObjectURI("urn:xpath");
call.setMethodName("Descubrir");
call.setEncodingStyleURI(Constants.NS_URI_LITERAL_XML);
Vector params = new Vector();
params.addElement(new Parameter("Requerimiento", String.class,
                  opcion, Constants.NS_URI_SOAP_ENC));
params.addElement(new Parameter("Propiedades", Element.class,
                 ListaPropiedad, Constants.NS_URI_LITERAL_XML));
call.setParams(params);
Response resp;
resp = call.invoke(url, "");
```

Figura 25 - Invocación (Java) al método Descubrir para obtener las bases de datos disponibles

El código anterior genera el siguiente mensaje *SOAP* que es enviado a la *URL* del servidor *SOAP*[5] :

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 587
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:Descubrir
         xmlns:ns1="urn:xpath"
         SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
         <Requerimiento xsi:type="xsd:string"
            SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            DESCUBRIR_BASES
         </Requerimiento>
         <Propiedades>
            <ListaPropiedad/>
         </Propiedades>
      </ns1:Descubrir>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 26 - Invocación (SOAP) al método Descubrir para obtener las bases de datos disponibles

[5] En negrita se muestran las invocaciones a los métodos de *ANS* y sus parámetros y los resultados de las invocaciones.

Como respuesta, el servidor *SOAP*, envía a la aplicación cliente una lista de las bases de datos disponibles:

```
 HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 764
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:DescubrirResponse
         xmlns:ns1="urn:xpath"
         SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
         <return>
          <Root>
            <!— Definicion del schema del resultado -->
            ...
            <BaseDatos>MiBase
              <Descripcion>Descripcion de MiBase</Descripcion>
              <ModoAutenticacion>NoRequerido</ModoAutenticacion>
            </BaseDatos>
            <BaseDatos>OtraBase
               <Descripcion>Descripcion de OtraBase</Descripcion>
              <ModoAutenticacion>NoRequerido</ModoAutenticacion>
            </BaseDatos>
          </Root>
        </return>
      </ns1:DescubrirResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 27 - Respuesta (SOAP) al método Descubrir para obtener las bases de datos disponibles

La aplicación cliente obtiene el resultado de la invocación anterior mediante el siguiente código Java:

```
Parameter ret = resp.getReturnValue();
Element E = (Element)ret.getValue();
```

Figura 28 - Resultado (Java) de la invocación.

la variable `E` es inicializada con el documento *XML* que se encuentra entre la etiqueta `<root>` en el mensaje *SOAP* enviado por el servidor.

La aplicación cliente analiza el resultado obtenido y selecciona la base de datos a usar.

De aquí en adelante solamente se muestran los mensajes *SOAP* que son enviados al servidor *SOAP* porque el código Java es similar en todos los casos y lo único que varía es el nombre del método y sus correspondientes parámetros. Además el código depende de la implementación de *SOAP* que se utilice mientras que los mensajes *SOAP* son, en general, más estándares.

La aplicación cliente busca las vistas disponibles de la base de datos seleccionada, para lo cual invoca al método **Descubrir** con el valor DESCUBRIR_VISTAS en el parámetro *Requerimiento* y el nombre de la base de datos en el parámetro *Propiedades*:

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 587
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:Descubrir xmlns:ns1="urn:xpath"
      SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
      <Requerimiento xsi:type="xsd:string"
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         DESCUBRIR_VISTAS
     </Requerimiento>
     <Propiedades>
      <ListaPropiedad>
       <BaseDatos>MiBase</BaseDatos>
      </ListaPropiedad>
     </Propiedades>
    </ns1:Descubrir>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 29 - Invocación (SOAP) al método Descubrir para obtener las vistas XML disponibles

Como respuesta se envía a la aplicación cliente la lista de las vistas disponibles en la base de datos indicada:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 764
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:DescubrirResponse
      xmlns:ns1="urn:xpath"
      SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
     <return>
      <Root>
       <!— Definicion del schema del resultado -->
       ...
        <BaseDatos>MiBase
        <Vista>VistaA
           <Descripcion>Descripcion de la vista A</Descripcion>
         </Vista>
        <Vista>VistaB
           <Descripcion>Descripcion de la vista B</Descripcion>
        </Vista>
       </BaseDatos>
      </Root>
     </return>
    </ns1:DescubrirResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 30 - Respuesta (SOAP) al método Descubrir para obtener las vistas XML disponibles

La aplicación cliente analiza el resultado obtenido y selecciona la vista a usar.

La aplicación cliente busca el *XML Schema* que define la vista seleccionada, para lo cual invoca al método **Descubrir** con el valor DESCUBRIR_SCHEMA_VISTA en el parámetro *Requerimiento* y el nombre de la base de datos y de la vista en el parámetro *Propiedades*:

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 674
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:Descubrir
         xmlns:ns1="urn:xpath"
         SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
         <Requerimiento
            xsi:type="xsd:string"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            DESCUBRIR_SCHEMA_VISTA
         </Requerimiento>
         <Propiedades>
            <ListaPropiedad>
               <BaseDatos>MiBase</BaseDatos>
               <Vista>VistaA</Vista>
            </ListaPropiedad>
         </Propiedades>
      </ns1:Descubrir>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 31 - Invocación (SOAP) al método Descubrir para obtener el XML Schema de una vista

como respuesta se envía el *XML Schema* correspondiente a la vista seleccionada. Por ejemplo, suponiendo que *MiBase* es la base de datos de la sección 3.5 y que *VistaA* es la vista mostrada en la sección 3.5:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 1601
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <SOAP-ENV:Body>
  <ns1:DescubrirResponse
   xmlns:ns1="urn:xpath"
   SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
   <return>
    <Root>
     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
       <xs:element name = "Vista">
        <xs:complexType contenent="elementOnly">
         <xs:element name="fabricante" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType contenent="elementOnly">
           <xs:element name="venta" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType contenent="elementOnly">
             <xs:element name="producto" minOccurs="1" maxOccurs="1">
              <xs:complexType contenent="elementOnly">
               <xs:attribute name="descripcion" type ="xs:string"/>
              </xs:complexType>
             </xs:element>
```

```
                <xs:attribute name="numero_producto" type= "xs:string"/>
                <xs:attribute name="precio" type="xs:decimal"/>
                <xs:attribute name="fecha_venta" type= "xs:date"/>
                <xs:attribute name="fecha_envio" type= "xs:date"/>
              </xs:complexType>
            </xs:element>
           <xs:attribute name="numero_fabricante"type="xs:positiveinteger"/>
           <xs:attribute name="direccion" type= "xs:string"/>
           <xs:attribute name="nombre" type="xs:string"/>
         </xs:complexType>
       </xs:element>
     </xs:complexType>
   </xs:element>
  </xs:schema>
 </Root>
</return>
</ns1:DescubrirResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 32 - Respuesta (SOAP) al método Descubrir para obtener el XML Schema de una vista

La aplicación cliente utiliza el *XML Schema* anterior para realizar las consultas usando un lenguaje de consulta como ser $XPath_{Sub}$ (sección 4.2.1).

Dado que existen dos alternativas para el formato del documento *XML* que contiene el resultado de una consulta, la que corresponde al *XML Schema* que especifica la vista y la que corresponde al *XML Schema* independiente, la aplicación cliente puede buscar el *XML Schema* independiente, para lo cual invoca al método **Descubrir** con el valor DESCUBRIR_SCHEMA_INDEP en el parámetro *Requerimiento* y el valor nulo en el parámetro *Propiedades*:

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 594 SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
    <ns1:Descubrir
       xmlns:ns1="urn:xpath"
       SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
       <Requerimiento
          xsi:type="xsd:string"
          SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
          DESCUBRIR_SCHEMA_INDEP
       </Requerimiento>
       <Propiedades>
          <ListaPropiedad/>
       </Propiedades>
    </ns1:Descubrir>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 33 - Invocación (SOAP) al método Descubrir para obtener el XML Schema independiente

como respuesta se envía el *XML Schema* independiente (Anexo 1) correspondiente a la estructura del documento *XML* que contiene el resultado de la consulta:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 980
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
    <ns1:DescubrirResponse
       xmlns:ns1="urn:xpath"
       SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
       <return>
        <Root>
         <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
          <xs:element name = "Root">
           <xs:complexType contenent="elementOnly">
            <xs:element name = "row" minOccurs="0" maxOccurs ="unbounded">
             <xs:complexType contenent="elementOnly">
              <xs:element name="column" minOccurs="1" maxOccurs="unbounded">
               <xs:complexType>
                <xs:attribute name="nombre" type="xs:string"/>
               </xs:complexType>
              </xs:element>
             </xs:complexType>
            </xs:element>
           </xs:complexType>
          </xs:element>
         </xs:schema>
        </Root>
       </return>
    </ns1:DescubrirResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 34 - Respuesta (SOAP) al método Descubrir para obtener el XML Schema independiente

Una vez que la aplicación cliente dispone del *XML Schema* que especifica la vista y conoce las posibles estructuras del resultado, puede comenzar a realizar consultas para lo cual invoca al método **Ejecutar**.

Por ejemplo si la consulta es saber las ventas realizadas por lo fabricantes, se invoca al método **Ejecutar** con el valor[6] `<Consulta>/fabricante/venta</Consulta>` en el parámetro *Sentencia* y los nombres de la base de datos y de la vista en el parámetro *Propiedades*:

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 595
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
    <ns1:Ejecutar
       xmlns:ns1="urn:xpath"
       SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
```

[6] El elemento `vista` no es considerado para realizar la consulta por las razones que se explican en la sección 4.4.3.

```
        <Sentencia>
           <Consulta>/fabricante/venta</Consulta>
        </Sentencia>
        <Propiedades>
           <ListaPropiedad>
              <BaseDatos>MiBase</BaseDatos>
              <Vista>VistaA</Vista>
           </ListaPropiedad>
        </Propiedades>
     </ns1:Ejecutar>
   </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Figura 35 - Invocación (SOAP) al método Ejecutar para realizar una consulta

como respuesta se envía un documento *XML* con los datos obtenidos de la base de datos (al no haber definido el valor del formato, el resultado es retornado en un documento *XML* con la misma estructura que el *XML Schema* de la vista)

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 2908
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <SOAP-ENV:Body>
  <ns1:EjecutarResponse
   xmlns:ns1="urn:xpath"
   SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
   <return>
    <Root>
     <venta numero_producto=”X3” precio=”100” fecha_venta=”3/1/01”
     fecha_envio=”4/1/01”>
      <producto descripcion=”jabon”/>
     </venta>
     <venta numero_producto=”X2” precio=”200” fecha_venta=”2/2/01”
     fecha_envio=”3/2/01”>
      <producto descripcion=”esponja”/>
     </venta>
     ...
    </Root>
   </return>
  </ns1:EjecutarResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 36 - Respuesta (SOAP) al método Ejecutar con los resultados de la consulta

Ejemplos de otras consultas y como las mismas se escriben en $XPath_{Sub}$ se muestran en el siguiente capítulo.

Capítulo IV Lenguaje de Consulta Basado en XML

4.1 Introducción

El `framework` presentado en el capítulo anterior brinda una solución al problema de realizar consultas a bases de datos relacionales a través de `internet` haciendo transparente el diseño lógico de la misma. Sin embargo, representa una solución general debido a que no define cual es el lenguaje de consulta *XML* a usar, abriendo la posibilidad de que se utilice cualquier lenguaje.
Las implementaciones del `framework`, luego de definir el lenguaje de consulta a usar por las aplicaciones cliente, deben definir la conversión de las consultas a sentencias *SQL*, para que sean resueltas por un motor de bases de datos relacional.

En este capítulo se presenta, formalmente, un lenguaje de consulta *XML*, denominado $XPath_{Sub}$, entre los muchos disponibles, y un algoritmo de conversión a álgebra relacional. La correspondencia entre $XPath_{Sub}$ y álgebra relacional se hace únicamente a los efectos de especificar el algoritmo de conversión a *SQL*.
La generación de expresiones en álgebra relacional tiene claras ventajas frente a la generación directa de *SQL*:

- Permite una clara separación de la especificación del algoritmo de su implementación, dado que el álgebra relacional es un lenguaje relacional abstracto sin elementos de implementación. Esto permite una asociación más clara entre las expresiones de ambos lenguajes ($Xpath_{Sub}$ y álgebra relacional).

- Álgebra relacional es el lenguaje estándar de referencia para el modelo relacional.

- Las expresiones de álgebra relacional pueden descomponerse de manera simple y clara en subexpresiones (lo cual no es tan simple en *SQL*) lo cual facilita el mapeo de expresiones $Xpath_{Sub}$ a subexpresiones de álgebra relacional para la posterior construcción de la expresión completa.

- La especificación es válida para cualquier versión de *SQL*, de cualquier manejador de base de datos, y no para un *SQL* particular.

- Se obtiene un control más fino entre la generación y la optimización. Al utilizar álgebra relacional el énfasis está en la generación de la expresión y no en la optimización, que puede ser realizada a posteriori. En cambio si la especificación se realizara directamente en *SQL* se deberían realizar, tanto la generación como la optimización, en un solo paso[7].

[7] El proceso de optimización no se encuentra dentro del alcance de esta tesis.

4.2 XPath

Como lenguaje a utilizar, para expresar las consultas sobre las vistas *XML* de la base de datos relacional, por parte de las aplicaciones cliente, fue seleccionado un subconjunto del lenguaje *XPath*, versión 1.0 [27].

Si bien *XPath* no es exactamente un lenguaje de consulta sino un lenguaje para navegar y seleccionar partes de un documento *XML*, puede ser usado a los efectos de realizar consultas. La elección de *XPath* obedece a los siguientes motivos:

- *XPath* versión 1.0 es una recomendación oficial del W3C (**W**orld **W**ide **W**eb **C**onsortium).
- La funcionalidades de *XPath* se encuentran presentes en muchos otros lenguajes de consulta de *XML* (sección 2.5).
- Es mandatorio soportar las funcionalidades de *XPath* en el futuro estándar de lenguajes de consulta de *XML* [48].
- Es una componente fundamental de varios estándares de procesamiento de *XML*, como es el caso de *XSLT* [49].
- Las consultas *XPath* son compactas y simples (sección 2.5.1).
- Las consultas *XPath* son fáciles de escribir y de leer.
- Actualmente existen muchas implementaciones de *XPath*, lo cual muestra que es uno de los lenguajes más difundidos y de más amplio uso por parte usuarios y aplicaciones.

4.2.1 *Xpath$_{Sub}$*: subconjunto de XPath

Para expresar las consultas se considera un subconjunto (en cuanto a funcionalidades) del lenguaje *XPath* [27], que de aquí en adelante se denomina *Xpath$_{Sub}$* por conveniencia, el cual consiste en las funcionalidades que se detallan en la siguiente tabla:

Funcionalidades soportadas

Característica	***Xpath$_{Sub}$***
Ejes	`child` `parent`, en su forma abreviada (..) `attribute`, en su forma abreviada (@)
Predicados que evalúan a un valor `booleano` incluyendo predicados sucesivos y anidados	
Operadores de Comparación	=, !=, <, <=, >, >=
Funciones booleanas	`not()`
Operadores booleanos	`and, or`
Paréntesis	`()`

Figura 37 – Funcionalidades de *Xpath$_{Sub}$*

XPath dispone de otras funcionalidades que son más potentes que lo necesario para efectuar consultas a bases de datos relacionales, por ejemplo las funcionalidades relacionadas con el concepto de orden de los elementos presentes en los documentos *XML* el cual no tiene, en general, un equivalente en las bases de datos relacionales. Por este motivo *Xpath$_{Sub}$* incluye las funcionalidades de *XPath* que se consideran suficientes para realizar consultas a bases de datos relacionales.

La siguiente es la gramática de $Xpath_{Sub}$, la cual se basa en la gramática definida en [27]:

```
Camino          ::= '/'[CaminoRelativo]

CaminoRelativo  ::= CaminoComun |
                    CaminoAtributo

CaminoComun     ::= Paso |
                    CaminoComun'/'Paso

CaminoAtributo  ::= CaminoComun'/'Atributo

Paso            ::= {Nombre |'..'} ListaPredicado

ListaPredicado  ::= ListaPredicado'['Predicado']' |
                    ε

Predicado       ::= Pred |
                    'not(' Predicado ')'|
                    Predicado 'and' Pred |
                    Predicado 'or' Pred |
                    '(' Predicado ')'

Pred            ::= CaminoRelativo |
                    Atributo |
                    Operando Operador Operando

Operando        ::= CaminoAtributo |
                    Atributo |
                    Valor
Atributo        ::= '@'Nombre

Operador        ::= '=' |
                    '!=' |
                    '>' |
                    '>=' |
                    '<' |
                    '<='

Nombre          ::= QName
```

donde:

- `Qname` determina los nombres de los elementos y se encuentra definido en [17].
- `Valor` corresponde a la definición de los dominios de los distintos tipos de datos que pueden utilizarse.

4.3 Semántica de *Xpath$_{Sub}$*

En esta sección se define formalmente la semántica de *Xpath$_{Sub}$*, dado que la especificación dada en [27] es descriptiva e informal. La semántica es una adaptación y mejora de [50] que define la semántica de una versión anterior (diciembre de 1998) y simplificada del lenguaje *XPath*.

En primer lugar se presenta un modelo de datos para *Xpath$_{Sub}$* que brinda las herramientas necesarias para definir la semántica y en segundo lugar se realiza la definición de la semántica formal de *Xpath$_{Sub}$* usando semántica denotacional.

4.3.1 Modelo de datos de *Xpath$_{Sub}$*

Dado un documento *XML*, *Xpath$_{Sub}$* (*XPath*) lo representa como un árbol , sobre el cual se resuelven las consultas. Considerando nuevamente el documento del ejemplo de la sección 2.5.1:

```
<?xml version='1.0' encoding='UTF-8' ?>
<personas>
   <persona fnac = “1912” >
    <datos>
     <nombre>Alan</nombre>
     <apellido>Turing</apellido>
    </datos>
    <profesion>matematico</profesion>
    <profesion>criptografo></profesion>
   </persona>
   <persona fnac = “1918” >
    <datos>
     <nombre>Richard</nombre>
     <apellido>Feynman</apellido>
    </datos>
    <profesion>fisico</profesion>
   </persona>
</personas>
```

Figura 38 – Ejemplo de documento XML

el árbol que representa el documento anterior es el siguiente[8]:

[8] Los nodos del árbol han sido numerados por conveniencia para hacer referencia a los mismos.

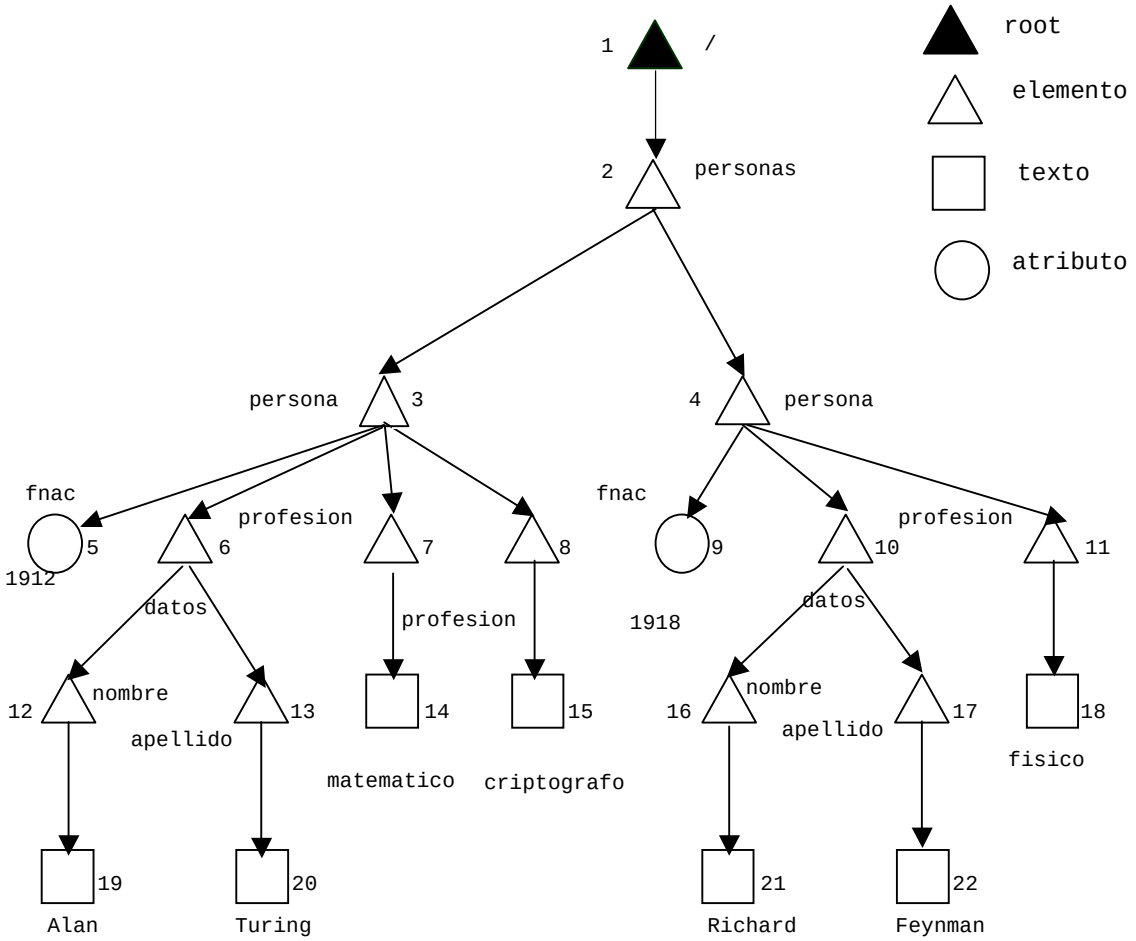


Figura 39 – Árbol de nodos $Xpath_{Sub}$

En la representación en forma de árbol del documento *XML* el tipo básico es `Nodo`. Un ítem del tipo `Nodo`, de aquí en adelante nodo, es de una de las siete clases definidas en el modelo de datos de *XPath* (sección 2.5.1)[9]. En particular existe un nodo distinguido (/) que es la raíz del árbol. Este nodo ficticio es agregado por *XPath* y no coincide con el elemento documento *XML*.

Para indicar la clase de cada nodo se definen funciones `booleanas`, una para cada clase de nodo, tales que para cada nodo solo una de ellas retorna verdadero, por ejemplo:

EsRaiz: Nodo → Boolean

EsElemento: Nodo → Boolean

EsAtrib: Nodo → Boolean

Las siguientes funciones permiten establecer relaciones entre los nodos del árbol[10]:

Raiz: Nodo → Nodo
Dado un nodo del árbol, devuelve el nodo raíz del árbol.

[9] Si se consideran los documentos *XML* pertenecientes a la clase definida por el *XML Schema*, construido según las reglas de la sección 3.5, solamente existen tres clases de nodos: raíz, elemento y atributo.
[10] Estas funciones definen las relaciones estándar en el árbol y son suficientes para definir la semántica de $Xpath_{Sub}$. Eventualmente pueden definirse otras funciones que establezcan otras relaciones entre los nodos, como ser `descendant`, `ancestor`, etc.

Hijos: Nodo → Conjunto(Nodo)
Dado un nodo del árbol, devuelve el conjunto de nodos hijos del nodo o bien el conjunto vacío, si el nodo no tiene hijos. Recordar que en *XPath*, y por lo tanto en $Xpath_{Sub}$, los atributos de un nodo no se consideran hijos del nodo.

Padre: Nodo → Conjunto(Nodo)
Dado un nodo del árbol, devuelve el nodo padre (conjunto con un solo elemento) o bien el conjunto vacío, si el nodo es la raíz del árbol.

Atrib: Nodo → Conjunto(Nodo)
Dado un nodo del árbol, devuelve el conjunto de nodos atributos del nodo o bien el conjunto vacío, si el nodo no tiene atributos.

Por ejemplo, en el árbol del ejemplo anterior[11]:

Raiz(2) = /
Raiz(3) = /
Raiz(7) = /

Hijos(3) = {6,7,8}
Hijos(12) = {19}
Hijos(19) = ∅

Padre(2) = {/}
Padre(/) = ∅
Padre(16) = {10}

Atrib(3) = {5}
Atrib(4) = {9}
Atrib(8) = ∅

Usando las funciones definidas, es posible formalizar las afirmaciones realizadas en la sección 2.5.1. A modo de ejemplo:

1. Solo el nodo raíz (/) o un nodo elemento pueden tener hijos:
   Hijos(x) ≠ ∅ → EsRaiz(x) ∨ EsElemento(x)

2. Solo los nodos elementos pueden tener atributos:
   Atrib(x) ≠ ∅ → EsElemento(x)

3. Un nodo y es hijo (atributo) de un nodo x si y solo si el padre del nodo y es el nodo x:
   y ∈ Hijos(x) ⇔ Padre(y) = {x}
   y ∈ Atrib(x) ⇔ Padre(y) = {x}

Para definir la semántica son necesarias funciones auxiliares[12] sobre el contenido de los nodos:

Nombre: Nodo → String
La función *Nombre* se define según la clase del nodo de acuerdo a la siguiente tabla:

Dominio	**Recorrido**
raíz (/)	NULL
elemento	etiqueta XML
atributo	nombre del atributo

[11] Se utilizan los números de los nodos para hacer referencia a los mismos.
[12] De manera análoga estas funciones auxiliares pueden ser extendidas para considerar otras clases de nodos.

Val: Nodo → Valor
La función *Val* se define según la clase del nodo de acuerdo a la siguiente tabla:

Dominio	**Recorrido**
raíz (*/*)	V_1
elemento	V_1
atributo	V_2

donde, de acuerdo a lo indicado en [27]:

> V_1 es el resultado de concatenar los valores de los nodos de clase texto descendientes del nodo.
> V_2 es el valor del atributo.

4.3.2 Semántica

La definición de la semántica formal de $Xpath_{Sub}$ se realiza utilizando semántica denotacional, la cual tiene como punto de partida la sintaxis abstracta de un lenguaje.

La gramática presentada en la sección 4.2.1 define la sintaxis concreta de $Xpath_{Sub}$. La sintaxis concreta trata a un lenguaje como un conjunto de `strings` sobre un alfabeto de símbolos y es especificada mediante una gramática que define un conjunto de producciones para generar `strings` de símbolos, usando símbolos auxiliares (denominados no terminales y terminales) [51].

En general, para evitar la ambigüedad, las gramáticas que especifican la sintaxis concreta de un lenguaje tienden a ser complejas y requieren del uso de símbolos y producciones que no tienen relevancia a los efectos de la estructura esencial del lenguaje. Un ejemplo de lo anterior son los símbolos y producciones necesarios para establecer la precedencia de los operadores o para definir agrupaciones.
La sintaxis abstracta es más simple que la sintaxis concreta porque identifica la estructura (semánticamente) relevante del lenguaje, sin la complejidad de la sintaxis concreta.

La siguiente es la sintaxis abstracta correspondiente a la gramática del lenguaje $Xpath_{Sub}$:

```
n:Qname

v:Valor

c:Camino
c  ::= /cc | /ca | /

cc:CaminoComun
cc ::= cc/n | cc/.. | n | .. | cc p

ca:CaminoAtributo
ca ::= cc/@n

p:Predicado
p ::= not p | p1 and p2 | p1 or p2 | @n | cc | ca | o1 op o2

o:Operando
o ::= ca | @n | v

op ::= '=' | '!=' |'>' /|'>=' |'<' | '<='
```

La semántica de $Xpath_{Sub}$ se muestra en la Figura 40 considerando que el árbol de nodos es global a todas las funciones de la semántica y usando conjuntos por comprensión.

La definición de la semántica se realiza mediante una función $\boldsymbol{X}$ que dado un camino retorna un conjunto de nodos del árbol. $\boldsymbol{X}[c]$ es el conjunto de nodos seleccionado por el camino absoluto *c*. Esta función utiliza las funciones $\boldsymbol{X}_{cc}$ y $\boldsymbol{X}_{ca}$ dependiendo del tipo de camino (común o camino que finaliza en un atributo). Como el camino *c* es absoluto ambas funciones son invocadas con el nodo raíz (/) como nodo contexto en el árbol de nodos.

Para definir la semántica se utilizan, asimismo, las funciones $\boldsymbol{P}$ y $\boldsymbol{O}$. La función $\boldsymbol{P}$ retorna un valor booleano, dados un predicado y un nodo contexto. $\boldsymbol{P}[p](x)$ devuelve verdadero si el predicado *p* es satisfecho cuando el nodo contexto es x.

La función $\boldsymbol{O}$ es usada para evaluar si un nodo contexto satisface un caso particular de predicado en el cual intervienen operadores de comparación. En este caso la evaluación del predicado se realiza de la siguiente manera (sin considerar las conversiones de tipo que se realizan en [27]):

- si ambos operandos determinan un conjunto de nodos, el predicado es verdadero si existe un nodo, en el primer conjunto de nodos, y existe un nodo, en el segundo conjunto de nodos, tal que la comparación de los valores de ambos nodos devuelve verdadero.
- si uno de los operandos determina un conjunto de nodos y el otro operando es un valor, el predicado es verdadero si existe un nodo, en el conjunto de nodos, tal que la comparación del valor del nodo y el valor devuelve verdadero.
- si ninguno de los operandos determina un conjunto de nodos, o sea ambos operandos son valores, el predicado es verdadero si la comparación de los valores devuelve verdadero.

A continuación se explican algunas de las reglas que definen la semántica:

- La regla $\boldsymbol{X}[/]$ indica que el camino / selecciona el conjunto de nodos formado por la raíz del árbol.

- La regla $\boldsymbol{X}[/cc]$ ($\boldsymbol{X}[/ca]$) selecciona los nodos, seleccionados por el camino *cc* (*ca*) siendo el nodo contexto la raíz del árbol.

- La regla $\boldsymbol{X}_{cc}[..](x)$ indica que el camino .. , siendo x el nodo contexto, selecciona el conjunto de nodos formado por el padre del nodo contexto.

- La regla $\boldsymbol{X}_{cc}[n](x)$ indica que el camino *n* , siendo x el nodo contexto, selecciona el conjunto de nodos formado por los nodos hijos del nodo x, cuyo nombre es *n*.

- La regla $\boldsymbol{X}_{cc}[cc\ p](x)$ resuelve primero el camino *cc*, con el nodo contexto x, obteniendo un conjunto de nodos seleccionados y de estos selecciona únicamente aquellos que satisfacen el predicado *p*.

- Un camino puede ser usado a su vez como un predicado. En este caso (reglas $\boldsymbol{P}[cc](x)$, $\boldsymbol{P}[ca](x)$ y $\boldsymbol{P}[@n](x)$) el predicado es satisfecho únicamente si el camino selecciona un conjunto de nodos no vacío al ser usado como predicado, considerando al nodo x como contexto.

Notar que por las características de la gramática de la sección 4.2.1, y por lo tanto de la sintaxis abstracta, la función Val (en la regla $\boldsymbol{P}[o_1\ op\ o_2](x)$) solo se aplica a nodos atributo o a valores de un dominio (tipo de datos), por lo tanto se redefine de la siguiente manera:

Val: Nodo $\cup$ Valor $\rightarrow$ Valor

Dominio	**Recorrido**
Atributo	V_2
Valor	Valor

donde:

- V_2 es el valor del atributo.

$\boldsymbol{X}$ **: Camino → Conjunto(Nodo)**[13]

$\boldsymbol{X}[/cc] = \boldsymbol{X_{cc}}[cc](/)$
$\boldsymbol{X}[/ca] = \boldsymbol{X_{ca}}[ca](/)$
$\boldsymbol{X}[/] = /$[14]

$\boldsymbol{X_{cc}}$ **: CaminoComun × Nodo → Conjunto(Nodo)**

$\boldsymbol{X_{cc}}[cc/n](x) = \{x_2 / \exists x_1.(x_1 \in \boldsymbol{X_{cc}}[cc](x) \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = n)\}$
$\boldsymbol{X_{cc}}[cc/..](x) = \{x_2 / \exists x_1.(x_1 \in \boldsymbol{X_{cc}}[cc](x) \wedge x_2 \in Padre(x_1))\}$
$\boldsymbol{X_{cc}}[n](x) = \{x_1 / x_1 \in Hijos(x) \wedge Nombre(x_1) = n\}$
$\boldsymbol{X_{cc}}[..](x) = Padre(x)$
$\boldsymbol{X_{cc}}[cc\ p](x) = \{x_1 / x_1 \in \boldsymbol{X_{cc}}[cc](x) \wedge \boldsymbol{P}[p](x_1)\}$

$\boldsymbol{X_{ca}}$ **: CaminoAtributo × Nodo → Conjunto(Nodo)**

$\boldsymbol{X_{ca}}[cc/@n](x) = \{x_2 / \exists x_1.(x_1 \in \boldsymbol{X_{cc}}[cc](x) \wedge x_2 \in Atrib(x_1) \wedge Nombre(x_2) = n)\}$

$\boldsymbol{P}$ **: Predicado × Nodo → Boolean**

$\boldsymbol{P}[not\ p](x) = \neg\ \boldsymbol{P}[p]x$
$\boldsymbol{P}[@n](x) = \{x_1 / x_1 \in Atrib(x) \wedge Nombre(x_1) = n\} \# \emptyset$
$\boldsymbol{P}[cc](x) = \boldsymbol{X_{cc}}[cc](x) \# \emptyset$
$\boldsymbol{P}[ca](x) = \boldsymbol{X_{ca}}[ca](x) \# \emptyset$
$\boldsymbol{P}[p_1\ and\ p_2](x) = \boldsymbol{P}[p_1](x) \wedge \boldsymbol{P}[p_2](x)$
$\boldsymbol{P}[p_1\ or\ p_2](x) = \boldsymbol{P}[p_1](x) \vee \boldsymbol{P}[p_2](x)$
$\boldsymbol{P}[o_1\ op\ o_2](x) = \exists x_1, \exists x_2.(x_1 \in \boldsymbol{O}[o_1](x) \wedge x_2 \in \boldsymbol{O}[o_2](x) \wedge Val(x_1)\ \text{op}\ Val(x_2))$

$\boldsymbol{O}$ **: Operando × Nodo → Conjunto(Nodo) ∪ Conjunto(Valor)**

$\boldsymbol{O}[ca](x) = \boldsymbol{X_{ca}}[ca](x)$
$\boldsymbol{O}[@n](x) = \{x_1 / x_1 \in \in Atrib(x) \wedge Nombre(x_1) = n\}$
$\boldsymbol{O}[v](x) = \{v\}$

Figura 40 – Semántica de $Xpath_{Sub}$

[13] Referirse a la sección 2.5.1 por más detalles.
[14] El símbolo / que aparece a la derecha de la igualdad denota el nodo raíz y no un camino.

Por ejemplo, considerando el *XML Schema* definido en la sección 3.5:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name = "Vista">
    <xs:complexType contenent="elementOnly">
      <xs:element name="fabricante" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType contenent="elementOnly">
          <xs:element name="venta" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType contenent="elementOnly">
              <xs:element name="producto" minOccurs="1" maxOccurs="1">
                <xs:complexType contenent="elementOnly">
                  <xs:attribute name="descripcion" type ="xs:string"/>
                </xs:complexType>
              </xs:element>
              <xs:attribute name="numero_producto" type= "xs:string"/>
              <xs:attribute name="precio" type= "xs:decimal"/>
              <xs:attribute name="fecha_venta" type= "xs:date"/>
              <xs:attribute name="fecha_envio" type= "xs:date"/>
            </xs:complexType>
          </xs:element>
          <xs:attribute name="numero_fabricante" type= "xs:positiveinteger"/>
          <xs:attribute name="direccion" type= "xs:string"/>
          <xs:attribute name="nombre" type= "xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 41 – XML Schema de una vista XML

un documento *XML* que corresponde al *XML Schema* anterior es el siguiente:

```
<?xml version='1.0' encoding='UTF-8' ?>
<vista>
  <fabricante numero_fabricante="1" direccion="Salto 1025" nombre="Juan Diaz">
    <venta numero_producto="X1" precio="300" fecha_venta="2/1/01" fecha_envio="5/1/01">
       <producto  descripcion="Toalla"></prod>
    </venta>
    <venta numero_producto="X2" precio="200" fecha_venta="3/1/01" fecha_envio="4/1/01">
      <producto  descripcion="Esponja"></prod>
    </venta>
  </fabricante>
  <fabricante numero_fabricante="2"  direccion="Chana 1122" nombre="Jorge Rodriguez">
    <venta numero_producto="X2" precio="100" fecha_venta="4/1/01" fecha_envio="6/1/01">
      <producto  descripcion="Esponja"></prod>
    </venta>
 </fabricante>
 <fabricante numero_fabricante="3"  direccion="Rodo 2139" nombre="Luis Perez" >
  <venta numero_producto="X3" precio="100" fecha_venta="2/2/01" fecha_envio="3/2/01">
     <producto  descripcion="Jabon"></prod>
    </venta>
 </fabricante>
  </vista>
```

Figura 42 – Documento XML

el árbol que representa el documento *XML* es el que se muestra a continuación (donde los nodos han sido numerados por conveniencia). El mismo se utilizará para mostrar como se resuelven determinadas consultas, usando la definición de la semántica de $Xpath_{Sub}$:

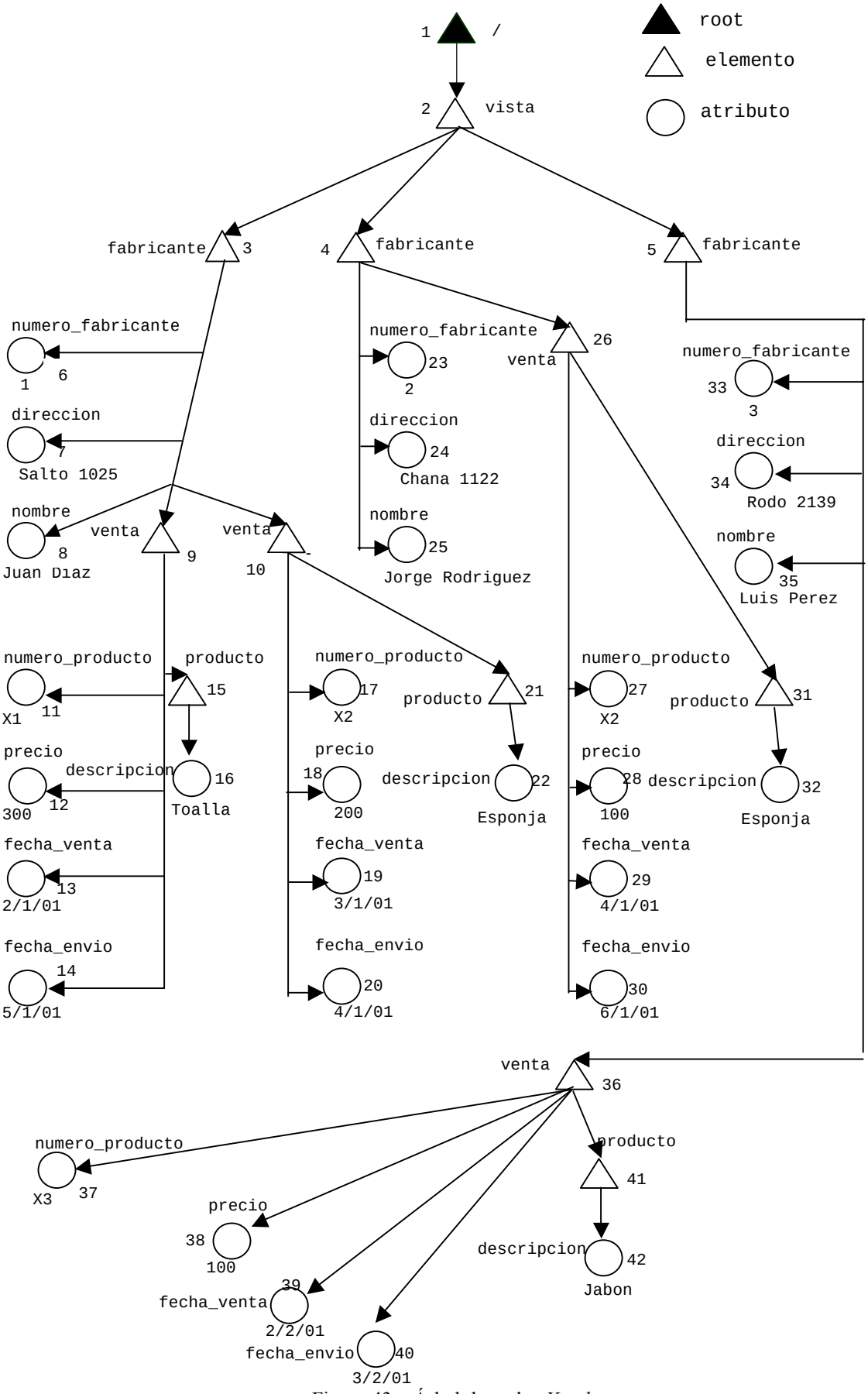


Figura 43 – Árbol de nodos $Xpath_{Sub}$

Consulta 1

Devolver los números de producto que fueron vendidos a un precio mayor que 200 pesos.

Esta consulta se expresa en $Xpath_{Sub}$ como:

```
/vista/fabricante/venta[@precio > 200]/@numero_producto
```

$\boldsymbol{X}$[*/vista/fabricante/venta[@precio > 200]/@numero_producto*] =

$\boldsymbol{X}_{ca}$[*vista/fabricante/venta[@precio > 200]/@numero_producto*](/) =
$\{x_2 / \exists x_1.(x_1 \in \boldsymbol{X}_{cc}[vista/fabricante/venta\ @precio > 200](/) \wedge x_2 \in Atrib(x_1) \wedge Nombre(x_2) = numero\_producto)\}$

$\boldsymbol{X}_{cc}$[*vista/fabricante/venta @precio > 200*](/) =
$\{x_1 / x_1 \in \boldsymbol{X}_{cc}[vista/fabricante/venta](/) \wedge \boldsymbol{P}[@precio > 200](x_1)\}$

$\boldsymbol{X}_{cc}$[*vista/fabricante/venta*](/) =
$\{x_2 / \exists x_1.(x_1 \in \boldsymbol{X}_{cc}[vista/fabricante](/) \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = venta)\}$

$\boldsymbol{X}_{cc}$[*vista/fabricante*](/) =
$\{x_2 / \exists x_1.(x_1 \in \boldsymbol{X}_{cc}[vista](/) \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = fabricante)\}$

$\boldsymbol{X}_{cc}[vista](/) = \{x_1 / x_1 \in Hijos(/) \wedge Nombre(x_1) = \text{vista}\} = \{2\}$

$\Rightarrow \boldsymbol{X}_{cc}[vista/fabricante](/) = \{3,4,5\}$

$\Rightarrow \boldsymbol{X}_{cc}[vista/fabricante/venta](/) = \{9,10,26,36\}$

$\Rightarrow \boldsymbol{X}_{cc}[vista/fabricante/venta\ @precio > 200](/) =$
$\{x_1 / x_1 \in \{9,10,26,36\} \wedge \boldsymbol{P}[@precio > 200](x_1)\} =$

Para $x_1 = 9$
$\boldsymbol{P}[@precio > 200](9) = \exists x_1, \exists x_2.(x_1 \in \boldsymbol{O}[@precio](9) \wedge x_2 \in \boldsymbol{O}[200](9) \wedge Val(x_1) > Val(x_2))$

$\boldsymbol{O}[@precio](9) = \{x_1 / x_1 \in Atrib(9) \wedge Nombre(x_1) = precio\} = \{12\}$

$\boldsymbol{O}[200](9) = \{200\}$

$\Rightarrow \boldsymbol{P}[@precio > 200](9) = \exists x_1, \exists x_2.(x_1 \in \{12\} \wedge x_2 \in \{200\} \wedge Val(x_1) > Val(x_2))$ el nodo 9 cumple el predicado

$x_1 = 10$, $x_1 = 26$ y $x_1 = 36$ no cumplen el predicado

$\Rightarrow \boldsymbol{X}_{cc}[vista/fabricante/venta\ @precio > 200](/) = \{9\}$

$\Rightarrow \boldsymbol{X}_{ca}[vista/fabricante/venta[@precio > 200]/@numero\_producto](/) =$
$\{x_2 / \exists x_1.(x_1 \in \{9\} \wedge x_2 \in Atrib(x_1) \wedge Nombre(x_2) = numero\_producto)\} = \{11\}$

$\Rightarrow \boldsymbol{X}[/vista/fabricante/venta[@precio > 200]/@numero\_producto] = \{11\}$

La consulta retorna como resultado el nodo 11 del árbol, o lo que es lo mismo en *XML*:

```
numero_producto="X1"
```

El resultado de esta consulta es un nodo de clase atributo. Por lo tanto el resultado que se obtiene sobre el documento *XML* es el atributo *XML* correspondiente.

Consulta 2

Devolver las ventas del fabricante número 1.

Esta consulta se expresa en $Xpath_{Sub}$ como:

```
/vista/fabricante[@numero_fabricante = 1]/venta
```

$\boldsymbol{X}$[*/vista/fabricante[@numero_fabricante = 1]/venta*] =

$\boldsymbol{X}_{cc}$[*vista/fabricante[@numero_fabricante = 1]/venta*](/) =
$\{x_2 / \exists x_1.(x_1 \in \boldsymbol{X}_{cc}[vista/fabricante\ @numero\_fabricante = 1](/) \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = venta)\}$

$\boldsymbol{X}_{cc}$[*vista/fabricante @numero_fabricante = 1*](/) =
$\{x_1 / x_1 \in \boldsymbol{X}_{cc}[vista/fabricante](/) \wedge \boldsymbol{P}[@numero\_fabricante = 1](x_1)\}$

$\boldsymbol{X}_{cc}$[*vista/fabricante*](/) =
$\{x_2 / \exists x_1.(x_1 \in \boldsymbol{X}_{cc}[vista](/) \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = fabricante)\}$

$\boldsymbol{X}_{cc}$[*vista*](/) = $\{x_1 / x_1 \in Hijos(/) \wedge Nombre(x_1) = \text{vista}\} = \{2\}$

$\Rightarrow \boldsymbol{X}_{cc}$[*vista/fabricante*](/) = {3,4,5}

$\Rightarrow \boldsymbol{X}_{cc}$[*vista/fabricante[@numero_fabricante = 1]*](/) =
$\{x_1 / x_1 \in \{3,4,5\} \wedge \boldsymbol{P}[@numero\_fabricante = 1](x_1)\}$

Para $x_1 = 3$

$\boldsymbol{P}$[*@numero_fabricante = 1*](3) =
$\exists x_1, \exists x_2.(x_1 \in \boldsymbol{O}[@numero\_fabricante](3) \wedge x_2 \in \boldsymbol{O}[1](3) \wedge Val(x_1) > Val(x_2))$

$\boldsymbol{O}$[*@numero_fabricante*](3) =
$\{x_1 / x_1 \in \in Atrib(3) \wedge Nombre(x_1) = numero\_fabricante\} = \{6\}$

$\boldsymbol{O}$[1](3) = {1}

$\Rightarrow \boldsymbol{P}[@numero\_fabricante = 1](3) = \exists x_1, \exists x_2.(x_1 \in \{6\} \wedge x_2 \in \{1\} \wedge Val(x_1) > Val(x_2))$ el nodo 3 cumple el predicado

$x_1 = 4$ y $x_1 = 5$ no cumplen el predicado

$\Rightarrow \boldsymbol{X}_{cc}$[*vista/fabricante @numero_fabricante = 1*](/) = {3}

$\Rightarrow \boldsymbol{X}_{cc}$[*vista/fabricante[@numero_fabricante = 1]/venta*](/) =
$\{x_2 / \exists x_1.(x_1 \in \{3\} \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = venta)\} = \{9,10\}$

$\Rightarrow \boldsymbol{X}$[*/vista/fabricante[@numero_fabricante = 1]/venta*] = {9,10}

La consulta retorna como resultado los nodos 9 y 10 del árbol, o lo que es lo mismo en *XML*:

```
<venta numero_producto="X1" precio="300" fecha_venta="2/1/01" fecha_envio="5/1/01">
   <producto  descripcion="Toalla"></prod>
</venta>
<venta numero_producto="X2" precio="200" fecha_venta="3/1/01" fecha_envio="4/1/01">
  <producto  descripcion="Esponja"></prod>
</venta>
```

El resultado de esta consulta son nodos de la clase elemento. Por lo tanto el resultado que se obtiene sobre el documento *XML* son los elementos *XML* correspondientes (y, por lo tanto, sus subelementos y atributos).

En términos del documento *XML* representado por el árbol sobre el cual se resuelve la consulta, los resultados anteriores son equivalentes a considerar el conjunto de nodos que retorna la consulta como raíces de subárboles. Dicho de otra manera, cada nodo retornado por la consulta es la raíz de un subárbol y cada subárbol determina la parte del documento *XML* que retorna la consulta.

En el caso de la consultas anteriores, la primer consulta retorna el nodo número 11, como se observa en la figura 44.

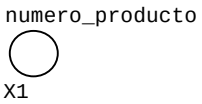


Figura 44 – Nodo del árbol

Este nodo corresponde a la clase atributo (*EsAtrib*(11) = true) y por lo tanto es una hoja del árbol, dado que los nodos atributo no tienen hijos:
Atrib(x) ≠ ∅ → EsElemento(x)

La segunda consulta retorna los nodos números 9 y 10, como se observa en la figura 45.

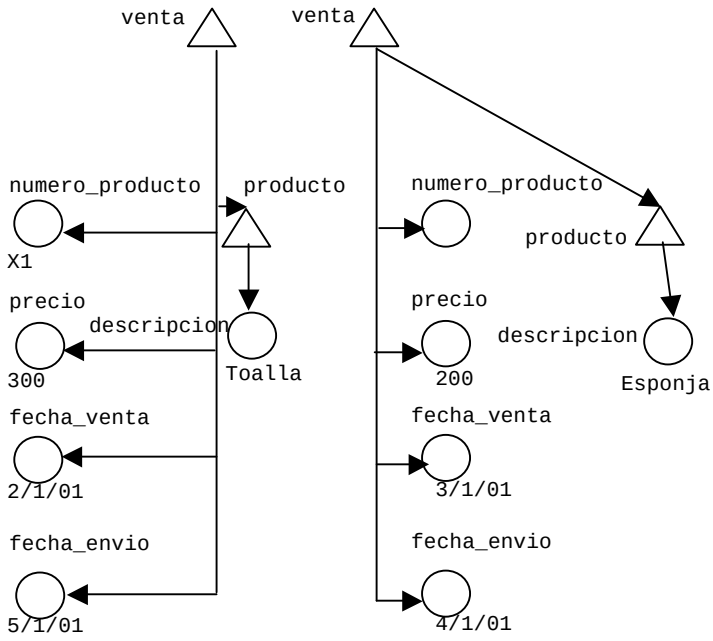


Figura 45 – Otros nodos del árbol

Estos nodos corresponden a la clase elemento (*EsElemento*(9) = true, *EsElemento*(10) = true) y por lo tanto pueden tener subelementos (descendientes) y atributos.

4.4 Conversión a álgebra relacional

En las siguientes secciones se detallan los operadores que componen el álgebra relacional, se muestran ejemplos de las expresiones algebraicas que genera el algoritmo de conversión y, por último, se describe brevemente el algoritmo de conversión, cuya especificación se presenta en la sección 4.5.

4.4.1 Álgebra Relacional

Para la conversión de consultas *Xpath*$_{Sub}$ a álgebra relacional se consideran los siguientes operadores relacionales:

- Selección
  $\sigma_c(E)$ selecciona las tuplas de una relación E que satisfacen una condición c booleana.
  $\sigma_c(E) = \{t / t \in E \wedge c(t)\}$
- Proyección
  $\Pi_{atrs}(E)$ retorna una relación que es el resultado de particionar verticalmente, por las columnas indicadas en atrs, la relación E.
- Producto Cartesiano
  E_1 x E_2 combina dos relaciones E_1 y E_2.
  $E_1 \times E_2 = \{t / \exists t_1, \exists t_2.(t_1=(v_1..v_j) \in E_1 \wedge t_2=(v_{j+1},..v_n) \in E_2 \wedge t = (v_1..v_n))\}$
- Join Natural
  E_1 * E_2 combina dos relaciones E_1 y E_2, aplicando la condición de igualdad entre las columnas del mismo nombre y eliminando las columnas repetidas.
  $E_1 * E_2 = \Pi_{e1 \cup e2}(\sigma_{\forall a \in e1 \cap e2.(E1.a = E2.a)}(E_1 \times E_2))$ siendo e1 y e2 las columnas de E_1 y E_2 respectivamente.
- Renombre
  $\delta_{(NV \rightarrow NN)}(E)$ renombra el conjunto de columnas NV, de una relación E, por el conjunto de columnas NN.
- Diferencia
  E_1 - E_2 devuelve una relación con las tuplas que están en la relación E_1 pero que no están en la relación E_2. Ambas relaciones deben tener el mismo esquema.
  $E_1 - E_2 = \{t / t \in E_1 \wedge t \notin E_2\}$
- Intersección
  $E_1 \cap E_2$ devuelve una relación con las tuplas que están en la relación E_1 y en la relación E_2. Ambas relaciones deben tener el mismo esquema.
  $E_1 \cap E_2 = \{t / t \in E_1 \wedge t \in E_2\} = E_1 - (E_1 - E_2)$
- Unión
  $E_1 \cup E_2$ devuelve una relación con las tuplas que están en la relación E_1 ó en la relación E_2. Ambas relaciones deben tener el mismo esquema.
  $E_1 \cup E_2 = \{t / t \in E_1 \vee t \in E_2\}$
- Left Outer join
  En un join natural las tuplas de una relación que no machean con tuplas de la otra relación no aparecen en el resultado. El outer join retiene estas tuplas y coloca valores nulos en las restantes columnas.
  $E_1 \, {}_{Left}* E_2 = E_1 * E_2 \cup ((E_1 - \Pi_{e1}(E_1 * E_2)) \times E'_2)$ siendo e1 las columnas de E_1 y E'_2 una relación que tiene como columnas las columnas de E_2 que no están en E_1, conteniendo una única tupla formada por valores null en todas sus columnas.

4.4.2 Ejemplos de conversión

Previamente a definir el algoritmo de conversión de $Xpath_{Sub}$ a álgebra relacional se mostrarán ejemplos[15] de las expresiones algebraicas que se generan, para lo cual se usará la base de datos y el *XML Schema* del ejemplo de la sección 3.5:

Fabricantes *(numero_fabricante, direccion, nombre)*
Productos *(numero_producto, descripción)*
Ventas *(numero_fabricante, numero_producto, precio, fecha_venta, fecha_envio)*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name = "Vista">
  <xs:complexType contenent="elementOnly">
   <xs:element name="fabricante" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType contenent="elementOnly">
     <xs:element name="venta" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType contenent="elementOnly">
       <xs:element name="producto" minOccurs="1" maxOccurs="1">
        <xs:complexType contenent="elementOnly">
         <xs:attribute name="descripcion" type ="xs:string"/>
        </xs:complexType>
       </xs:element>
       <xs:attribute name="numero_producto" type= "xs:string"/>
       <xs:attribute name="precio" type= "xs:decimal"/>
       <xs:attribute name="fecha_venta" type= "xs:date"/>
       <xs:attribute name="fecha_envio" type= "xs:date"/>
      </xs:complexType>
     </xs:element>
     <xs:attribute name="numero_fabricante" type= "xs:positiveinteger"/>
     <xs:attribute name="direccion" type= "xs:string"/>
     <xs:attribute name="nombre" type= "xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 46 – XML Schema de una vista XML

A los efectos de que los ejemplos resulten claros se supondrá que la base de datos contiene los siguientes datos:

Fabricantes

numero_fabricante	*direccion*	*nombre*
1	Salto 1025	Juan Diaz
2	Chana 1122	Jorge Rodriguez
3	Rodo 2139	Luis Perez

Productos

numero_producto	*descripcion*
X1	Toalla
X2	Esponja
X3	Jabon

Ventas

numero_fabricante	*numero_producto*	*precio*	*fecha_venta*	*fecha_envio*
1	X1	300	2/1/01	5/1/01
1	X2	200	3/1/01	4/1/01
2	X2	100	4/1/01	6/1/01
3	X3	100	2/2/01	3/2/01

Figura 47 – Base de datos relacional

[15] En todos los ejemplos se aplica un post procesamiento de la expresión algebraica generada por el algoritmo que realiza una optimización muy simple de la misma. Este post procesamiento se explica en la sección 4.7. Asimismo se aplica una mejora al algoritmo, la cual se explica en la sección 4.6.1.

un posible documento *XML* que corresponde al *XML Schema* anterior y que contiene los datos de la base de datos es el siguiente, ya mostrado en la sección 4.3:

```
<?xml version='1.0' encoding='UTF-8' ?>
<vista>
 <fabricante numero_fabricante="1" direccion="Salto 1025" nombre="Juan Diaz">
  <venta numero_producto="X1" precio="300" fecha_venta="2/1/01" fecha_envio="5/1/01">
     <producto  descripcion="Toalla"></prod>
  </venta>
  <venta numero_producto="X2" precio="200" fecha_venta="3/1/01" fecha_envio="4/1/01">
    <producto  descripcion="Esponja"></prod>
  </venta>
 </fabricante>
 <fabricante numero_fabricante="2"  direccion="Chana 1122" nombre="Jorge Rodriguez">
  <venta numero_producto="X2" precio="100" fecha_venta="4/1/01" fecha_envio="6/1/01">
    <producto  descripcion="Esponja"></prod>
  </venta>
 </fabricante>
 <fabricante numero_fabricante="3"  direccion="Rodo 2139" nombre="Luis Perez" >
  <venta numero_producto="X3" precio="100" fecha_venta="2/2/01" fecha_envio="3/2/01">
     <producto  descripcion="Jabon"></prod>
  </venta>
 </fabricante>
 </vista>
```

Figura 48 – Documento XML correspondiente a una vista

en la sección 4.3 se muestra también el árbol de nodos que representa el documento *XML* anterior.
Se debe recordar que las aplicaciones cliente solamente disponen del *XML Schema* que especifica la vista *XML* y que el documento *XML* mostrado no existe en realidad , dado que los datos se encuentran en la base de datos.

Ejemplo 1:
Una consulta sobre el *XML Schema* anterior podría ser: "devolver los números de producto que fueron vendidos a un precio mayor que 200 pesos". Esta consulta se expresa en $Xpath_{Sub}$ como:

```
/fabricante/venta[@precio > 200]/@numero_producto
```

Esta consulta implica retornar los valores de los atributos `numero_producto` de los elementos `venta`, hijos de un elemento `fabricante`, tales que el valor del atributo `precio` es mayor que 200. El algoritmo convierte la consulta $Xpath_{Sub}$ a la siguiente expresión de álgebra relacional:

$$\Pi_{\texttt{numero\_producto}}(\texttt{Fabricantes} * \texttt{Ventas} * \sigma_{\texttt{precio} > 200}(\texttt{Fabricantes} * \texttt{Ventas}))$$

Ejemplo 2:

La consulta: "devolver las ventas del fabricante número 1", se expresa en $Xpath_{Sub}$ como:

```
/fabricante[@numero_fabricante = 1]/venta
```

Esta consulta implica retornar los valores de los elementos `venta`, hijos de un elemento `fabricante` tal que el valor del atributo `numero_fabricante` es 1.
En este caso, el último paso de la consulta $Xpath_{Sub}$ un elemento, por lo cual la expresión algebraica retornada por el algoritmo es una proyección por las columnas que corresponden a los atributos del elemento `venta` y las columnas que corresponden a los atributos de los elementos descendientes, de manera que se disponga de toda la información relativa del elemento y de los subelementos que el mismo pueda tener, a los efectos de construir un documento *XML* con los datos a retornar, ya sea conforme al *XML Schema* de la vista *XML* de la base de datos o bien al *XML Schema* independiente (sección 8.3):

$$\Pi_{\texttt{numero\_producto,precio,fecha\_venta,fecha\_envio,descipcion}}(\texttt{Fabricantes} * \sigma_{\texttt{numero\_fabricante} = 1}(\texttt{Fabricantes}) * \texttt{Ventas}\ {}_{\texttt{Left}} * \texttt{Productos})$$

Ejemplo 3:
La consulta: "devolver los números de fabricantes que hayan realizado ventas", se expresa en *Xpath*$_{Sub}$ como:

```
/fabricante[venta]/@numero_fabricante
```

El predicado de la consulta es un camino que evalúa a verdadero si determina un conjunto de nodos no vacío. Dicho de otra manera la consulta retorna los números de fabricante de aquellos fabricantes que tienen por lo menos un hijo `venta`.
La expresión en álgebra relacional que genera el algoritmo es la siguiente:

$$\Pi_{\texttt{numero\_fabricante}}(\texttt{Fabricantes} * \rho_{\texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion} \rightarrow \texttt{numero\_fabricante,nombre,direccion}}(\Pi_{\texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\Pi_{\texttt{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\sigma_{\texttt{ctx\_numero\_fabricante=numero\_fabricante} \wedge \texttt{ctx\_nombre=nombre} \wedge \texttt{ctx\_direccion = direccion}}(\rho_{\texttt{numero\_fabricante,nombre,direccion} \rightarrow \texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\texttt{Fabricantes}) \texttt{ x Fabricantes * Ventas}))))))$$

Otra alternativa es escribir la consulta *Xpath*$_{Sub}$ como:

```
/fabricante/venta/../@numero_fabricante
```

La expresión en álgebra relacional que genera el algoritmo, en este caso, es la siguiente:

$$\Pi_{\texttt{numero\_fabricante}}(\Pi_{\texttt{numero\_fabricante,nombre,direccion}}(\texttt{Fabricantes} * \texttt{Ventas}))$$

Ejemplo 4:
La consulta: "devolver los nombres de fabricantes que realizaron ventas tales que la fecha de venta de alguna de sus ventas es igual a la fecha de envío de algunas de sus ventas", se expresa en *Xpath*$_{Sub}$ como:

```
/fabricante[venta/@fecha_venta=venta/@fecha_envio]/@nombre
```

la consulta *Xpath*$_{Sub}$ implica evaluar para cada fabricante el predicado:

```
[venta/@fecha_venta = venta/@fecha_envio]
```

esta expresión `booleana` es verdadera sólo si el valor del atributo `fecha_venta` de algún elemento `venta`, hijo de un elemento `fabricante`, es igual al valor del atributo `fecha_envio` de algún elemento `venta`, hijo del mismo elemento `fabricante`, de acuerdo a la semántica de *Xpath*$_{Sub}$.
La expresión de álgebra relacional que genera el algoritmo es la siguiente:

$$\Pi_{\texttt{nombre}}(\texttt{Fabricantes} * \rho_{\texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion} \rightarrow \texttt{numero\_fabricante,nombre,dirccion}}(\Pi_{\texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\sigma_{\texttt{ctx\_numero\_fabricante=ctx\_numero\_fabricante} \wedge \texttt{ctx\_nombre=ctx\_nombre} \wedge \texttt{ctx\_direccion=ctx\_direccion} \wedge \texttt{fecha\_venta=fecha\_envio}}(R_1 \texttt{ x } R_2))))$$

siendo R_1:

$$\Pi_{\texttt{fecha\_venta,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\Pi_{\texttt{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\sigma_{\texttt{ctx\_numero\_fabricante=numero\_fabricante} \wedge \texttt{ctx\_mombre=nombre} \wedge \texttt{ctx\_direccion=direccion}}(\rho_{\texttt{numero\_fabricante,nombre,direccion} \rightarrow \texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\texttt{Fabricantes}) \texttt{ x Fabricantes * Ventas})))$$

y siendo R_2:

$$\Pi_{\texttt{fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\Pi_{\texttt{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\sigma_{\texttt{ctx\_numero\_fabricante=numero\_fabricante} \wedge \texttt{ctx\_mombre=nombre} \wedge \texttt{ctx\_direccion=direccion}}(\rho_{\texttt{numero\_fabricante,nombre,direccion} \rightarrow \texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\texttt{Fabricantes}) \texttt{ x Fabricantes * Ventas})))$$

Ejemplo 5:

La consulta "devolver los números de fabricante que solo hayan realizado ventas a un precio de 200" , se expresa en *Xpath*$_{Sub}$ como:

```
/fabricante[not(venta/@precio != 200)]/@numero_fabricante
```

el predicado `venta/@precio != 200` es verdadero, para un elemento `fabricante`, si algún elemento `venta`, hijo, tiene un precio distinto de 200. En cambio el predicado `not(venta/@precio!= 200)` es verdadero si todos los elementos `venta`, hijos de un `fabricante`, tienen un precio distinto de 200.

La expresión de álgebra relacional que genera el algoritmo es:

$$\Pi_{\texttt{numero\_fabricante}}(\texttt{Fabricantes} * \texttt{Fabricantes} - \rho_{\texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion} \rightarrow \texttt{numero\_fabricante,nombre,dirccion}}(\Pi_{\texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\sigma_{\texttt{ctx\_numero\_fabricante=ctx\_numero\_fabricante} \wedge \texttt{ctx\_nombre=ctx\_nombre} \wedge \texttt{ctx\_direccion=ctx\_direccion} \wedge \texttt{precio =valor}}(R_1 \texttt{ x } R_2))))$$

siendo R_1:

$$\rho_{\texttt{numero\_fabricante,nombre,direccion} \rightarrow \texttt{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\texttt{Fabricantes}) \texttt{xValor}$$

donde `Valor` es la relación:

valor
200

y siendo R_2:

$$\Pi_{\text{precio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\Pi_{\text{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\sigma_{\text{ctx\_numero\_fabricante=numero\_fabricante} \wedge \text{ctx\_mombre=nombre} \wedge \text{ctx\_direccion=direccion}}(\rho_{\text{numero\_fabricante,nombre,direccion} \rightarrow \text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\text{Fabricantes}) \times \text{Fabricantes} * \text{Ventas})))$$

Ejemplo 6:

Por último la consulta *Xpath*$_{Sub}$:

```
/fabricante/venta[@fecha_venta]/producto
```

implica, en *Xpath*$_{Sub}$, el testeo de existencia del atributo *fecha_venta*. El algoritmo convierte este testeo de existencia a un testeo de no nulidad, de la forma `fecha_venta is not null` La expresión de álgebra relacional que genera el algoritmo es la siguiente:

$$\Pi_{\text{descripcion}}(\text{Fabricantes} * \text{Ventas} * \sigma_{\text{fecha\_venta is not null}}(\text{Fabricantes} * \text{Ventas}) * \text{Productos})$$

4.4.3 Fundamentos del algoritmo de conversión

El algoritmo, cuya especificación se presenta en la sección 4.5, está basado en las siguientes consideraciones:

1. En la base de datos relacional las columnas que denotan el mismo dato de información tienen el mismo nombre en todas las tablas en que aparecen (por ejemplo, *numero_fabricante* en las tablas ***Fabricantes*** y ***Ventas*** del ejemplo de la sección 4.4.2). Esto permite aplicar el `join` natural en las expresiones algebraicas que se generan.

2. Considerando las reglas indicadas en la sección 3.5, para definir las vistas *XML* de la base de datos relacional, se cumple:
   - Los elementos *XML* se corresponden con tablas de la base de datos relacional.
   - Los atributos *XML* se corresponden con columnas de las tablas de la base de datos relacional.
   - La relación de anidamiento entre elementos se corresponde con la relación de clave primaria-clave foránea de las tablas correspondientes.
   - En la vista *XML* no tienen porque estar todas las columnas de una determinada tabla, es decir que los atributos de un elemento *XML* pueden ser un subconjunto de las columnas de la tabla correspondiente al elemento.

3. El elemento `vista` aparece en el *XML Schema*, que especifica la vista *XML* de la base de datos, a los efectos de que los documentos *XML* pertenecientes a la clase de dicho *XML Schema* sean bien formados. Al ser un elemento ficticio y no tener un correspondiente en la base de datos no debe ser utilizado en las consultas. Por lo tanto las consultas de la sección 4.3.2 deben reescribirse como:

```
/fabricante/venta[@precio > 200]/@numero_producto
/fabricante[@numero_fabricante = 1]/venta
```

4. El algoritmo se basa en un mapeo que define las correspondencias entre los elementos y atributos del *XML Schema*, que especifica la vista *XML*, y la base de datos relacional. En el mapeo se especifica, para cada elemento y atributo *XML* la siguiente información, donde cada campo del mapeo se separa por ; (punto y coma):
   - Nombre del elemento o atributo.
   - Si es atributo (S) o no (N).
   - Nombre de la tabla o columna correspondiente.

Por ejemplo, para la base de datos y el *XML Schema* del ejemplo de la sección 3.5 el mapeo es:

```
fabricante;N;Fabricantes
numero_fabricante;S;numero_fabricante
nombre;S; nombre
direccion;S; direccion
venta;N;Ventas
precio;S; precio
fecha_venta;S;fecha_venta
fecha_envio;S;fecha_envio
numero_producto;S;numero_producto
producto:N;Productos
descripcion;S;descripción
```

Figura 49- Mapeo entre una base de datos y un XML Schema

Con el mapeo se resuelven las diferencias, si estas existen, entre los nombres de elementos y atributos *XML* y sus correspondientes nombres de tablas y columnas (sección 3.5).
Debido a que el mapeo es usado para establecer las correspondencias entre elementos *XML* con la base de datos en el *XML Schema* que especifica la vista *XML* de la base de datos no pueden existir nombres repetidos (sección 3.5).

5. El algoritmo asume que la consulta $Xpath_{Sub}$ que recibe es sintácticamente correcta y que los identificadores que aparecen en la consulta corresponden a identificadores de elementos o atributos *XML* que aparecen en el *XML Schema*. También se asume que la consulta es conforme a la relación padre hijo entre los elementos establecida en el *XML Schema* (o sea que no aparecen en la consulta ítems tales como `A/B` donde `B` no es hijo de `A`).

El algoritmo se basa en la reescritura de las reglas de la semántica denotacional, de la sección 4.3.2 las que definen la semántica formal del lenguaje $Xpath_{Sub}$, generando, en vez de conjuntos de nodos, una expresión en álgebra relacional.

La diferencia fundamental, entre el algoritmo y la semántica denotacional, es que el árbol de nodos, el cual es global a todas las funciones de la semántica, no es construído por el algoritmo, pero es simulado mediante el uso de parámetros adecuados (una secuencia de expresiones algebraicas y un valor entero).

El algoritmo consta de una función principal ($\mathbf{XPath_{Sub}ToAlg}$) que recibe como parámetros de entrada:

- un camino absoluto que corresponde a la consulta $Xpath_{Sub}$
- el mapeo que define las correspondencias entre el *XML Schema* y la base de datos
- el *XML Schema* que define la vista *XML* de la base de datos

y retorna una cadena de caracteres que representa la expresión en álgebra relacional o bien un error.

Asimismo, se definen funciones que se corresponden a cada una de las funciones utilizadas en la definición de la semántica denotacional. Estas funciones son invocadas por la función principal, directa o indirectamente.

4.5 Especificación del algoritmo de conversión de XpathSub a Álgebra Relacional

En esta sección se presenta la especificación de un algoritmo[16] que convierte una consulta $Xpath_{Sub}$ a álgebra relacional y en el Anexo 3 se muestran dos ejemplos de la aplicación del algoritmo.

El algoritmo (figura 50) tiene como parámetros de entrada un camino absoluto que corresponde a la consulta $Xpath_{Sub}$, el mapeo y el *XML Schema* que define la vista *XML* de la base de datos y retorna una cadena de caracteres que representa la expresión en álgebra relacional o un error.

El algoritmo de conversión no construye el árbol de nodos que representa los documentos de la clase definida por el *XML Schema* que especifica la vista *XML* de la base de datos, dado que dichos documentos no se generan. Sin embargo la conversión se realiza simulando en todo momento la existencia del árbol de nodos.

Para simular el árbol de nodos se utiliza una secuencia de expresiones algebraicas y un valor entero. La secuencia de expresiones algebraicas contiene las expresiones algebraicas que se generan a medida que se resuelve cada paso de localización de la consulta $Xpath_{Sub}$ y el valor entero representa el nivel en el árbol[17].

Con la información del nivel se detectan posibles errores en la consulta $Xpath_{Sub}$ como es el caso de la aplicación excesiva del eje `parent` (..). Por ejemplo en la consulta:

```
/fabricante/venta/../../..
```

- el nivel de / es 1
- el nivel de `/fabricante` es 2
- el nivel de `/fabricante/venta` es 3
- el nivel de `/fabricante/venta/..` es 2
- el nivel de `/fabricante/venta/../..` es 1 (se está referenciando a la raíz)
- al aplicar nuevamente `..` el nivel sería 0 y esto constituye un error porque la raíz (/) no tiene padre por definición.

El algoritmo se basa en la función X_{ALG} (sección 4.5.1) que se corresponde con la función $\boldsymbol{X}$ utilizada para definir la semántica de $Xpath_{Sub}$.
Esta función recibe como entradas un camino absoluto que corresponde a la consulta $Xpath_{Sub}$, una secuencia de expresiones algebraicas vacía, un entero (inicializado en 0), el mapeo y el *XML Schema* que define la vista *XML* de la base de datos.
Retorna un valor del tipo abstracto $\mathrm{ResX}_{\mathrm{ALG}}$ (Anexo 2) uno de cuyos componentes es una secuencia de expresiones algebraicas. Por la manera en que se construye esta secuencia de expresiones algebraicas, en el último lugar se encuentra una expresión algebraica que es equivalente al conjunto de nodos que retorna la función $\boldsymbol{X}$.
A partir de esta expresión se genera la expresión algebraica final, retornada por el algoritmo de conversión, que involucra los descendientes[18], si existen, del elemento involucrado en el último paso de localización de la consulta $Xpath_{Sub}$.

Para construir la expresión algebraica final se utilizan funciones auxiliares, las que se encuentran detalladas en el Anexo 2. Entre las funciones auxiliares utilizadas, las más relevantes son:

[16] El algoritmo completo se encuentra en el Anexo 2 (sección 9).
[17] El nivel de la raíz del árbol (/) se considera 1.
[18] Recordar que cada nodo es la raíz de un subárbol.

- La función `Descendientes` retorna el conjunto de nombres de elementos *XML* que son descendientes del elemento *XML* que forma parte del último paso de la consulta $Xpath_{Sub}$, o bien el conjunto vacío si el último paso involucra un atributo *XML*.

- La función `AtributosVista` recibe como parámetro de entrada un elemento *XML* y retorna un conjunto de nombres de columnas de la base de datos relacional que corresponden a los atributos del elemento *XML*, declarados en el *XML Schema* que define la vista *XML*[19].

Si el conjunto de descendientes es vacío se retorna la expresión algebraica obtenida por la invocación a la función $\boldsymbol{X_{ALG}}$.

En caso contrario se aplica `left outer join` (función $App_{Left}*$) para cada descendiente, de manera que la expresión algebraica final permita obtener todos los descendientes tengan o no hijos. Por último se aplica una proyección (función $App\Pi$) por los atributos del elemento involucrado en el último paso de la consulta y los atributos de sus descendientes que aparecen en la vista *XML* (dado que los atributos pueden ser un subconjunto de las columnas de la tabla correspondiente al elemento).
De esta manera se dispone, en la expresión algebraica, de todas las columnas de la tabla correspondiente al elemento (y de las tablas correspondientes a sus subelementos), a los efectos de construir un documento *XML* con los datos a retornar, ya sea conforme al *XML Schema* de la vista *XML* de la base de datos o bien al *XML Schema* independiente (sección 8.3).

$\boldsymbol{XPath_{Sub}ToAlg}$: Camino × Mapeo × Vista → String

$\boldsymbol{XPath_{Sub}ToAlg}$ [$XPath_{Sub}$](M,V) =
  let X = $\boldsymbol{X_{ALG}}$[$XPath_{Sub}$](∅,0,M,V)
  if EsError(X) then
   return "Error"
  else
   let Exp = Ult(SelSec(X))
   let Ult_Paso = Ult($XPath_{Sub}$)
   let D = Descendientes(ult_paso,V)
   if D = ∅ then
     return Exp
   else
     let a = ∅
     foreach d_i in D
       let a = a ∪ AtributosVista(d_i,M,V)
       let Exp = App_{Left}* (Exp,Tabla(M,d_i))
     return AppΠ(Exp,a)
   end if
  end if

Figura 50 – Algoritmo de conversión a álgebra relacional

[19] Según lo indicado en la sección 3.5, en la vista *XML* pueden no estar todas las columnas de una tabla.

4.5.1 Función X_{ALG}

La función X_{ALG} (figura 51) se corresponde con la función X utilizada para definir la semántica de $Xpath_{Sub}$. Esta función recibe como entradas un camino absoluto que corresponde a la consulta $Xpath_{Sub}$, una secuencia de expresiones algebraicas, un entero, el mapeo y el *XML Schema* que define la vista *XML* de la base de datos. Retorna un valor del tipo abstracto $\mathrm{ResX}_{\mathrm{ALG}}$ (Anexo 2) que es una terna formada por una secuencia de expresiones, un entero y un `booleano` que representa los casos de error.

El último ítem de la secuencia de expresiones retornada es, por construcción, la expresión algebraica que corresponde a la consulta $Xpath_{Sub}$.

Las reglas que definen esta función utilizan funciones auxiliares las cuales se detallan en el Anexo 2.

La regla X_{ALG}`[/]` retorna un valor de tipo $\mathrm{ResX}_{\mathrm{ALG}}$, uno de cuyos componentes es una secuencia de expresiones con un único ítem, que corresponde al `left outer join` (resultado de la invocación a la función `Oj`[20]) de todas las tablas que integran la vista *XML* de la base de datos y otro es el nivel (valor entero). Observar que se inicializa la secuencia de expresiones algebraicas y el nivel porque se procesa `/` (raíz) que representa todo el árbol de nodos lo cual es equivalente a la base de datos en su totalidad.
Las reglas X_{ALG}`[/cc]` y X_{ALG}`[/ca]` son similares, pero además realizan la conversión a álgebra de la subconsulta correspondiente al camino *cc* y *ca* respectivamente.

En las reglas X_{cc_ALG}`[cc/n]` y X_{cc_ALG}`[n]` no se aplica, en el caso general, proyección. Se aplica un `join` natural que es equivalente a la relación padre hijo, o de anidamiento, entre elementos, porque la misma se basa en la relación de clave primaria-clave foránea entre las correspondientes tablas de la base de datos.
De este modo la expresión algebraica que se genera tiene como columnas las columnas de la tabla correspondiente al elemento *n* y mantiene las columnas de la expresión algebraica generada al resolver la subconsulta *cc*. Esto permite que si un paso de localización posterior involucra el eje `parent (..)` la expresión algebraica correspondiente pueda ser generada aplicando una proyección (reglas X_{cc_ALG}`[cc/..]` y X_{cc_ALG}`[..]`)[21]
Se aplica proyección cuando la expresión algebraica generada, al resolver la subconsulta *cc*, es equivalente (tiene el mismo esquema) al `left outer join` de todas las tablas. En este caso es necesario proyectar para que las columnas de la expresión algebraica sean las de la tabla que corresponde al elemento *n* (en caso contrario tendría todas las columnas involucradas en la vista).

En las reglas X_{cc_ALG}`[cc/..]` y X_{cc_ALG}`[..]` se aplica, en el caso general, proyección por las razones ya expuestas. No se aplica proyección cuando la expresión algebraica a la cual se está haciendo referencia con `..` es equivalente (tiene el mismo esquema) al `left outer join` de todas las tablas (o sea la raíz /). En este caso se aplica un `join` natural con dicha expresión y la proyección es innecesaria porque la misma ya contiene todas las columnas posibles.

Además esta forma de resolución evita realizar proyecciones (innecesarias) en las reglas X_{cc_ALG}`[cc/n]` y X_{cc_ALG}`[n]` y en consecuencia realizar `joins` (innecesarios) en las reglas X_{cc_ALG}`[cc/..]` y X_{cc_ALG}`[..]`.

En la regla X_{cc_ALG}`[cc p]` se realiza, en primer lugar, la conversión de la subconsulta correspondiente al camino *cc* (X_{cc_ALG}`[cc]`) y luego se invoca la función P_{ALG}`[p]` (sección 4.5.2), que corresponde a la conversión a álgebra del predicado *p*.

[20] La función `Oj` retorna una expresión algebraica que corresponde `al left outer join` de todas las tablas que integran la vista *XML* de la base de datos.
[21] De todas maneras en el algoritmo principal se aplica la proyección.

Por construcción, la secuencia de expresiones algebraicas que retorna $\mathcal{X}_{cc\_ALG}[cc]$, contiene como último ítem (E_1) la expresión algebraica que representa los valores de cada contexto del predicado *p*. La función $\mathcal{P}_{ALG}[p]$ se define de manera que retorna una expresión algebraica (E_2) que representa solamente aquellos valores de contexto que satisfacen el predicado *p*, es decir que la expresión algebraica (E_2) tiene la misma estructura (columnas) que la expresión algebraica (E_1), lo cual permite determinar quienes satisfacen el predicado *p* aplicando un `join` natural de las expresiones E_1 y E_2.

$\mathcal{X}_{ALG}$: **Camino** × **SecExprAlg x Entero x Mapeo** × **Vista** → **Res**$\mathbf{X}_{ALG}$

$\mathcal{X}_{ALG}[/](S,N,M,V) = Cons(Ins(\emptyset,Oj(M,V)),1,\mathit{false})$

$\mathcal{X}_{ALG}[/cc](S,N,M,V) = \mathcal{X}_{cc\_ALG}[cc](Ins(\emptyset,Oj(M,V)),1,M,V)$

$\mathcal{X}_{ALG}[/ca](S,N,M,V) = \mathcal{X}_{ca\_ALG}[ca](Ins(\emptyset,Oj(M,V)),1,M,V)$

$\mathcal{X}_{cc\_ALG}$: **CaminoComun** × **SecExprAlg x Entero x Mapeo** × **Vista** → **Res**$\mathbf{X}_{ALG}$

```
X_cc_ALG[cc/n](S,N,M,V)=
      let X = X_cc_ALG[cc](S,N,M,V)
      if EsError(X) then
         return Cons(⊥,⊥,true)
      else
         let S1 = SelSec(X)
         let N1 = SelNiv(X)
         let E1 = Ult(S1)
         let E2 = Tabla(M,n)
         if IgualEsquema(E1,Oj(M,V)) then
            return Cons(Ins(S1,AppΠ(App*(E1,E2),Atributos(M,n))),N1+1,false)
         else
            return Cons(Ins(S1,App*(E1,E2)),N1+1,false)
         end if
     end if
```

```
X_cc_ALG[cc/..](S,N,M,V)=
      let X = X_cc_ALG[cc](S,N,M,V)
      if EsError(X) then
       return Cons(⊥,⊥,true)
      else
       if SelNiv(X) = 1 then
           return Cons(⊥,⊥,true)
       else
         let S1 = SelSec(X)
         let N1 = SelNiv(X)
         let E1 = Ult(S1)
         let E2 = Obt(S1,N1-1)
         if IgualEsquema(E2,Oj(M,V)) then
            return Cons(Ins(S1,App*(E1,E2)),N1-1,false)
         else
            return Cons(Ins(S1,AppΠ(E1,Atr(E2)),N1-1,false)
         end if
       end if
      end if
```

```
X_cc_ALG[cc p](S,N,M,V) =
      let X = X_cc_ALG[cc](S,N,M,V)
      if EsError(X) then
       return Cons(⊥,⊥,true)
      else
         let S1 = SelSec(X)
         let S2 = ElimUlt(S1)
         let E1 = Ult(S1)
         let N1 = SelNiv(X)
         let P = P_ALG[p](S1,N1,M,V)
         if EsError(P) then
          return Cons(⊥,true)
         else
          let E2 = SelExpr(P)
          return Cons(Ins(S2,App*(E1,E2)),N1,false)
       end if
      end if


X_cc_ALG[n](S,N,M,V)=
      if Vacia(S) then
         return Cons(⊥,⊥,true)
      else
         let E1 = Ult(S)
         let E2 = Tabla(M,n)
         if IgualEsquema(E1,Oj(M,V)) then
          return Cons(Ins(S,AppΠ(App*(E1,E2),Atributos(M,n))),N+1,false)
         else
          return Cons(Ins(S,App*(E1,E2)),N+1,false)
         end if
      end if


X_cc_ALG[..](S,N,M,V) =
      if N = 1 then
         return Cons(⊥,⊥,true)
      else
       if Vacia(S) then
          return Cons(⊥,⊥,true)
       else
         let E1 = Ult(S)
         let E2 = Obt(S,N-1)
         if IgualEsquema(E2,Oj(M,V)) then
            return Cons(Ins(Sec,App*(E1,E2)),N-1,false)
         else
            return Cons(Ins(Sec,AppΠ(E1,Atr(E2)),N-1,false)
         end if
       end if
      end if

X_ca_ALG: CaminoAtributo × SecExprAlg x Entero x Mapeo ×  Vista → ResX_ALG

X_ca_ALG[cc/@n](S,N,M,V)=
    let X = X_cc_ALG[cc](S,N,M,V)
    if EsError(X) then
     return Cons(⊥,⊥,true)
    else
     let S1 = SelSec(X)
     let N1 = SelNiv(X)
     let E = Ult(S1)
     return Cons(Ins(S1,AppΠ(E,Atributos(M,n))),N1+1,false)
   end if
```

Figura 51 – Función X_{ALG}

4.5.2 Función P_{ALG}

La función P_{ALG} (figura 52) se corresponde con la función P utilizada para definir la semántica de $Xpath_{Sub}$. Esta función recibe como entradas un predicado, una secuencia de expresiones algebraicas, un entero que representa el nivel, el mapeo y el *XML Schema* que define la vista *XML* de la base de datos.
Retorna un valor del tipo abstracto $ResP_{ALG}$ (Anexo 2) que es una pareja formada por un `string` que representa una expresión algebraica y un `booleano` que representa los casos de error.

Las funciones auxiliares utilizadas en la definición de la función P_{ALG} se encuentran detalladas en el Anexo 2.

Un predicado de la forma *@n* implica en $Xpath_{Sub}$ el testeo de existencia del atributo *n*. El algoritmo traduce este testeo de existencia a un testeo de no nulidad de la columna correspondiente (reglas $P_{ALG}[@n]$ y $P_{ALG}[ca]$).

Un predicado de la forma $not\ p$ se convierte a álgebra retornando la expresión algebraica que corresponde a la diferencia entre la expresión que representa a todos los candidatos y la expresión que representa a los que cumplen el predicado p.

Un predicado de la forma $p_1\ and\ p_2$ se convierte a álgebra retornando la expresión algebraica que corresponde a la intersección de las expresiones resultantes de convertir a álgebra los predicados p_1 y p_2, dado que se debe satisfacer ambos predicados.

Un predicado de la forma $p_1\ or\ p_2$ se convierte a álgebra retornando la expresión algebraica que corresponde a la unión de las expresiones resultantes de convertir a álgebra los predicados p_1 y p_2, dado que se debe satisfacer uno de los predicados o ambos.

Un predicado de la forma *cc* ó *ca* (caminos relativos) es convertido a álgebra usando las funciones X_{cc_ALG} y X_{ca_ALG} respectivamente (sección 4.5.1). La función auxiliar `Generar` asegura que la expresión algebraica retornada por las funciones X_{cc_ALG} y X_{ca_ALG} incluya como columnas las columnas de la expresión algebraica contexto del predicado (con un nombre diferente para que se conserven).
La función auxiliar `ResolverCam` asegura que la expresión algebraica tenga como columnas únicamente aquellas que corresponden a la expresión contexto con el nombre correcto (es decir con el nombre original).

Un predicado de la forma $o_1\ op\ o_2$ es convertido a álgebra utilizando la función O_{ALG}, que se aplica a cada uno de los dos operandos o_1 y o_2. Sobre las expresiones algebraicas que retorna la función O_{ALG} se aplica la función auxiliar `Generar` por las razones ya mencionadas.
La función auxiliar `ResolverComp` es la que efectivamente realiza la comparación para lo cual aplica el producto cartesiano y una selección (recordar la definición de la semántica de $Xpath_{Sub}$ de la sección 4.3.2 que implica la existencia de por lo menos dos valores que cumplan la comparación). Esta función, finalmente, asegura que la expresión algebraica final tenga como columnas únicamente aquellas que corresponden a la expresión algebraica contexto, con el nombre correcto (es decir con el nombre original).

P_{ALG}: Predicado × SecExprAlg x Entero x Mapeo × Vista → $ResP_{ALG}$

```
P_ALG[not p](S,N,M,V) =
    let E = Ult(S)
    let P = P_ALG[p](S,N,M,V)
    if ¬ EsError(P) then
     return Cons(App-(E,SelExpr(P)),false)
    else
     return Cons(⊥,true)
    end if
```

```
P_ALG[@n](S,N,M,V) =
    let E = Ult(S)
    let c = GenCond(Atributos(M,n),{"is not null"},"=")
    return Cons(Appσ(E,c),false)

P_ALG[cc](S,N,M,V) =
    let X = X_cc_ALG[cc](S,N,M,V)
    if ¬ EsError(X) then
     let S1 = SelSec(X)
     let E = Ult(S)
     let R = Generar(S1,S,E)
     return Cons(ResolverCam(R),false)
    else
     return Cons(⊥,true)
    end if

P_ALG[ca](S,N,M,V) =
    let X = X_ca_ALG[ca](S,N,M,V)
    if ¬ EsError(X) then
       let S1 = SelSec(X)
       let E = Ult(S)
       let R = Generar(S1,S,E)
       let c = GenCond(Atr(E),{"is not null"},"=")
       let Q = Appσ(R,c)
       return Cons(ResolverCam(Q),false)
    else
       return Cons(⊥,true)
    end if

P_ALG[p1 and p2](S,N,M,V)=
     let P1 = P_ALG[p1](S,N,M,V)
     let P2 = P_ALG[p2](S,N,M,V)
     if ¬ EsError(P1) ∧ ¬ EsError(P2)  then
        return Cons(App∩(SelExpr(P1),SelExpr(P2)),false)
     else
        return Cons(⊥,true)
     end if

P_ALG[p1 or p2](S,N,M,V) =
     let P1 = P_ALG[p1](S,N,M,V)
     let P2 = P_ALG[p2](S,N,M,V)
     if ¬ EsError(P1) ∧ ¬ EsError(P2)  then
        return Cons(App∪(SelExpr(P1),SelExpr(P2)),false)
     else
        return Cons(⊥,true)
     end if
P_ALG[o1 op o2](S,N,M,V)=
     let O1 = O_ALG[o1](S,N,M,V)
     let O2 = O_ALG[o2](S,N,M,V)
     if ¬ EsError(O1) ∧ ¬ EsError(O2)  then
        let S1 = SelSec(O1)
        let S2 = SelSec(O2)
        let E = Ult(S)
        let R1 = Generar(S1,S,E)
        let R2 = Generar(S2,S,E)
        return Cons(ResolverComp(R1,R2,SelAtr(O1),SelAtr(O2),op),false)
    else
        return Cons(⊥,true)
    end if
```

Figura 52 – Función P_{ALG}

4.5.3 Función O_{ALG}

La función $\boldsymbol{O_{ALG}}$ (figura 53) se corresponde con la función $\boldsymbol{O}$ utilizada para definir la semántica de $Xpath_{Sub}$. Esta función recibe como entradas un operando de un predicado, una secuencia de expresiones algebraicas, un entero que representa el nivel, el mapeo y el *XML Schema* que define la vista *XML* de la base de datos.
Retorna un valor del tipo abstracto ResO_{ALG} (Anexo 2) que es una terna formada por una secuencia de expresiones, un conjunto de nombres de columnas (con un único ítem) y un booleano que representa los casos de error.

Las funciones auxiliares utilizadas en la definición de la función $\boldsymbol{O_{ALG}}$ se encuentran detalladas en el Anexo 2.

El conjunto de nombres de columna es utilizado en la función $\boldsymbol{P_{ALG}}$ para definir la condición de comparación en la invocación a la función ResolverComp en la regla $\boldsymbol{P_{ALG}}[o_1\ op\ o_2]$.

Cuando se aplica a un valor *v* se crea una nueva relación, utilizando la función auxiliar CrearRel, de manera de poder aplicar la función ResolverComp en la regla $\boldsymbol{P_{ALG}}[o_1\ op\ o_2]$.

$\boldsymbol{O_{ALG}}$**: Operando × SecExprAlg x Entero x Mapeo × Vista → ResO**$_{\text{ALG}}$

$\boldsymbol{O_{ALG}}[ca](S,N,M,V) =$
  let X = $\boldsymbol{X_{ca\_ALG}}[ca](S,N,M,V)$
  if ¬ EsError(X) then
    let S_1 = SelSec(X)
    let E = Ult(S_1)
    return Cons(S_1,Atr(E),*false*)
  else
    return Cons(⊥,⊥,*true*)
  end if

$\boldsymbol{O_{ALG}}[@n](S,N,M,V)=$
  let E = Ult(S)
  return Cons(Ins(S,AppΠ(E,Atributos(M,*n*))),Atributos(M,*n*),*false*)

$\boldsymbol{O_{ALG}}[v](S,N,M,V) =$
  return Cons(Ins(S,CrearRel(v,"valor","valor")),{"valor"},*false*)

Figura 53 – Función $\boldsymbol{O_{ALG}}$

4.6 Mejoras al algoritmo

En esta sección se analizan dos mejoras al algoritmo presentado en la sección 4.5. En primer lugar se analiza como definir la regla $P_{ALG}[o_1\ op\ o_2]$ de la función P_{ALG} de manera de generar expresiones algebraicas más simples y en segundo lugar se muestra que es posible eliminar de la consulta $Xpath_{Sub}$ el eje `parent (..)` transformándola en una consulta equivalente, dejado planteado el proceso a desarrollar.

4.6.1 Función P_{ALG}

El hecho de realizar la conversión a expresiones algebraicas tratando de reflejar lo más fielmente posible la semántica de $Xpath_{Sub}$ produce, en la aplicación de la regla $P_{ALG}[o_1\ op\ o_2]$, un tratamiento de manera uniforme de todas las posibles combinaciones de operandos que pueden darse. Esto implica que aún en los casos mas simples de comparaciones (por ejemplo entre un atributo *XML* y un valor) se generen expresiones algebraicas de cierta complejidad. Estas expresiones algebraicas complejas pueden ser simplificadas si se analiza la forma[22] de cada uno de los operandos que intervienen en la comparación.

La nueva definición de la regla $P_{ALG}[o_1\ op\ o_2]$, basada en el análisis de la forma de los operandos, se muestra en la figura 54. Las restantes reglas de la función P_{ALG} permanecen incambiadas.

En esta nueva definición de la regla se observa que cuando ambos operandos son atributos o valores se retorna una expresión algebraica que consiste en una selección y que solamente se realiza un tratamiento similar al descripto en la sección 4.5.2 (utilizando las funciones `Generar` y `ResolverComp`) cuando uno de los operandos, o ambos, son caminos que finalizan en atributos.

Las funciones auxiliares utilizadas se encuentran detalladas en el Anexo 2.

Como consecuencia de la nueva definición de regla $P_{ALG}[o_1\ op\ o_2]$ es posible modificar la función O_{ALG}, dado que solamente es invocada cuando un operando es un camino que finaliza en atributo. La nueva versión se muestra en la figura 55.

En los ejemplos de conversión presentados en la sección 4.4.2 y en el Anexo 3 (sección 10) se utiliza esta nueva versión de la función P_{ALG} porque permite simplificar considerablemente las expresiones algebraicas generadas.

[22] Observar que por definición los operandos pueden ser atributos *XML*, valores o caminos que finalizan en un atributo.

```
P_ALG: Predicado × SecExprAlg x Entero x Mapeo ×  Vista → ResP_ALG

P_ALG[not p](S,N,M,V) =
    let E = Ult(S)
    let P = P_ALG[p](S,N,M,V)
    if ¬ EsError(P) then
     return Cons(App-(E,SelExpr(P)),false)
    else
     return Cons(⊥,true)
    end if


P_ALG[@n](S,N,M,V) =
    let E = Ult(S)
    let c = GenCond(Atributos(M,n),{"is not null"},"=")
    return Cons(Appσ(E,c),false)

P_ALG[cc](S,N,M,V) =
    let X = X_cc_ALG[cc](S,N,M,V)
    if ¬ EsError(X) then
     let S1 = SelSec(X)
     let E = Ult(S)
     let R = Generar(S1,S,E)
     return Cons(ResolverCam(R),false)
    else
     return Cons(⊥,true)
    end if

P_ALG[ca](S,N,M,V) =
    let X = X_ca_ALG[ca](S,N,M,V)
    if ¬ EsError(X) then
       let S1 = SelSec(X)
       let E = Ult(S)
       let R = Generar(S1,S,E)
       let c = GenCond(Atr(E),{"is not null"},"=")
       let Q = Appσ(R,c)
       return Cons(ResolverCam(Q),false)
    else
       return Cons(⊥,true)
    end if

P_ALG[p1 and p2](S,N,M,V)=
     let P1 = P_ALG[p1](S,N,M,V)
     let P2 = P_ALG[p2](S,N,M,V)
     if ¬ EsError(P1) ∧ ¬ EsError(P2)  then
        return Cons(App∩(SelExpr(P1),SelExpr(P2)),false)
     else
        return Cons(⊥,true)
     end if

P_ALG[p1 or p2](S,N,M,V) =
     let P1 = P_ALG[p1](S,N,M,V)
     let P2 = P_ALG[p2](S,N,M,V)
     if ¬ EsError(P1) ∧ ¬ EsError(P2)  then
        return Cons(App∪(SelExpr(P1),SelExpr(P2)),false)
     else
        return Cons(⊥,true)
end if
```

```
P_ALG[o1 op o2](S,N,M,V) =
   let E = Ult(S)
   if EsAtributo(o1) ∧ EsAtributo(o2) then
      return ResolverAtr_Atr(M,E,o1,o2,op)
   else
      if EsAtributo(o1) ∧ EsValor(o2) then
         return ResolverAtr_Val(M,E,o1,o2,op)
      else
         if EsAtributo(o1) ∧ EsCaminoAtributo(o2) then
          return ResolverAtr_Cam(M,S,E,o1,o2,op,N)
         else
            if EsValor(o1) ∧ EsAtributo(o2) then
               return P_ALG[o2 NuevoOp(op) o1](S,N,M,V)
            else
               if EsValor(o1) ∧ EsValor(o2) then
                  return ResolverVal_Val(M,E,o1,o2,op)
               else
                  if EsValor(o1) ∧  EsCaminoAtributo(o2) then
                     return ResolverVal_Cam(M,S,E,o1,o2,op,N)
                  else
                     if EsCaminoAtributo(o1) ∧ EsAtributo(o2) then
                        return P_ALG[o2 NuevoOp(op) o1](S,N,M,V)
                     else
                        if EsCaminoAtributo(o1) ∧ EsValor(o2) then
                           return P_ALG[o2 NuevoOp(op) o1](S,N,M,V)
                        else
                           if EsCaminoAtributo(o1) ∧ EsCaminoAtributo(o2) then
                              return ResolverCam_Cam(M,S,E,o1,o2,op,N)
                           end if
                        end if
                     end if
                  end if
               end if
            end if
         end if
      end if
    end if

ResolverAtr_Atr: Mapeo x string x Operando x Operando x Operador → ResP_ALG
ResolverAtr_Atr(M,E,o1,o2,op) =
   let c= GenCond(Atributos(M,ObtAtrib(o1)),Atributos(M,ObtAtrib(o2)),op)
   return Cons(Appσ(E,c),false)


ResolverAtr_Val: Mapeo x string x Operando x Operando x Operador → ResP_ALG
ResolverAtr_Val(M,E,o1,o2,op) =
   let c = GenCond(Atributos(M,ObtAtrib(o1)),{o2},op)
   return Cons(Appσ(E,c),false)

ResolverAtr_Cam : Mapeo x SecExprAlg x string x Operando x Operando x
                  Operador x Entero→ ResP_ALG
ResolverAtr_Cam(M,S,E,o1,o2,op,N) =
   let A = Atr(E) – Atributos(M,ObtAtrib(o1))
   let E_ctx= Appρ(E,A,FuncRen(A))
   let O2 = O_ALG[o2](S,N,M,V)
   if ¬ EsError(O2)  then
      let S2 = SelSec(O2)
      let R = Generar(S2,S,E)
      returnCons(ResolverComp(E_ctx,R,Atributos(M,ObtAtrib(o1)),SelAtr(O2),op),false)
   else
      return Cons(⊥,true)
  end if
```

```
ResolverVal_Val: Mapeo x string x Operando x Operando x Operador→ ResP_ALG
ResolverVal_Val(M,E,o1,o2,op) =
   let c = GenCond({o1},{o2},op)
   return Cons(Appσ(E,c),false)

ResolverVal_Cam: Mapeo x SecExprAlg x string x Operando x Operando x
                 Operador x Entero→ ResP_ALG
ResolverVal_Cam(M,S,E,o1,o2,op,N) =
   let A = Atr(E)
   let E_ctx= Appρ(E,A,FuncRen(A))
   let R1 = Appx(E_ctx,CrearRel(o1,"valor","valor"))
   let O2 = O_ALG[o2](S,N,M,V)
   if ¬ EsError(O2)  then
      let S2 = SelSec(O2)
      let R2 = Generar(S2,S,E)
      return Cons(ResolverComp(R1,R2,"valor",SelAtr(O2),op),false)
   else
      return Cons(⊥,true)
 end if


ResolverCam_Cam: Mapeo x SecExprAlg x string x Operando x Operando x
                 Operador x Entero→ ResP_ALG
ResolverCam_Cam(M,S,E,o1,o2,op,N) =
   let O1 = O_ALG[o1](S,N,M,V)
   let O2 = O_ALG[o2](S,N,M,V)
   if ¬ EsError(O1) ∧ ¬ EsError(O2)  then
      let S1 = SelSec(O1)
      let S2 = SelSec(O2)
      let R1 = Generar(S1,S,E)
      let R2 = Generar(S2,S,E)
      return Cons(ResolverComp(R1,R2,SelAtr(O1),SelAtr(O2),op),false)
 else
      return Cons(⊥,true)
 end if
```

Figura 54 – Redefinición de la función P_{ALG}

```
O_ALG: Operando × SecExprAlg x Entero x Mapeo ×  Vista → ResO_ALG


O_ALG[ca](S,N,M,V) =
      let X = X_ca_ALG[ca](S,N,M,V)
      if ¬ EsError(X) then
         let S1 = SelSec(X)
         let E = Ult(S1)
         return Cons(S1,Atr(E),false)
      else
         return Cons(⊥,⊥,true)
      end if
```

Figura 55 – Redefinición de la función O_{ALG}

4.6.2 Eliminación del eje `parent`

Los ejes de *XPath* [27] pueden ser divididos en dos categorías dependiendo de los nodos del árbol que permiten seleccionar:

- `Forward`: son aquellos ejes que seleccionan nodos que ocurren después que el nodo contexto, en el orden del documento.
- `Reverse`: son aquellos ejes que seleccionan nodos que ocurren antes que el nodo contexto, en el orden del documento.

De acuerdo a lo anterior, en $Xpath_{Sub}$, el eje `child` es un eje de categoría `forward` y el eje `parent (..)` es un eje de categoría `reverse`.

En esta sección se analiza la posibilidad de eliminar de las consultas $Xpath_{Sub}$ el eje `parent (..)`, generando consultas $Xpath_{Sub}$ equivalentes. Si bien no se pretende profundizar en este aspecto sino simplemente mostrar que es posible, esto permitiría definir un nuevo enfoque consistente en realizar un preprocesamiento de la consulta $Xpath_{Sub}$, previamente a su conversión a álgebra relacional, con el objetivo de simplificar dicha conversión.

Un paso de localización que involucra al eje `parent (..)`, que forma parte de un camino de localización puede ser eliminado, si el camino de localización es modificado de forma conveniente:

Sean a y b nombres de elementos, $a/b/.. \equiv a[b]$

La demostración de la equivalencia se realiza usando la semántica definida en la sección 4.3.2:

$\boldsymbol{X}_{cc}[a\ b](x)=$

$\{x_1 / x_1 \in \boldsymbol{X}_{cc}[a](x) \wedge \boldsymbol{P}[b](x_1)\}=$

$\{x_1 / x_1 \in Hijos(x) \wedge Nombre(x_1)=a \wedge \boldsymbol{P}[b](x_1)\}=$

$\{x_1 / x_1 \in Hijos(x) \wedge Nombre(x_1)=a \wedge (\boldsymbol{X}_{cc}[b](x_1) \# \emptyset)\}$ **(1)**

aplicando $\boldsymbol{X}_{cc}[b](x_1) \# \emptyset \Rightarrow \exists x_2.(x_2 \in \boldsymbol{X}_{cc}[b](x_1))$ en **(1)**

(1) = $\{x_1 / \exists x_2.(x_1 \in Hijos(x) \wedge Nombre(x_1)=a \wedge x_2 \in \boldsymbol{X}_{cc}[b](x_1))\}=$

$\{x_1 / \exists x_2.(x_1 \in Hijos(x) \wedge Nombre(x_1)=a$
$\wedge x_2 \in \{x_3 / x_3 \in Hijos(x_1) \wedge Nombre(x_3)=b\})\} =$

$\{x_1 / \exists x_2.(x_1 \in Hijos(x) \wedge Nombre(x_1)=a \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2)=b)\}$ **(A)**

$\boldsymbol{X}_{cc}[a/b/..](x)=$

$\{x_1 / \exists x_2.(x_2 \in \boldsymbol{X}_{cc}[a/b](x) \wedge x_1 \in Padre(x_2))\}=$

$\{x_1 / \exists x_2.(x_2 \in \{x_3 / \exists x_4.(x_4 \in \boldsymbol{X}_{cc}[a](x) \wedge x_3 \in Hijos(x_4) \wedge Nombre(x_3)=b)\} \wedge$
$x_1 \in Padre(x_2))\}=$

$\{x_1 / \exists x_2\ \exists x_4.(x_4 \in \boldsymbol{X}_{cc}[a](x) \wedge x_2 \in Hijos(x_4) \wedge Nombre(x_2)=b \wedge x_1 \in Padre(x_2))\}=$

$\{x_1 / \exists x_2\ \exists x_4.(x_4 \in \{x_5 / x_5 \in Hijos(x) \wedge Nombre(x_5)=a\} \wedge x_2 \in Hijos(x_4)$
$\wedge Nombre(x_2)=b \wedge x_1 \in Padre(x_2))\}=$

$\{x_1 / \exists x_2\ \exists x_4.(x_4 \in Hijos(x) \wedge Nombre(x_4)=a \wedge x_2 \in Hijos(x_4) \wedge Nombre(x_2)=b \wedge$
$x_1 \in Padre(x_2))\}$ **(2)**

aplicando $x_2 \in Hijos(x_4) \Rightarrow x_4 \in Padre(x_2)$ en **(2)**

(2) = $\{x_1 / \exists x_2 \exists x_4.(x_4 \in Hijos(x) \wedge Nombre(x_4)=a \wedge x_4 \in Padre(x_2) \wedge Nombre(x_2)=b \wedge x_1 \in Padre(x_2))\}$ **(3)**

aplicando $x_1 \in Padre(x_2) \wedge x_4 \in Padre(x_2) \Rightarrow x_1 = x_4$ en **(3)**

(3) = $\{x_1 / \exists x_2 (x_1 \in Hijos(x) \wedge Nombre(x_1)=a \wedge Nombre(x_2) = b \wedge x_1 \in Padre(x_2))\}$ **(4)**

aplicando $x_1 \in Padre(x_2) \Rightarrow x_2 \in Hijos(x_1)$ en **(4)**

(4) = $\{x_1 / \exists x_2 (x_1 \in Hijos(x) \wedge Nombre(x_1)=a \wedge x_2 \in Hijos(x_1) \wedge Nombre(x_2) = b)\}$ **(B)**

$\Rightarrow$ **(A)** $\equiv$ **(B)** $\Rightarrow \boldsymbol{X}_{cc}[a/b/..](x) \equiv \boldsymbol{X}_{cc}[a\ b](x) \Rightarrow a/b/.. \equiv a[b]$

La equivalencia demostrada permite sustituir parte de una consulta por su equivalente, por ejemplo:

$$/p_1/p_2/p_3/../p_4/p_5$$

es equivalente a:

$$/p_1/p_2[p_3]/p_4/p_5$$

La demostración de equivalencia anterior es una de las varias que pueden ser analizadas y, en todo caso, ser demostradas, por lo cual un posible trabajo futuro consiste en analizar otras equivalencias (por ejemplo cuando el eje `parent` ocurre dentro de un predicado) de manera de obtener un conjunto de reglas de equivalencia que permitan eliminar completamente los ejes `parent` de una consulta $Xpath_{Sub}$ y transformar la misma en una consulta $Xpath_{Sub}$ equivalente que solamente involucre al eje `child`. Esta transformación permitiría simplificar considerablemente el algoritmo presentando, por ejemplo, reduciendo la cantidad de reglas del mismo.

4.7 Observaciones sobre el algoritmo de conversión

Xpath$_{Sub}$, al ser un subconjunto de *XPath*, maneja un conjunto sumamente reducido de tipos de datos [27]. Este conjunto de tipos de datos es insuficiente para expresar consultas a una base de datos relacional por lo cual se asume que los tipos de datos disponibles, para ser usados en las consultas, son los tipos de datos que manejan los *XML Schema* que son similares a los tipos de datos de las bases de datos relacionales.

XPath aplica una serie de reglas de conversión de tipos de datos [27], cuando ocurren comparaciones en los predicados, para determinar si un predicado evalúa a verdadero o falso. En cambio el algoritmo definido no aplica estas reglas de conversión de tipos de datos porque las consultas deberán ser resueltas, finalmente, por un manejador de bases de datos relacional y porque al extender el conjunto de tipos de datos soportados, las reglas, tal como están definidas, no son aplicables.

Sobre las expresiones algebraicas que genera el algoritmo puede definirse, en caso de considerarse necesario, un proceso de optimización de las mismas. A modo de ejemplo y sin profundizar en el tema, por estar fuera del alcance de esta tesis, una optimización sumamente simple, de las expresiones generadas, es la de sustituir las subexpresiones algebraicas de la forma:

$$\Pi_{Atr(T)}(Oj(M,V)* T)$$

donde T es una tabla de la base de datos relacional, por la subexpresión algebraica:

$$T$$

Por otro lado, varios manejadores comerciales de bases de datos soportan la realización de consultas *XPath* a una base de datos relacional (sección 2.7), uno de los más difundidos es Microsoft SQL Server 2000 el cual soporta un subconjunto del lenguaje *XPath* (similar a *Xpath*$_{Sub}$) y realiza internamente la conversión a *SQL* mediante un algoritmo que no es público y del cual no se han encontrado muchos detalles.
En la documentación de Microsoft SQL Server 2000 se advierte explícitamente cuales son las limitaciones del algoritmo, una de ellas es que las consultas con predicados en los cuales ocurran comparaciones entre dos operandos, siendo ambos operandos caminos de localización (consultas denominadas en la documentación como `cross product`), como ser:

```
/fabricante[venta/@fecha_venta = venta/@fecha_envio]
```

no son soportadas [52]. Sin embargo el algoritmo de conversión a álgebra relacional definido sí soporta este tipo de predicados.

Capítulo 5 Implementación

5.1 Introducción

En este capítulo se describen las plataformas y lenguajes de programación utilizados para implementar el prototipo del `framework` y se muestra la aplicación cliente utilizada para su testeo.

5.2 Descripción del Prototipo

La implementación del `framework`, del lado del servidor, se realizó sobre una plataforma Windows.

La *API ANS*, incluyendo la conversión de las consultas en $Xpath_{Sub}$ a *SQL* (basada en el algoritmo definido) fue implementada utilizando el lenguaje Java versión 1.4.0 [53].

Como servidor *SOAP* se utilizó la distribución `open source` de Apache *SOAP* [46], versión 2.2, que es la implementación de Apache Software Foundation [47] de la especificación de *SOAP* del W3C. Esta distribución comprende un `servlet` que escucha los requerimientos *SOAP* y un conjunto de clases Java que permiten trasladar los requerimientos a invocaciones a los métodos de *ANS* y los resultados, de dichas invocaciones, a respuestas *SOAP* para ser enviadas al cliente.

Para ejecutar el `servlet` se requiere de un `Web Application Server` que soporte `servlets` y *JSP* (**J**ava **S**erver **P**ages), para lo cual se utilizó Tomcat, versión 3.2.3 [54], que es una distribución `open source` de Apache Software Foundation y también Resin [55].

Para realizar los testeos del `framework` se utilizó como manejador de bases de datos Access y se implementó una aplicación cliente, utilizando el lenguaje Java, versión 1.4.0, que utiliza las librerías cliente provistas por la distribución Apache *SOAP*.

A continuación se muestra la `interface` de la aplicación cliente y su utilización para testear los distintos aspectos de la *API ANS*, asumiendo la utilización de vistas *XML*:

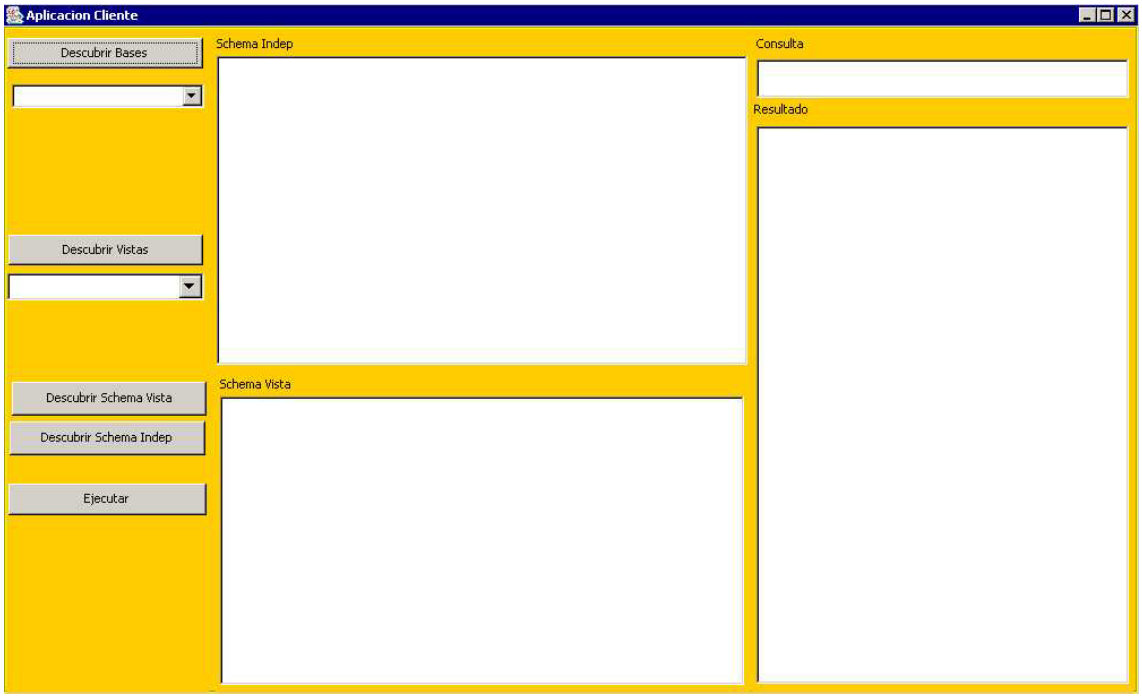


Figura 56 – Interface gráfica de aplicación cliente

La `interface` permite acceder a las principales funcionalidades de la *API ANS*.

A continuación se muestra la operación de la `interface` gráfica siguiendo el ejemplo de uso del `framework` de la sección 3.6.

Al presionar el botón *Descubrir Bases* se envía, al servidor, un mensaje *SOAP* conteniendo la invocación al método **Descubrir** de la *API ANS*. La aplicación cliente, al recibir la respuesta *SOAP*, extrae la respuesta a la invocación realizada y muestra la lista de las bases de datos disponibles:

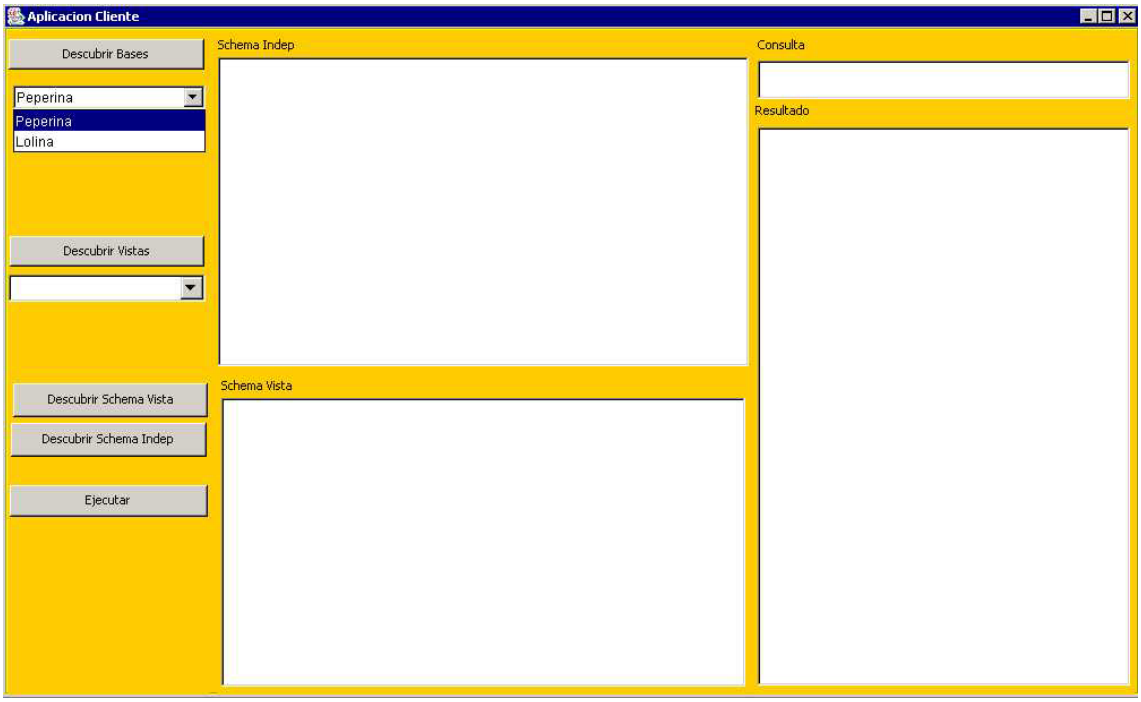


Figura 57 – Interface gráfica de aplicación cliente, lista de bases de datos

Al presionar el botón *Descubrir Vistas* se envía, al servidor, un mensaje *SOAP* conteniendo la invocación al método **Descubrir** de la *API ANS*. La aplicación cliente, al recibir la respuesta *SOAP*, extrae la respuesta a la invocación realizada y muestra la lista de las vistas disponibles para la base de datos seleccionada:

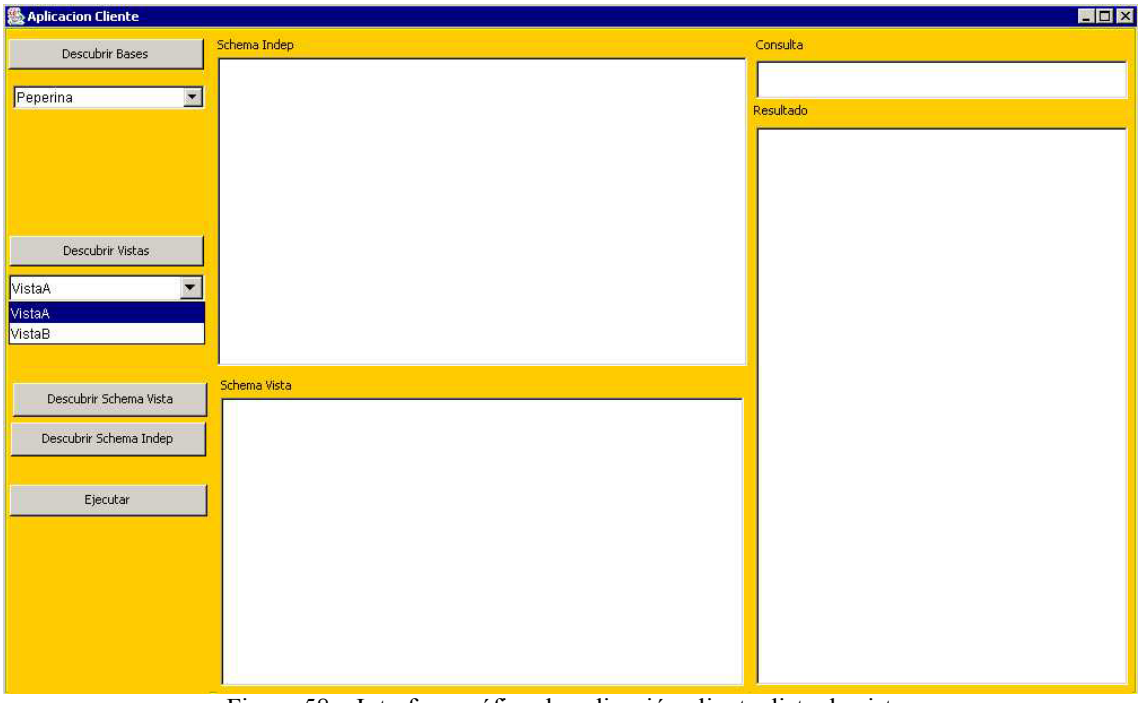


Figura 58 – Interface gráfica de aplicación cliente, lista de vistas

Al presionar el botón *Descubrir Schema Indep* se envía, al servidor, un mensaje *SOAP* conteniendo la invocación al método **Descubrir** de la *API ANS*. La aplicación cliente, al recibir la respuesta *SOAP*, extrae la respuesta a la invocación realizada y muestra el *XML Schema* independiente, el cual corresponde a uno de los posibles formatos en que se obtienen los resultados de las consultas. El otro formato corresponde al *XML Schema* que especifica la vista *XML* de la base de datos:

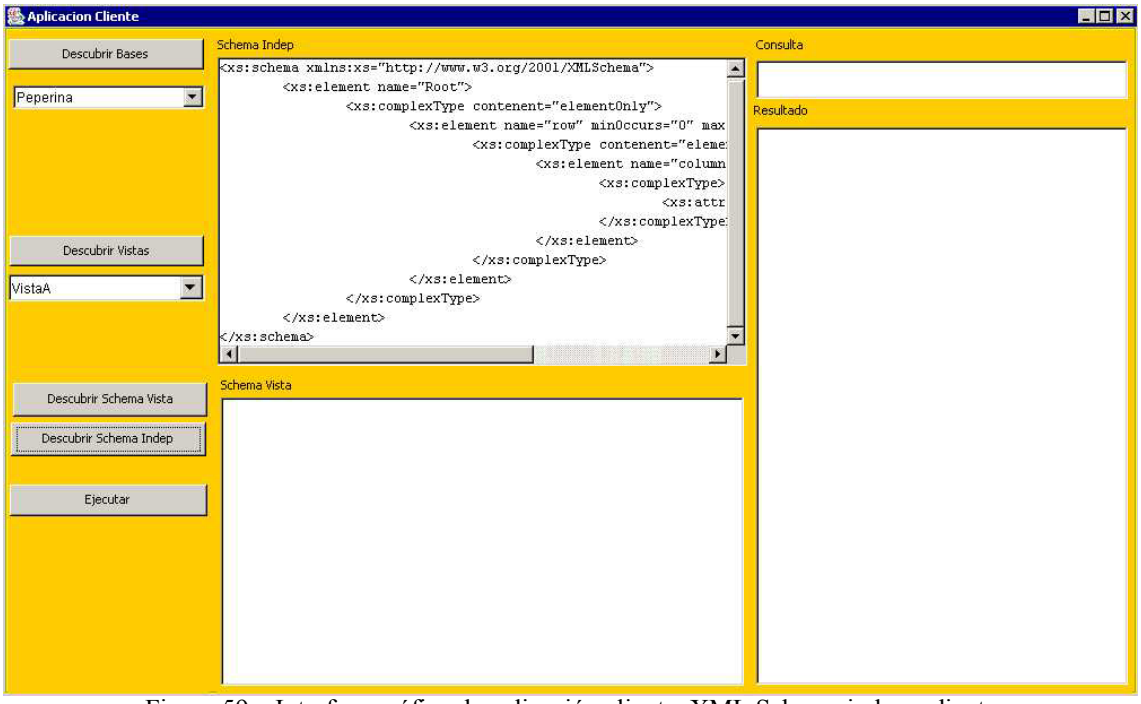


Figura 59 – Interface gráfica de aplicación cliente, XML Schema independiente

Al presionar el botón *Descubrir Schema Vista* se envía, al servidor, un mensaje *SOAP* conteniendo la invocación al método **Descubrir** de la *API ANS*. La aplicación cliente, al recibir la respuesta *SOAP*, extrae la respuesta a la invocación realizada y muestra el *XML Schema* de la vista *XML* seleccionada de la base de datos, sobre el cual se realizan las consultas $Xpath_{Sub}$:

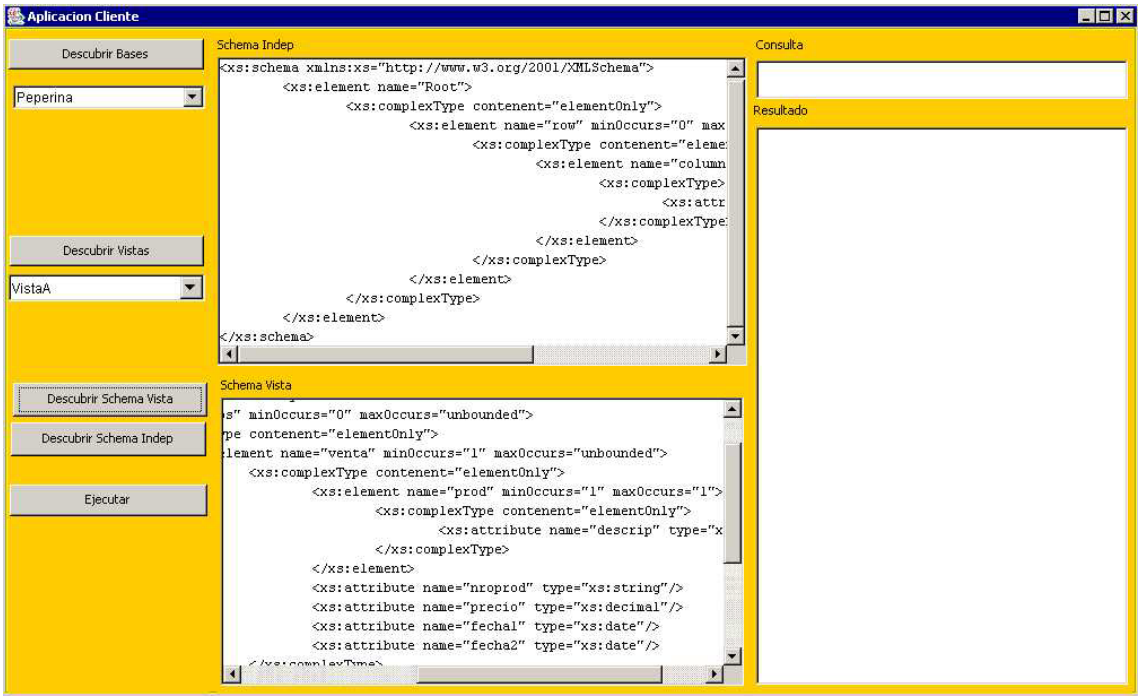


Figura 60 – Interface gráfica de aplicación cliente, XML Schema de una vista

Una vez seleccionada la base de datos y la vista y teniendo visibles los *XML Schemas*, se efectúa la consulta $Xpath_{Sub}$.
Al presionar el botón *Ejecutar* se envía, al servidor, un mensaje *SOAP* conteniendo la invocación al método **Ejecutar** de la *API ANS*. La aplicación cliente, al recibir la respuesta *SOAP*, extrae la respuesta a la invocación realizada y muestra un documento *XML* que contiene los datos[23]:

[23] En esta implementación de la *API ANS* los resultados de una consulta se obtienen en el formato especificado por el *XML Schema* independiente.

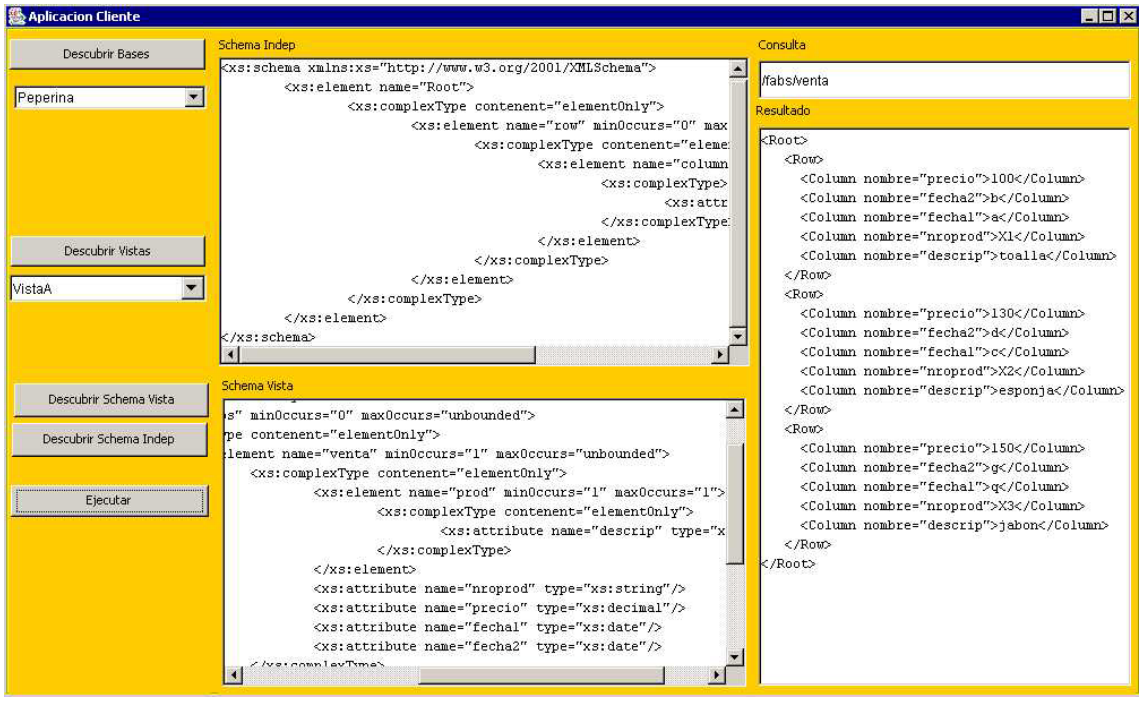


Figura 61 – Interface gráfica de aplicación cliente, consulta y su resultado

Capítulo VI Conclusiones

6.1 Introducción

En este capítulo se presentan conclusiones sobre la tesis realizada y sobre las principales tecnologías involucradas.

6.2 Conclusiones de la tesis

Un aspecto relevante de la presente tesis es que integra varias tecnologías open source, algunas de ellas sumamente recientes o emergentes y otras ya consolidadas, de manera de definir un framework que cumple con los objetivos planteados de permitir la realización de consultas a bases de datos relacionales a través de internet haciendo transparente el diseño lógico de las mismas.

El uso de tecnologías de base recientes o emergentes dificultó, inicialmente, la investigación debido a los frecuentes cambios que presentan. Un ejemplo típico de esto es el caso de *SOAP* que si bien está construido sobre tecnologías más estables su especificación ha cambiado varias veces desde el inicio de esta tesis.
A pesar que *SOAP* no es aún estándar por si mismo, el apoyo de las industrias de software creciente y el hecho de estar basado en estándares de amplia aceptación permiten prever que se transformará en un estándar en un plazo sumamente breve.

Entre las tecnologías de base consolidadas se encuentran *XML*, *XML Schemas* y *XPath* que son recomendaciones oficiales del W3C, y por lo tanto pueden ser consideradas estándares en sus respectivas áreas.

En definitiva puede concluirse que el framework diseñado en esta tesis integra tecnologías open source de punta que actualmente son, o que pronto llegarán a ser, estándares. El framework representa una solución que:

- es tecnológicamente independiente de lenguajes de programación y plataformas
- presenta un bajo acoplamiento entre la aplicación cliente y el servidor de base de datos
- brinda flexibilidad a las aplicaciones cliente para la realización de consultas

en un área donde muchas de las soluciones disponibles son soluciones propietarias provistas por los principales sistemas comerciales de bases de datos y por lo tanto no tienen las características enumeradas.

Otro aspecto relevante del framework es el diseño y la especificación de la *API ANS* que permite la interacción en el acceso remoto a datos entre una aplicación cliente y bases de datos relacionales en el ámbito de internet, mediante los métodos ***Descubrir*** y ***Ejecutar***, los cuales reciben y envían datos en formato *XML*, que se está transformando en el formato estándar para intercambio de información en internet.

Por último, en esta tesis se presenta la especificación de la semántica formal del lenguaje $Xpath_{Sub}$ (subconjunto funcional de *XPath*) y de un algoritmo que realiza la conversión de consultas $XPath_{Sub}$ a álgebra relacional. Por lo que está en nuestro conocimiento no existen estudios previos, por lo menos que sean públicos, en lo relativo a la especificación de un algoritmo que convierte consultas expresadas en un lenguaje de consulta *XML* a expresiones de álgebra relacional.

Además el algoritmo presentado soporta la conversión de consultas con predicados en los cuales ocurren comparaciones entre dos operandos, siendo ambos operandos caminos de localización. Este tipo de consultas no son soportadas por Microsoft SQL Server que utiliza un subconjunto de *XPath*, similar a $Xpath_{Sub}$, el cual es uno de los manejadores de bases de datos comerciales más difundidos actualmente.

Capítulo VII Trabajo Futuro

7.1 Introducción

En este capítulo se enumeran algunos posibles trabajos futuros que surgen de aspectos que no han sido considerados en profundidad en la presente tesis o que no se encuentran dentro del alcance de la misma.

7.2 Trabajos Futuros

1. Incorporar la generación semiautomática de vistas *XML* de una base de datos. Esto implica el desarrollo de herramientas informáticas (posiblemente con `interfaces` gráficas) adecuadas mediante las cuales un usuario experto con conocimiento de la base de datos relacional (posiblemente un administrador) pueda generar los *XML Schemas* que especifican las vistas *XML*. Estas herramientas deben ser necesariamente semiautomáticas dado que se requiere tomar ciertas decisiones (como ser cuales serán las tablas que integren la vista, cuales serán las columnas seleccionadas, etc.) que deben ser hechas por un usuario experto. De todas maneras estas herramientas pueden asistir al usuario experto interactuando con él cuando sea requerido.
2. Definir y aplicar técnicas de optimización a las expresiones en álgebra relacional generadas por el algoritmo de conversión previamente a su conversión a *SQL*.
3. Incorporar nuevas funcionalidades a $XPath_{Sub}$. En particular se pueden incorporar otros ejes (por ejemplo `descendant`, `ancestor`, etc.) que permitan un mayor poder de expresividad en las consultas por parte de las aplicaciones cliente.
4. Profundizar en el enfoque de buscar y demostrar nuevas equivalencias. De esta manera se podría transformar completamente una consulta $XPath_{Sub}$, previamente a su conversión a álgebra relacional, eliminando de la misma cualquier aparición del eje `parent` (`..`). De todas maneras es necesario profundizar en este enfoque estudiando su factibilidad y aplicabilidad.
   Además, si se incorporan nuevos ejes pertenecientes a la categoría de ejes `reverse` (por ejemplo `ancestor`), es necesario buscar y demostrar otras equivalencias que permitan eliminarlos y así transformar la consulta $XPath_{Sub}$ en una consulta equivalente que involucre únicamente ejes de la categoría `forward`.
5. Sustituir el lenguaje de consulta $XPath_{Sub}$. En particular si bien actualmente no existe un lenguaje de consulta de *XML* que sea estándar, el W3C se encuentra trabajando en su definición y todo apunta a que el lenguaje *XQuery* [28] se transforme en el lenguaje de consulta estándar de *XML*. Por lo tanto en el futuro puede sustituirse $XPath_{Sub}$ por *XQuery*. Este cambio en el lenguaje de consulta implica la especificación de nuevos algoritmos de conversión, los cuales pueden tomar como base el trabajo realizado en la presente tesis dado que *XQuery* incorpora muchas de las funcionalidades de *XPath* (y en consecuencia de $XPath_{Sub}$) [48].
6. Por último es sumamente simple transformar el `framework` presentado en un `Web Service`. Un `Web Service` es una colección de funciones que puede ser descripta, publicada, localizada e invocada a través de una red (generalmente `internet`) para ser usada por otros programas [56]. Los `Web Services` son los bloques constructivos que permiten crear una nueva generación de sistemas distribuidos [57].
   Las tecnologías en las cuales se basan los `Web Service` son las siguientes:

   - *XML*: un formato estándar para intercambiar información a través de `internet`
   - *SOAP*: un protocolo de comunicación para realizar las invocaciones remotas

   que son las tecnologías sobre las que se basa el `framework` presentado en esta tesis.

   En cuanto a la descripción, publicación y localización intervienen otras tecnologías como ser *WSDL* (**W**eb **S**ervices **D**escription **L**anguage) [58] y *UDDI* (**U**niversal **D**escription, **D**iscovery and **I**ntegration) [59].

Bibliografía

[1] D. Winer. *"XML-RPC Specification".* UserLand Software, Inc. http://www.xmlrpc.com/spec. Julio 2002.

[2] D.Box, D.Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S.Thatte, D. Winer. *"Simple Object Access Protocol (SOAP) 1.1"*. World Wide Web Consortium. http://www.w3.org/TR/SOAP/. Julio 2002.

[3] M. Gudgin, M. Hadley, J. Moreau, H. Nielsen. *"SOAP version 1.2 Working Draft"*. World Wide Web Consortium. http://www.w3.org/TR/2001/WD-soap12-20010709/. Julio 2002.

[4] N. Mitra. *"SOAP Version 1.2 Part 0: Primer".* World Wide Web Consortium. http://www.w3.org/TR/2001/WD-soap12-part0-20011217/. Julio 2002.

M. Gudgin, M. Hadley, J. Moreau, H. Nielsen. *"SOAP Version 1.2 Part 1: Messaging Framework".* World Wide Web Consortium. http://www.w3.org/TR/2001/WD-soap12-part1-20011217/. Julio 2002.

M. Gudgin, M. Hadley, J. Moreau, H. Nielsen. *"SOAP Version 1.2 Part 2: Adjuncts".* World Wide Web Consortium. http://www.w3.org/TR/2001/WD-soap12-part2-20011217/. Julio 2002.

[5] S. Graham, S. Simenov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, R. Neyam. *"Building Web Services with Java: making sense of XML, SOAP, WSDL and UUDI".* Sams, ISBN: 0-672-32181-5, Diciembre 2001.

[6] T.Berners-Lee,R. Fieding, L. Masinter. *"IETF RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax".*Internet Engineering Task Force. http://www.ietf.org/rfc/rfc2396.txt. Julio 2002.

[7] I. Nadrog. "*Simplified Object Access Protocol".* New Jersey Institute of Technology. http://eies.njit.edu/~turoff/coursenotes/CIS679/sample/soap.doc. Julio 2002.

[8] K. Brown. *"SOAP for Platform-Neutral interoperability".* http://www.fawcette.com/Archives/premier/mgznarch/xml/2000/04fal00/kb0004/kb0004.asp. Julio 2002.

[9] A. Skonnard. *"SOAP: The Simple Object Access Protocol".* http://www.microsoft.com/Mind/0100/soap/soap.asp. Julio 2002.

[10] P. van Dyk. *"SOAP: Coming Clean with XML".* http://sa.internet.com/technews/01/03/03a.htm. Julio 2002.

[11] B. Parikh. *"Research paper: SOAP-XML RPC Protocol".* http://parikhfamily.virtualave.net/soap.htm. Julio 2002.

[12] K. Scribner, M. Stiver. *"Understanding the simple Object Access Protocol (SOAP)".* http://webdeveloper.earthweb.com/webxml/article/0,,12015_641321,00.htm. Julio 2002.

[13] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler. *"Extensible Markup Language (XML) 1.0 (Second Edition)".* World Wide Web Consortium. http://www.w3.org/TR/2000/REC-xml-20001006. Julio 2002.

[14] International Organization for Standardization. *"ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML)".* International Organization for Standardization. Octubre 1986.

[15] B. Bos. *"XML in 10 Points".* World Wide Web Consortium. http://www.w3.org/XML/1999/XML-in-10-points.html. Julio 2002.

[16] W3Schools. *"XML Tutorial".* http://www.w3schools.com/xml/default.asp. Julio 2002.

[17] T. Bray, D. Hollander, A. Layman. *"Namespaces in XML".* World Wide Web Consortium. http://www.w3.org/TR/REC-xml-names/. Julio 2002.

[18] D. Fallside. *"XML Schema Part 0: Primer".* World Wide Web Consortium. http://www.w3.org/TR/xmlschema-0/. Julio 2002.

[19] **"***XML Schema" W*orld Wide Web Consortium. http://www.w3.org/XML/Schema. Julio 2002.

[20] N. Walsh. *"Understanding XML Schemas".* http://www.xml.com/pub/a/1999/07/schemas/index.html?page=1. Julio 2002.

[21] T. Bray, J. Paoli, C. Sperberg-McQueen. *"Extensible Markup Language (XML) 1.0".* World Wide Web Consortium. http://www.w3.org/TR/1998/REC-xml-19980210. Julio 2002.

[22] S. Adler, A. Berglund , J. Caruso , S. Deach , T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell , J. Richman , S. Zilles. *"Extensible Stylesheet Language (XSL)".* World Wide Web Consortium. http://www.w3.org/TR/WD-xsl/. Julio 2002.

[23] J. Robie, J. Lapp, D. Schach *"XML query Language (XQL)".* World Wide Web Consortium. http://www.w3.org/TandS/QL/QL98/pp/xql.html. Julio 2002.

[24] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. *"XML-QL: A Query Language for XML".* World Wide Web Consortium. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/. Julio 2002.

[25] S.Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener. *"The Lorel Query Language for Semistructured Data",* International Journal on Digital Libraries 1(1), páginas 68-88, Abril 1997.

[26] D. Chamberlain, J. Robie, D. Florescu. *"Quilt: An XML query Language for Heterogeneus Data Sources",* WebDB, páginas 53-62, 2000.

[27] J. Clark, S. DeRose. *"XML Path Language (XPath) version 1.0".* World Wide Web Consortium http://www.w3.org/TR/xpath. Julio 2002.

[28] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Siméon, M. Stefanescu. *"XQuery 1.0: An XML Query Language".* World Wide Web Consortium. http://www.w3.org/TR/xquery/. Julio 2002.

[29] X. Zhang. *"Rainbow: Relational Database Auto-Tunning for Efficient XML Query Processing".* Worcester Polytechnic Institute. http://davis.wpi.edu/dsrg/rainbow/publication/proposal.pdf. Julio 2002.

[30] A. Bonifati, S.Ceri. *"Comparative analysis of five XML query Language".* Dipartimento di Elettronica e Informazione, Politecnico de Milano. http://www.cs.auc.dk/~tbp/Teaching/DAT5E00/bonifati.pdf. Julio 2002.

[31] Microsoft Corporation, Hyperion Solution Corporation. *"XML for Analysis Specification version 1.0".* http://msdn.microsoft.com/library/en-us/dnxmlspec/html/xmlanalysis.asp. Julio 2002.

[32] IBM. *"DB2 XML Extender".* http://www-4.ibm.com/software/data/db2/extenders/xmlext/. Julio 2002.

[33] Oracle. *"Oracle9i Application Developer's Guide – XML Release 1 (9.0.1)".* http://download-east.oracle.com/otndoc/oracle9i/901_doc/appdev.901/a88894/toc.htm. Julio 2002.

[34] Microsoft TechNet. "SQL Server 2000 XML Overview". http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/sql/evaluate/featfunc/xmlsql.asp. Julio 2002.

[35] Microsoft MSDN. *"Creating XML views Using Annotated XDR Schema"* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsql/ac_mschema_5cfn.asp. Julio 2002.

[36] Microsoft Product Support Services. *"HOWTO: Retrieve XML Data by Using an XPath Query in Visual Basic Client".* http://support.microsoft.com/support/kb/articles/Q271/6/19.ASP. Julio 2002.

[37] M. Fernández, W. Tan, D. Suciu. *"Silkroute: Trading between Relations and XML".* http://db.cis.upenn.edu/DL/rxl/rxl.html. Julio 2002.

[38] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk. *"Querying XML Views of Relational Data".*http://www.cs.cornell.edu/People/jai/papers/QueryingXMLViews.pdf. Julio 2002.

[39] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, S. Subramanian. *"XPERANTO Publishing Object-Relational Data as XML".* http://www.cs.cornell.edu/People/jai/papers/XperantoOverview.pdf. Julio 2002.

[40] M. Yoshikawa, T. Amagasa, T. Shimura, S. Uemura *"Xrel: A Path-based approach to storage and retrieval of XML documents using relational databases",* ACM Transaction on Internet Technology 1(1), páginas 110-141, Agosto 2001.

[41] John White. *"The SOAP solution".* http://www.developernews.net/991112jw.html. Julio 2002.

[42] V. Turau. *"Making legacy data accesible for XML applications".* Department of Computer Science, University of Applied Sciences. http://www-1.informatik.fh-wiesbaden.de/~turau/ps/legacy.pdf. Julio 2002.

[43] Z. Z. Tun, A. Godchild, L. Bird, H. Sue. *"Exporting SQL resultsets into XML".* Distributed System Technology Centre, University of Queensland. http://www.dstc.edu.au/Research/Projects/Titanium/papers/DataExport/DataExport.html. Julio 2002.

[44] R. Bourret. *"Defining XML views over relational data".* http://www.rpbourret.com/xml/XMLViews.htm. Julio 2002.

[45] C. Baru. *"Xviews: XML views of relational schema".* San Diego Supercomputer Center, University of California San Diego. http://www.npaci.edu/DICE/Pubs/sdsc-tr-1999-3.pdf. Julio 2002.

[46] Apache Software Foundation. "*Apache SOAP*". http://xml.apache.org/dist/soap/. Julio 2002.

[47] Apache Software Foundation. http://www.apache.org/. Julio 2002.

[48] D. Chamberlin, P. Fankhauser, M. Marchiori, J. Robie. *"XML Query Requirements W3C Working Draft".* World Wide Web Consortium. http://www.w3.org/TR/2000/WD-xmlquery-req-20000815. Julio 2002.

[49] J. Clark. *"XSL Transformation (XSLT) version 1.0".* World Wide Web Consortium. http://www.w3.org/TR/xslt. Julio 2002.

[50] P. Wadler. *"A formal semantics of patterns in XSLT".* Bell Labs, Lucent Technologies http://www.research.avayalabs.com/user/wadler/papers/xsl-semantics/xsl-semantics.pdf. Julio 2002.

[51] P. Mosses. *"Denotational Semantic".* Handbook of Theoretical Computer Science, vol. B: Formal models and semantics, The MIT Press/Elsevier. ISBN: 0-444-88074-7 (Elsevier), ISBN: 0-262-22039-3 (The MIT Press). 1990.

[52] Microsoft MSDN. *"Guidelines for using XPath queries".* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsql/ac_mschema_0dtf.asp. Julio 2002.

[53] Sun. "*Java 2 Platform, Standard Edition*". http://java.sun.com/j2se/1.4/. Julio 2002.

[54] Apache Software Foundation. "Apache Tomcat". http://jakarta.apache.org/tomcat/index.html. Julio 2002.

[55] Caucho Technology. "Resin". http://www.caucho.com/resin/. Julio 2002.

[56] Doug Tidwell. *"Web services – The Web's next revolution".* http://www6.software.ibm.com/developerworks/education/wsbasics/index.html. Julio 2002.

[57] Graham Glass. *"The Web services (r)evolution, Part 1".* http://www-106.ibm.com/developerworks/webservices/library/ws-peer1.html?dwzone=ws. Julio 2002.

[58] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. *"Web Services Description Language (WSDL) 1.1".* World Wide Web Consortium. http://www.w3.org/TR/wsdl. Julio 2002.

[59] Universal Description, Discovery and Integration Project. *"UDDI".* http://uddi.org/specification.html. Julio 2002.

Anexo 1 *API* Núcleo del Sistema (ANS)

8.1 Introducción

Esta especificación define dos métodos, **Descubrir** y **Ejecutar**, los cuales reciben y devuelven datos en formato *XML*, permitiendo el descubrimiento y la consulta de datos relacionales a aplicaciones cliente.

8.2 Descubrir

Este método de *ANS* es usado por la aplicación cliente para obtener información. La información que se obtiene puede incluir una lista de las bases de datos disponibles, una lista de las vistas creadas para cada base de datos, los *XML Schemas* que especifican estas vistas e información específica para acceder a los datos, por ejemplo `password` y usuario. La información obtenida por la aplicación cliente depende de los valores de los parámetros con que se invoca al método.

8.2.1 Sintaxis

```
Descubrir (
        [in]   Requerimiento  TipoEnumString,
        [in]   Propiedades    TipoPropiedades,
        [out]  Resultado      TipoResultDesc
       )
```

8.2.2 Parámetros

8.2.2.1 Requerimiento

Este parámetro es requerido y consiste en un valor de tipo enumerado (*TipoEnumString*). Junto con el parámetro *Propiedades*, determinan qué información es retornada en el parámetro *Resultado* y su estructura. En la sección *ANS TipoResultDesc* (8.5) se detalla la estructura y contenido de los posibles conjuntos de datos retornados en el parámetro *Resultado*.

Los siguientes son los valores enumerados que son soportados:

Valor enumerado	Descripción
DESCUBRIR_BASES	Retorna una lista de las bases de datos disponibles.
DESCUBRIR_VISTAS	Retorna una lista de las vistas, relacionales[24] y/o *XML* creadas para una base de datos determinada.
DESCUBRIR_PROPIEDADES	Retorna una lista de las propiedades soportadas por *ANS*.
DESCUBRIR_ENUMERADOS	Retorna una lista con los valores enumerados soportados por *ANS*.
DESCUBRIR_SCHEMA_VISTA	Retorna el *XML Schema* que especifica la vista *XML*, para una determinada vista de una determinada base de datos.
DESCUBRIR_SCHEMA_INDEP	Retorna el *XML Schema* que especifica un formato para obtener resultados que es independiente de la especificación de la vista particular sobre la cual se realizan las consultas.

[24] Una vista relacional particular es la totalidad de la base de datos.

8.2.2.2 Propiedades

Este parámetro, de tipo *TipoPropiedades*, consiste en una colección de propiedades. Cada propiedad especifica un dato que es usado por el método **Descubrir**, como ser nombre de la base de datos, tipo de las vistas (*XML* o relacional), nombre de la vista, usuario y `password`.
Las propiedades que se deben incluir en este parámetro dependen del valor especificado para el parámetro *Requerimiento* y ambos parámetros determinan la información que es retornada en el parámetro *Resultado* y su estructura. En la sección *ANS TipoResultDesc* (8.5) se detalla la estructura y contenido de los posibles conjuntos de datos retornados en el parámetro *Resultado*.

Las propiedades disponibles y sus valores pueden ser obtenidas usando el valor DESCUBRIR_PROPIEDADES del parámetro *Requerimiento*. Las propiedades se detallan en la sección *ANS* Propiedades (8.6).

No se requiere un orden determinado en las propiedades listadas en este parámetro. Este parámetro es requerido pero puede ser vacío.

8.2.2.3 Resultado

Este parámetro contiene el resultado de la invocación al método **Descubrir**. La estructura y contenido de este parámetro está determinado por los valores de los parámetros *Requerimiento* y *Propiedades*.
Las posibles estructuras y contenidos se detallan en la sección *ANS TipoResultDesc* (8.5)

8.2.3 Ejemplo

En este ejemplo se muestra la invocación al método **Descubrir** realizada por un cliente, usando *SOAP*, para obtener las vistas[25] *XML* disponibles para la base de datos de nombre *MiBase*:

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 587
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:Descubrir
      xmlns:ns1="urn:xpath"
      SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
      <Requerimiento xsi:type="xsd:string"
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         DESCUBRIR_VISTAS
      </Requerimiento>
      <Propiedades>
       <ListaPropiedad>
        <BaseDatos>MiBase</BaseDatos>
       </ListaPropiedad>
      </Propiedades>
    </ns1:Descubrir>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 62 – Ejemplo de invocación (SOAP) al método Descubrir para obtener las vistas disponibles

[25] Por defecto se retornan las vistas *XML*.

ANS retorna a la aplicación cliente el siguiente resultado:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 764
<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
   <ns1:DescubrirResponse
     xmlns:ns1="urn:xpath"
     SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
    <return>
     <Root>
      <!— Definicion del schema del resultado -->
      ...
       <BaseDatos>MiBase
       <Vista>VistaA
          <Descripcion>Descripcion de la vista A</Descripcion>
        </Vista>
       <Vista>VistaB
          <Descripcion>Descripcion de la vista B</Descripcion>
       </Vista>
      </BaseDatos>
     </Root>
    </return>
   </ns1:DescubrirResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 63 - Ejemplo de respuesta (SOAP) con las vistas disponibles

8.3 Ejecutar

Este método es usado para que las aplicaciones cliente realicen consultas.

8.3.1 Sintaxis

Execute (
[in] *Sentencia* TipoSentencia,
[in] *Propiedades* TipoPropiedades,
[out] *Resultado* TipoResultEjec
)

8.3.2 Parámetros

8.3.2.1 Sentencia

Este parámetro es requerido y es de tipo *TipoSentencia*. Consiste en una sentencia que contiene la consulta a la base de datos relacional expresada en un lenguaje de consulta de *XML* (sección 2.5).

8.3.2.2 Propiedades

Este parámetro, de tipo *TipoPropiedades*, consiste en una colección de propiedades. Cada propiedad especifica un dato que es usado por el método **Ejecutar**, como ser nombre de la base de datos, nombre de la vista, formato del resultado, usuario y `password`.

Las propiedades disponibles y sus valores pueden ser obtenidas usando el valor DESCUBRIR_PROPIEDADES del parámetro *Requerimiento*. Las propiedades se detallan en la sección *ANS* Propiedades (8.6).

No se requiere un orden determinado en las propiedades listadas en este parámetro. Este parámetro es requerido.

8.3.2.3 Resultado

Este parámetro es de tipo *TipoResultEjec* y contiene el resultado de la invocación al método **Ejecutar**. La estructura del parámetro *Resultado* depende de si la consulta se realiza sobre una vista relacional o *XML*. En el caso de una vista *XML* depende, también, del valor de la propiedad *Formato*, existiendo dos posibilidades, si no se incluye esta propiedad el formato por defecto corresponde al del *XML Schema*, obtenido mediante la invocación al método **Descubrir** con el valor DESCUBRIR_SCHEMA_VISTA, es decir que el resultado se retorna en un documento *XML* con la misma estructura que la vista.

Las posibles estructuras y contenidos se detallan en las secciones *ANS TipoResultDesc* (8.5) y 8.4.4

8.3.3 Ejemplo

En este ejemplo se muestra la invocación al método **Ejecutar** realizada por un cliente, usando *SOAP*, para realizar una consulta expresada en $XPath_{Sub}$ sobre la vista *XML VistaA* de la base de datos *MiBase*:

```
POST /apache_soap22/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml;
charset=utf-8
Content-Length: 619
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
    <ns1:Ejecutar
       xmlns:ns1="urn:xpath"
       SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
       <Sentencia>
        <Consulta>
           /fabricantes[@numero_fabricante=1]/venta[@precio=200]
        </Consulta>
       </Sentencia>
       <Propiedades>
        <ListaPropiedad>
          <BaseDatos>MiBase</BaseDatos>
          <Vista>VistaA</Vista>
          <Formato>SCHEMA_INDEP</Formato>
        </ListaPropiedad>
       </Propiedades>
    </ns1:Ejecutar>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 64 – Ejemplo de invocación (SOAP) al método Ejecutar para realizar una consulta

ANS retorna al cliente el siguiente resultado:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 1000
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:EjecutarResponse
         xmlns:ns1="urn:xpath"
         SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
         <return>
          <Root>
            <Row>
              <Column nombre="precio">200</Column>
              <Column nombre="fecha_venta">a</Column>
              <Column nombre="fecha_envio">c</Column>
              <Column nombre="numero_producto">X2</Column>
              <Column nombre="descripcion">esponja</Column>
            </Row>
            <Row>
              <Column nombre="precio">200</Column>
              <Column nombre="fecha_venta">z</Column>
              <Column nombre="fecha_envio">z</Column>
              <Column nombre="numero_producto">X2</Column>
              <Column nombre="descripcion">esponja</Column>
            </Row>
          </Root>
         </return>
      </ns1:EjecutarResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 65 - Ejemplo de respuesta (SOAP) con los datos obtenidos por la consulta

8.4 Tipos de Datos de la API

En esta sección se describen los tipos de datos usados en *ANS* y su relación con *XML*.

8.4.1 TipoSentencia

Este tipo de datos es un elemento *XML* de etiqueta <Consulta> y contenido de tipo string. Por ejemplo:

```
<Consulta>
   /fabricante/venta
</Consulta>
```

8.4.2 TipoEnumString

Este tipo define un conjunto de constantes simbólicas. Para cada constante se utiliza el tipo string estándar de *XML*.

8.4.3 TipoPropiedades

Este tipo de datos es un elemento *XML* de etiqueta <ListaPropiedad> y representa una colección de propiedades. Cada propiedad está definida por un elemento *XML* y el valor de la propiedad es el dato contenido por el elemento *XML*. El nombre del elemento *XML* se corresponde con el nombre de la propiedad. Por ejemplo:

```
<ListaPropiedad>
   <BaseDatos>MiBase</BaseDatos>
   <Vista>Vista1</Vista>
</ListaPropiedad>
```

8.4.4 TipoResultEjec

Este tipo de datos es un documento *XML* cuya estructura depende del tipo de vista y además, en el caso de vistas XML, del valor de la propiedad *Formato*, con la cual se invoca al método **Ejecutar**.
Existen dos valores posibles para la propiedad *Formato* que se detallan a continuación:

Valor Formato	Descripción
SCHEMA_VISTA	La estructura del documento *XML* retornado con el resultado de la consulta está definido por el *XML Schema* que se obtiene mediante la invocación al método **Descubrir** con el valor DESCUBRIR_SCHEMA_VISTA.
SCHEMA_INDEP	La estructura del documento *XML* retornado con el resultado de la consulta está definido por el *XML Schema* que se obtiene mediante la invocación al método **Descubrir** con el valor DESCUBRIR_SCHEMA_INDEP. En la sección *API TipoResultDesc* (8.5) se detalla este *XML Schema*.

En el caso de vistas relacionales la estructura es la misma que la que corresponde al valor SCHEMA_INDEP para las vistas *XML*.

8.4.5 TipoResultDesc

Este tipo de datos es un documento *XML* cuya estructura depende de los valores especificados en los parámetros *Requerimiento* y *Propiedades* del método **Descubrir**. En la sección *ANS TipoResultDesc* (8.5) se detallan las posibles estructuras.

8.4.6 String

El tipo `string` se corresponde al tipo estándar `string` de *XML*.

8.5 ANS TipoResultDesc

La información retornada en el parámetro *Resultado* del método **Descubrir** se estructura de acuerdo a lo que se especifica continuación.
Para cada estructura posible se incluye una tabla que provee la siguiente información para cada elemento:

- Nombre Elemento nombre del elemento en el resultado
- Tipo tipo del elemento
- Descripción descripción del propósito del elemento
- Admite Nulos indica si el dato del elemento deber ser retornado o no

8.5.1 DESCUBRIR_BASES

Cuando el método **Descubrir** es invocado con el valor DESCUBRIR_BASES en el parámetro *Requerimiento*, el valor del parámetro *Propiedades* no es tenido en cuenta.
Se retorna en el parámetro *Resultado* un documento *XML* que contiene una lista de las bases de datos disponibles, con la siguiente estructura:

Nombre Elemento	Tipo	Descripción	Admite nulos
BaseDatos	String	Nombre de la base de datos.	No
Descripcion	String	Descripción de la base de datos.	Sí
ModoAutenticacion	TipoEnumString	Enumerado que indica el modo de seguridad de la base de datos. Los valores posibles son: • *NoRequerido*: no es necesario enviar un usuario y password. • *Requerido*: Se requiere un usuario y password.	No

Ejemplo:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 764
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:DescubrirResponse
         xmlns:ns1="urn:xpath"
         SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
         <return>
          <Root>
            <!— Definicion del schema del resultado -->
            ...
            <BaseDatos>MiBase
              <Descripcion>Descripcion de MiBase</Descripcion>
              <ModoAutenticacion>NoRequerido</ModoAutenticacion>
            </BaseDatos>
            <BaseDatos>OtraBase
               <Descripcion>Descripcion de OtraBase</Descripcion>
              <ModoAutenticacion>NoRequerido</ModoAutenticacion>
            </BaseDatos>
          </Root>
        </return>
      </ns1:DescubrirResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 66 – Ejemplo de invocación (SOAP) al método Descubrir para obtener las bases de datos

8.5.2 DESCUBRIR_VISTAS

Cuando el método **Descubrir** es invocado con el valor DESCUBRIR_VISTAS en el parámetro *Requerimiento*, solamente se tienen en cuenta las propiedades *BaseDatos* y *Tipo* del parámetro *Propiedades* (eventualmente pueden tenerse en cuenta los valores de las propiedades *Password* y *Usuario* si son necesarios).
Se retorna en el parámetro *Resultado* un documento *XML* conteniendo una lista de las vistas creadas de la base de datos que corresponden al tipo indicado, con la siguiente estructura:

Nombre Elemento	Tipo	Descripción	Admite nulos
BaseDatos	String	Nombre de la base de datos.	No
Vista	String	Nombre de la vista.	No
Descripcion	String	Descripción de la vista.	Si

Ejemplo:

```
HTTP/1.0 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 764
<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
   <ns1:DescubrirResponse
     xmlns:ns1="urn:xpath"
     SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
    <return>
     <Root>
      <!— Definicion del schema del resultado -->
      ...
       <BaseDatos>MiBase
       <Vista>VistaA
          <Descripcion>Descripcion de la vista A</Descripcion>
        </Vista>
       <Vista>VistaB
          <Descripcion>Descripcion de la vista B</Descripcion>
       </Vista>
      </BaseDatos>
     </Root>
    </return>
   </ns1:DescubrirResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 67 – Ejemplo de invocación (SOAP) al método Descubrir para obtener las vistas

8.5.3 DESCUBRIR_PROPIEDADES

Cuando el método **Descubrir** es invocado con el valor DESCUBRIR_PROPIEDADES en el parámetro *Requerimiento*, el valor del parámetro *Propiedades* no es tenido en cuenta.
Se retorna en el parámetro *Resultado* un documento *XML* que contiene una lista de las propiedades soportadas por *ANS*, con la siguiente estructura:

Nombre Elemento	Tipo	Descripción	Admite nulos
NombrePropiedad	String	Nombre de la propiedad.	No
DescripcionPropiedad	String	Descripción de la propiedad.	Sí
TipoPropiedad	String	El tipo *XML* de la propiedad.	Si
ValorDefectoPropiedad	String	El valor por defecto de la propiedad	Si

8.5.4 DESCUBRIR_ENUMERADOS

Cuando el método **Descubrir** es invocado con el valor DESCUBRIR_ENUMERADOS en el parámetro *Requerimiento*, el valor del parámetro *Propiedades* no es tenido en cuenta.
Se retorna en el parámetro *Resultado* un documento *XML* que contiene una lista de los enumerados soportados por *ANS*, con la siguiente estructura:

Nombre Elemento	Tipo	Descripción	Admite nulos
NombreEnum	String	Nombre del enumerado.	No
DescripcionEnum	String	Descripción del enumerado.	Si
TipoEnum	String	El tipo de los valores enumerados	No
NombreElemento	String	El nombre de uno de los valores enumerados. Por ejemplo: NoRequerida ó DESCUBRIR_PROPIEDADES ó SCHEMA_INDEP	No
DescripcionElemento	String	Descripción del elemento	Si

Para cada enumerado soportado por *ANS*, existen tantos elementos como valores en la enumeración.

8.5.5 DESCUBRIR_SCHEMA_VISTA

Cuando el método **Descubrir** es invocado con el valor DESCUBRIR_SCHEMA_VISTA en el parámetro *Requerimiento*, solamente se tienen en cuenta las propiedades *BaseDatos* y *Vista* del parámetro *Propiedades* (eventualmente pueden tenerse en cuenta los valores de las propiedades `Password` y *Usuario* si son necesarios para conectarse a la base de datos).
Se retorna en el parámetro *Resultado* un documento *XML* que es el *XML Schema* que especifica la vista de la base de datos.

En el caso de ser una vista relacional se especifica la información necesaria para la realización de consultas, como ser claves primarias, claves foráneas, tablas, atributos de cada tabla, etc.

En el caso de ser una vista *XML*, el *XML Schema* es conforme a lo descripto en la sección 3.5.

8.5.6 DESCUBRIR_SCHEMA_INDEP

Cuando el método **Descubrir** es invocado con el valor DESCUBRIR_SCHEMA_INDEP en el parámetro *Requerimiento*, el valor del parámetro *Propiedades* no es tenido en cuenta.
Se retorna en el parámetro *Resultado* un documento *XML* que es el *XML Schema* que especifica un formato del parámetro *Resultado* retornado por el método **Ejecutar** que es independiente de la especificación (*XML Schema*) de la vista particular sobre la cual se realizan las consultas.
El *XML Schema* es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name = "Root">
  <xs:complexType contenent="elementOnly">
   <xs:element name = "row" minOccurs="0" maxOccurs ="unbounded">
    <xs:complexType contenent="elementOnly">
     <xs:element name="column" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType>
       <xs:attribute name="nombre" type="xs:string"/>
      </xs:complexType>
     </xs:element>
    </xs:complexType>
   </xs:element>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figura 68 – XML Schema independiente de la vista

El *XML Schema* [43] anterior provee una estructura común para todos los posibles `resultset` que se obtienen al realizar consultas *SQL* a la base de datos.
Cada tupla del `resultset` se mapea a un elemento (`row`) y las columnas de la tupla se mapean a elementos anidados (`column`) del elemento que representa la tupla.
Al utilizar nombres genéricos y predefinidos para las `tag` de los elementos (`row` y `column`) se obtiene independencia de los nombres particulares de cada `resultset`.

Los valores de las columnas de cada tupla aparecen como texto asociado al elemento `column` y el nombre de la columna en la tupla aparece como atributo del elemento. El valor de este atributo coincide con el nombre del atributo en el *XML Schema* que especifica la vista *XML* sobre la cual se realiza la consulta.

Devolver los resultados de las consultas a la base de datos en un documento *XML* acorde al *XML Schema* anterior tiene la ventaja de que las aplicaciones cliente obtienen el resultado en un formato único e independiente de la vista particular sobre la cual realizaron la consulta.

8.6 ANS Propiedades

En esta sección se describen las propiedades soportadas por *ANS*, que son usadas por los métodos **Descubrir** y **Ejecutar**, en el parámetro *Propiedades*.

Nombre	Tipo	Valor por Defecto	Métodos que la utilizan	Descripción
BaseDatos	String	vacío	Descubrir Ejecutar	Especifica la base de datos. Esta propiedad es requerida, junto con la propiedad *Vista*, cuando el método **Descubrir** es invocado con el valor DESCUBRIR_VISTAS en el parámetro *Requerimiento*, para obtener la lista de vistas disponibles para una determinada base de datos. Esta propiedad es requerida cuando el método **Descubrir** es invocado con el valor DESCUBRIR_SCHEMA_VISTA en el parámetro *Requerimiento*, para obtener el *XML Schema* que especifica la vista *XML* de la base de datos. Esta propiedad es requerida siempre cuando se invoca el método **Ejecutar**.
Formato	TipoEnumString	SCHEMA_VISTA	Ejecutar	Enumerado que determina el formato del parámetro *Resultado* retornado por el método **Ejecutar** cuando se utilizan vistas *XML*. Los valores posibles son: • SCHEMA_VISTA. La estructura del documento *XML* retornado con el resultado de la consulta está definido por el *XML Schema* que se obtiene mediante la invocación al método **Descubrir** con el valor DESCUBRIR_SCHEMA_VISTA. • SCHEMA_INDEP La estructura del documento *XML* retornado con el resultado de la consulta está definido por el *XML Schema* que se obtiene mediante la invocación al método **Descubrir** con el valor DESCUBRIR_SCHEMA_INDEP.

Password	String	vacío	Descubrir Ejecutar	Es un `string` mediante el cual el cliente provee la `password` si esta es requerida.
Tipo	TipoEnumString	VISTAS_XML	Descubrir	Enumerado que determina el tipo de vistas retornadas por el método **Descubrir**. Los valores posibles son: • VISTAS_XML retorna una lista de las vistas XML de una base de datos • VISTAS_REL retorna una lista de las vistas relacionales de una base de datos.
Usuario	String	vacío	Descubrir Ejecutar	Es un `string` mediante el cual el cliente provee el usuario si este es requerido.
Vista	String	vacío	Descubrir Ejecutar	Especifica la vista de una base de datos. Esta propiedad es requerida, junto con la propiedad BaseDatos, cuando el método **Descubrir** es invocado con el valor DESCUBRIR_SCHEMA_VISTA en el parámetro *Requerimiento*, para obtener el *XML Schema* que especifica la vista *XML* de la base de datos. Esta propiedad es requerida siempre cuando se invoca el método **Ejecutar**.

Es posible agregar nuevas propiedades, para contemplar aspectos específicos de un manejador particular de base de datos relacional.

8.7 Manejo de errores

Los errores son manejados de manera diferente, dependiendo de su tipo. Los siguientes son los tipos de error que pueden ocurrir:

- Error al ejecutar la invocación a un método. Esto es reportado directamente por el protocolo *SOAP* a través de un mensaje `Fault`. Cuando esto ocurre el parámetro *Resultado* no es devuelto.
- Errores o advertencias durante la ejecución de un método. Cuando un método se ejecuta, pero se producen errores o advertencias durante su ejecución, los mismos son retornados a la aplicación cliente en el parámetro *Resultado*. Por ejemplo:

```
<Error>
   <CodigoError>2148497527</CodigoError>
   <Descripcion>Error de seguridad</Descripcion>
</Error>
```

Anexo 2 – Algoritmo, Funciones y Tipos de Datos auxiliares

9.1 Algoritmo

$\mathcal{XPath}_{Sub}ToAlg$: Camino × Mapeo × Vista → String

```
XPath_SubToAlg [XPath_Sub](M,V) =
   let X = X_ALG[XPath_Sub](∅,0,M,V)
   if EsError(X) then
    return "Error"
   else
    let Exp = Ult(SelSec(X))
    let Ult_Paso = Ult(XPath_Sub)
    let D = Descendientes(ult_paso,V)
    if D = ∅ then
       return Exp
    else
       let a = ∅
       foreach d_i in D
          let a = a ∪ AtributosVista(d_i,M,V)
          let Exp = App_Left* (Exp,Tabla(M,d_i))
       return AppΠ(Exp,a)
    end if
   end if
```

X_{ALG}: Camino × SecExprAlg x Entero x Mapeo × Vista → ResX_{ALG}

$X_{ALG}[/](S,N,M,V) = Cons(Ins(\emptyset,Oj(M,V)),1,false)$

$X_{ALG}[/cc](S,N,M,V) = X_{cc\_ALG}[cc](Ins(\emptyset,Oj(M,V)),1,M,V)$

$X_{ALG}[/ca](S,N,M,V) = X_{ca\_ALG}[ca](Ins(\emptyset,Oj(M,V)),1,M,V)$

X_{cc_ALG}: CaminoComun × SecExprAlg x Entero x Mapeo × Vista → ResX_{ALG}

```
X_cc_ALG[cc/n](S,N,M,V)=
     let X = X_cc_ALG[cc](S,N,M,V)
     if EsError(X) then
        return Cons(⊥,⊥,true)
     else
        let S1 = SelSec(X)
        let N1 = SelNiv(X)
        let E1 = Ult(S1)
        let E2 = Tabla(M,n)
        if IgualEsquema(E1,Oj(M,V)) then
           return Cons(Ins(S1,AppΠ(App*(E1,E2),Atributos(M,n))),N1+1,false)
        else
           return Cons(Ins(S1,App*(E1,E2)),N1+1,false)
        end if
    end if
```

```
X_cc_ALG[cc/..](S,N,M,V)=
      let X = X_cc_ALG[cc](S,N,M,V)
      if EsError(X) then
       return Cons(⊥,⊥,true)
      else
       if SelNiv(X) = 1 then
           return Cons(⊥,⊥,true)
       else
         let S1 = SelSec(X)
         let N1 = SelNiv(X)
         let E1 = Ult(S1)
         let E2 = Obt(S1,N1-1)
         if IgualEsquema(E2,Oj(M,V)) then
            return Cons(Ins(S1,App*(E1,E2)),N1-1,false)
         else
            return Cons(Ins(S1,AppΠ(E1,Atr(E2)),N1-1,false)
         end if
       end if
      end if

X_cc_ALG[cc p](S,N,M,V) =
      let X = X_cc_ALG[cc](S,N,M,V)
      if EsError(X) then
       return Cons(⊥,⊥,true)
      else
         let S1 = SelSec(X)
         let S2 = ElimUlt(S1)
         let E1 = Ult(S1)
         let N1 = SelNiv(X)
         let P = P_ALG[p](S1,N1,M,V)
         if EsError(P) then
          return Cons(⊥,true)
         else
          let E2 = SelExpr(P)
          return Cons(Ins(S2,App*(E1,E2)),N1,false)
       end if
      end if

X_cc_ALG[n](S,N,M,V)=
      if Vacia(S) then
         return Cons(⊥,⊥,true)
      else
         let E1 = Ult(S)
         let E2 = Tabla(M,n)
         if IgualEsquema(E1,Oj(M,V)) then
          return Cons(Ins(S,AppΠ(App*(E1,E2),Atributos(M,n))),N+1,false)
         else
          return Cons(Ins(S,App*(E1,E2)),N+1,false)
         end if
      end if
```

```
X_cc_ALG[..](S,N,M,V) =
      if N = 1 then
         return Cons(⊥,⊥,true)
      else
       if Vacia(S) then
          return Cons(⊥,⊥,true)
       else
        let E1 = Ult(S)
        let E2 = Obt(S,N-1)
        if IgualEsquema(E2,Oj(M,V)) then
           return Cons(Ins(Sec,App*(E1,E2)),N-1,false)
        else
           return Cons(Ins(Sec,AppΠ(E1,Atr(E2)),N-1,false)
        end if
       end if
      end if
```

X_{ca_ALG}: CaminoAtributo × SecExprAlg x Entero x Mapeo × Vista → ResX_{ALG}

```
X_ca_ALG[cc/@n](S,N,M,V)=
    let X = X_cc_ALG[cc](S,N,M,V)
    if EsError(X) then
     return Cons(⊥,⊥,true)
    else
     let S1 = SelSec(X)
     let N1 = SelNiv(X)
     let E = Ult(S1)
     return Cons(Ins(S1,AppΠ(E,Atributos(M,n))),N1+1,false)
   end if
```

P_{ALG}: Predicado × SecExprAlg x Entero x Mapeo × Vista → ResP_{ALG}

```
P_ALG[not p](S,N,M,V) =
    let E = Ult(S)
    let P = P_ALG[p](S,N,M,V)
    if ¬ EsError(P) then
     return Cons(App-(E,SelExpr(P)),false)
    else
     return Cons(⊥,true)
    end if

P_ALG[@n](S,N,M,V) =
    let E = Ult(S)
    let c = GenCond(Atributos(M,n),{"is not null"},"=")
    return Cons(Appσ(E,c),false)

P_ALG[cc](S,N,M,V) =
    let X = X_cc_ALG[cc](S,N,M,V)
    if ¬ EsError(X) then
     let S1 = SelSec(X)
     let E = Ult(S)
     let R = Generar(S1,S,E)
     return Cons(ResolverCam(R),false)
    else
     return Cons(⊥,true)
    end if
```

```
P_ALG[ca](S,N,M,V) =
    let X = X_ca_ALG[ca](S,N,M,V)
    if ¬ EsError(X) then
       let S1 = SelSec(X)
       let E = Ult(S)
       let R = Generar(S1,S,E)
       let c = GenCond(Atr(E),{"is not null"},"=")
       let Q = Appσ(R,c)
       return Cons(ResolverCam(Q),false)
    else
       return Cons(⊥,true)
    end if

P_ALG[p1 and p2](S,N,M,V)=
     let P1 = P_ALG[p1](S,N,M,V)
     let P2 = P_ALG[p2](S,N,M,V)
     if ¬ EsError(P1) ∧ ¬ EsError(P2)  then
        return Cons(App∩(SelExpr(P1),SelExpr(P2)),false)
     else
        return Cons(⊥,true)
     end if

P_ALG[p1 or p2](S,N,M,V) =
     let P1 = P_ALG[p1](S,N,M,V)
     let P2 = P_ALG[p2](S,N,M,V)
     if ¬ EsError(P1) ∧ ¬ EsError(P2)  then
        return Cons(App∪(SelExpr(P1),SelExpr(P2)),false)
     else
        return Cons(⊥,true)
     end if
```

```
P_ALG[o1 op o2](S,N,M,V) =
   let E = Ult(S)
   if EsAtributo(o1) ∧ EsAtributo(o2) then
      return ResolverAtr_Atr(M,E,o1,o2,op)
   else
      if EsAtributo(o1) ∧ EsValor(o2) then
         return ResolverAtr_Val(M,E,o1,o2,op)
      else
         if EsAtributo(o1) ∧ EsCaminoAtributo(o2) then
          return ResolverAtr_Cam(M,S,E,o1,o2,op,N)
         else
            if EsValor(o1) ∧ EsAtributo(o2) then
               return P_ALG[o2 NuevoOp(op) o1](S,N,M,V)
            else
               if EsValor(o1) ∧ EsValor(o2) then
                  return ResolverVal_Val(M,E,o1,o2,op)
               else
                  if EsValor(o1) ∧  EsCaminoAtributo(o2) then
                     return ResolverVal_Cam(M,S,E,o1,o2,op,N)
                  else
                     if EsCaminoAtributo(o1) ∧ EsAtributo(o2) then
                        return P_ALG[o2 NuevoOp(op) o1](S,N,M,V)
                     else
                        if EsCaminoAtributo(o1) ∧ EsValor(o2) then
                           return P_ALG[o2 NuevoOp(op) o1](S,N,M,V)
                        else
                           if EsCaminoAtributo(o1) ∧ EsCaminoAtributo(o2) then
                              return ResolverCam_Cam(M,S,E,o1,o2,op,N)
                           end if
                        end if
                     end if
                   end if
                end if
             end if
          end if
       end if
    end if
```

O_{ALG}: Operando × SecExprAlg x Entero x Mapeo × Vista → $ResO_{ALG}$

```
O_ALG[ca](S,N,M,V) =
     let X = X_ca_ALG[ca](S,N,M,V)
     if ¬ EsError(X) then
        let S1 = SelSec(X)
        let E = Ult(S1)
        return Cons(S1,Atr(E),false)
     else
        return Cons(⊥,⊥,true)
     end if
```

9.2 Funciones sobre TAD SecExprAlg

1. `Elim: SecExprAlg x SecExprAlg→ SecExprAlg`
   `Elim(`S_1`,`S_2`)` retorna la secuencia resultado de eliminar de la secuencia S_1 la secuencia S_2.

2. `ElimUlt: SecExprAlg → SecExprAlg`
   `ElimUlt(S)` retorna la secuencia de expresiones algebraicas S sin la última expresión algebraica.

3. `Ins: SecExprAlg x string → SecExprAlg`
   `Ins(S,E)` inserta la expresión algebraica `E` al final de la secuencia `S`.

4. `Obt: SecExpralg x entero → string`
   `Obt(S,pos)` retorna la expresión algebraica que ocupa la posición `pos` en la secuencia S.

5. `Primera : SecExprAlg → string`
   `Primera(S)` retorna la primer expresión algebraica de la secuencia S.

6. `Resto: SecExprAlg → SecExprAlg`
   `Resto(S)` retorna la secuencia de expresiones algebraicas S sin la primer expresión algebraica.

7. `Ult: SecExprAlg → string`
   `Ult(S)` retorna la última expresión algebraica de la secuencia S.

8. `Vacia: SecExprAlg → Boolean`
   `Vacia(S)` retorna verdadero si la secuencia S es vacía y falso en caso contrario.

9.3 Funciones sobre TAD ResX_{ALG}

1. `Cons : SecExprAlg x Entero x Boolean →` ResX_{ALG}

2. `SelSec:` ResX_{ALG} `→ SecExprAlg`
   `SelSec(Cons(s,e,b)) = s`

3. `SelNiv:` ResX_{ALG} `→ Entero`
   `SelNiv(Cons(s,e,b)) = e`

4. `EsError:` ResX_{ALG} `→ Boolean`
   `EsError(Cons(s,e,b)) = b`

9.4 Funciones sobre TAD ResP_{ALG}

1. `Cons : string x Boolean →` ResP_{ALG}

2. `SelExpr:` ResP_{ALG} `→ string`
   `SelExpr(Cons(s,b)) = s`

3. `EsError:` ResP_{ALG} `→ Boolean`
   `EsError(Cons(s,b)) = b`

9.5 Funciones sobre TAD $ResO_{ALG}$

1. Cons : SecExprAlg x Conjunto(string) x Boolean→ $ResO_{ALG}$

2. SelSec: $ResO_{ALG}$ → SecExprAlg
   SelSec(Cons(s,c,b)) = s

3. SelAtr: $ResO_{ALG}$ → Conjunto(string)
   SelAtr(Cons(s,c,b = c

4. EsError: $ResO_{ALG}$ → Boolean
   EsError(Cons(s,c,b)) = b

9.6 Funciones sobre TAD ExprAlg

1. App* : string x string → string
   App*(E_1,E_2) retorna la expresión algebraica resultado de aplicar el join natural a las expresiones algebraicas E_1 y E_2 (E_1 * E_2).
   App*(E_1,E_2) = Concat(Concat(E_1,"*"),E_2)

2. AppΠ : string x string → string
   AppΠ(E,atrs) retorna la expresión algebraica resultado de proyectar la expresión algebraica E por las columnas indicadas en atrs ($\Pi_{atrs}(E)$).
   AppΠ(E,atrs)=
   ```
   Concat(
           Concat("Π",GenComas(atrs)),Concat(Concat("(",E),")")
   )
   ```

3. Appσ : string x string → string
   Appσ(E,cond) retorna la expresión algebraica resultado de aplicar una selección a la expresión algebraica E por la condición cond ($\sigma_{cond}(E)$).
   Appσ(E,cond)=Concat(Concat("σ",cond),Concat(Concat("(",E),")"))

4. Appx : string x string → string
   Appx(E_1,E_2) retorna la expresión algebraica resultado de aplicar el producto cartesiano a las expresiones algebraicas E_1 y E_2 (E_1 x E_2).
   Appx(E_1,E_2) = Concat(Concat(E_1,"x"),E_2)

5. App- : string x string → string
   App-(E_1,E_2) retorna la expresión algebraica que es la diferencia de las expresiones algebraicas E_1 y E_2 (E_1 - E_2).
   App-(E_1,E_2) = Concat(Concat(E_1,"-"),E_2)

6. App∩ : string x string → string
   App∩(E_1,E_2) retorna la expresión algebraica que es la intersección de las expresiones algebraicas E_1 y E_2 (E_1 ∩ E_2).
   App∩(E_1,E_2) = Concat(Concat(E_1,"∩"),E_2)

7. App∪ : string x string → string
   App∪(E_1,E_2) retorna la expresión algebraica que es la unión de las expresiones algebraicas E_1 y E_2 (E_1 ∪ E_2).
   App∪(E_1,E_2) = Concat(Concat(E_1,"∪"),E_2)

8. Appρ : string x string x string $\rightarrow$ string
   Appρ(E,NV,NN) retorna la expresión algebraica que es el resultado de renombrar, en la expresión algebraica E, las columnas indicadas en NV por las columnas indicadas en NN ($\rho_{NV \rightarrow NN}(E)$).

```
App$\rho$(E,NV,NN) =  Concat(
                        Concat("$\rho$",Concat(
                                    Concat(GenComas(NV),"$\rightarrow$"),
                                    GenComas(NN))
                                ),
                        Concat(Concat("(",E),")")
                    )
```

9. App$_{Left}$* : string x string $\rightarrow$ string
   App$_{Left}$* (E$_1$,E$_2$) retorna la expresión algebraica resultado de aplicar el left outer join natural a las expresiones algebraicas E$_1$ y E$_2$ (E$_1$ $_{Left}$* E$_2$)
   App$_{Left}$* (E$_1$,E$_2$) = Concat(Concat(E$_1$,"$_{Left}$*"),E$_2$)

10. Atr: string $\rightarrow$ Conjunto(string)
    Atr(E) retorna las columnas de la expresión algebraica E.

11. IgualEsquema: string x string $\rightarrow$ Boolean
    IgualEsquema(E$_1$,E$_2$) retorna verdadero si las columnas de las expresiones algebraicas E$_1$ y E$_2$ son las mismas y falso en caso contrario.
    IgualEsquema(E$_1$,E$_2$) = (Atr(E$_1$) = Atr(E$_2$))

12. ModifΠ : string x string $\rightarrow$ string
    ModifΠ(E,a) retorna la expresión algebraica resultado de modificar la proyección, de más afuera, de la expresión algebraica E (en caso que E involucre una proyección) de manera de realizarla por la unión de las columnas por las que ya se realizaba y el conjunto de columnas indicadas en a. Si E es de la forma AppΠ(X,atrs) retorna AppΠ(X,Concat(atrs,a)), sino retorna E.

13. Oj: Mapeo x Vista $\rightarrow$ string
    Oj(M,V) retorna la expresión algebraica correspondiente al left outer join natural de las tablas que corresponden a cada uno de los elementos de la vista *XML* V de la base de datos.

14. Sustituir : string x string x string $\rightarrow$ string
    Sustituir(E,E$_1$,E$_2$) sustituye en la expresión algebraica E, la expresión E$_1$ por la expresión E$_2$.

9.7 Funciones Generales

1. Atributos: Mapeo x string $\rightarrow$ Conjunto(string)
   Atributos(M,e) retorna el conjunto de nombres de columnas de la tabla correspondiente al elemento e en la base de datos relacional. Si e es un atributo retorna un conjunto formado por un solo nombre de columna que corresponde al atributo en el mapeo.

2. AtributosVista : Mapeo x Vista x string $\rightarrow$ Conjunto(string)
   AtributosVista(M,V,e) retorna un conjunto de nombres de columnas de la base de datos relacional que corresponden a los atributos del elemento *XML* e, declarados en la vista *XML* V.

3. Concat : string x string $\rightarrow$ string
   Concat(a,b) retorna la concatenación de a y b, poniendo b al final de a.

4. `CrearRel : valor x string x string → Relación`
   `CrearRel(v,atr,nom)` crea una relación de nombre `nom` que tiene una única columna de nombre `atr` y un único valor `v`.

5. `Descendientes : string x Vista → Conjunto(string)`
   `Descendientes(e,V)` retorna el conjunto de nombres de elementos *XML* que son descendientes del elemento `e` en la vista *XML* `V` (incluyendo el elemento `e`) o bien el conjunto vacío si `e` es un atributo XML.

6. `EsAtributo: string → Boolean`
   `EsAtributo(c)` devuelve verdadero si `c` es un atributo (o sea es conforme a la producción `Atributo` de la gramática de la sección 4.2.1) y falso en caso contrario.

7. `EsCaminoAtributo: string → Boolean`
   `EsCaminoAtributo(c)` devuelve verdadero si el camino de localización `c` finaliza en un atributo (o sea es conforme a la producción `CaminoAtributo` de la gramática de la sección 4.2.1) y falso en caso contrario.

8. `EsValor: string → Boolean`
   `EsValor(c)` devuelve verdadero si `c` es un valor (o sea es conforme a la producción `Valor` de la gramática de la sección 4.2.1) y falso en caso contrario.

9. `FuncRen : Conjunto(string) → Conjunto(string)`
   `FuncRen(A)` retorna el conjunto resultado de renombrar cada elemento $a_i \in A$ con el prefijo "ctx_".

10. `FuncRen`$^{-1}$ `: Conjunto(string) → Conjunto(string)`
    `FuncRen`$^{-1}$`(A)` retorna el conjunto resultado de renombrar cada elemento $a_i \in A$ quitándole el prefijo "ctx_".

11. `GenComas : Conjunto(string) → string`
    `GenComas(A)` retorna un string formado por los ítems de `A` separados por coma (,). Si `A` tiene un único ítem retorna un string con un solo formado por el ítem de `A`.

12. `GenCond : Conjunto(string) x Conjunto(string) x string → string`
    `GenCond(A,B,op)` genera una condición de la forma:

    $$a_1 \text{ op } b_1 \wedge a_2 \text{ op } b_2 \wedge \ldots \wedge a_n \text{ op } b_n$$

    para cada $a_i \in A$ y cada $b_i \in B$.

    Si `A` o `B` son vacíos ó no tienen la misma cantidad de elementos retorna el string nulo.

    Cuando es invocada en las reglas $\boldsymbol{P}_{ALG}[cc]$ y $\boldsymbol{P}_{ALG}[ca]$ recibe, en ambos casos, dos conjuntos como parámetro de entrada que tienen un único elemento, por lo tanto no hay ambigüedad para generar la condición.

    Cuando es invocada desde la función `Generar` recibe dos conjuntos como parámetros de entrada, en el primer conjunto los elementos tienen como prefijo "ctx_" y como sufijo un elemento del segundo conjunto, por lo tanto no hay ambigüedad para generar la condición, la cual tiene la siguiente forma:

    $$ctx\_a_1 \text{ op } a_1 \wedge ctx\_a_2 \text{ op } a_2 \wedge \ldots \wedge ctx\_a_n \text{ op } a_n$$

    Cuando es invocada desde la función `ResolverComp` recibe dos conjuntos como parámetros de entrada, en cada uno de los cuales los elementos tienen como prefijo

"ctx_" y como sufijo un string que es el mismo en ambos casos, ó bien ambos conjuntos tienen un único elemento. Por lo tanto no hay ambigüedad para generar la condición, la cual tiene la siguiente forma:

ctx_a_1 op $ctx\_a_1 \wedge ctx\_a_2$ op $ctx\_a_2 \wedge \ldots \wedge ctx\_a_n$ op ctx_a_n

13. Generar : SecExprAlg x SecExprAlg x ExprAlg → string
Generar(S_1,S_2,E) retorna una expresión algebraica que incluye como columnas las columnas de la expresión algebraica E (contexto del predicado) para lo cual estas columnas se renombran con un nombre particular.

Como los caminos dentro de un predicado son relativos la secuencia S_1 incluye a la secuencia S_2 (por construcción), por lo cual se elimina la secuencia S_2 de la secuencia S_1, para considerar únicamente la subsecuencia relativa al operando del predicado.

A continuación se modifica la primer expresión algebraica de la secuencia S_1, de manera que se incluyan, en la misma, las columnas correspondientes a la expresión algebraica contexto, para lo cual se aplica un producto cartesiano.
Se recorre el resto de la secuencia S_1 sustituyendo la expresión algebraica que ocupa la posición *i* por la expresión algebraica que ocupa la posición *i* −1. A su vez si la expresión que ocupa la posición *i* es una proyección se modifica la misma para que la proyección conserve las columnas de la expresión contexto.
A pesar que se realiza un trabajo extra, es conveniente recorrer la secuencia de manera de asegurar que las sustituciones y modificaciones se realizan correctamente, dado que la misma subexpresión algebraica puede aparecer en distintas partes de una expresión algebraica y no debe ser sustituida o modificada en todas ellas.

```
Generar(S1,S2,E) =
 let S = Elim(S1,S2)
 let A = Atr(E)
 let Ectx= Appρ(E,A,FuncRen(A))
 let Actx = Atr(Ectx)
 let Eprim = Primera(S)
 let Aprim = Atr(Eprim)
 let S = Resto(S)
 if IgualEsquema(Eprim,Oj(M,V)) then
   let R = Appx(Ectx,Eprim)
 else
  let cond = GenCond(FuncRen(Aprim ∩ A),Aprim ∩ A, "=")
  if ¬Isnull(cond) then
     let R=AppΠ(Appσ(Appx(Ectx,Eprim),cond),Aprim ∪ Actx)
  else
     let R = AppΠ(Appx(Ectx,Eprim),Aprim ∪ Actx)
  end if
 end if
 let anterior = Eprim
 let cambio = R
 while ¬ Vacia(S)
  let actual = Primera(S)
  let cambio=ModifΠ( Sustituir(actual,anterior,cambio),
                     Atr(Ectx))
  let anterior = actual
  let S = Resto(S)
 end while
 return cambio
```

14. `IsNull : string → Boolean`
    `IsNull(a)` retorna verdadero si a es nulo y falso en caso contrario.

15. `NuevoOp: string → string`
    `NuevoOp(op)` devuelve el opuesto del operador op, excepto si op es "=" o "!=" en cuyo caso retorna el mismo.

16. `ObtAtrib: string → string`
    `ObtAtrib(c)` devuelve c sin el símbolo @.

17. `ObtConPrefijo : Conjunto(string) x string → Conjunto(string)`
    `ObtconPrefijo(A,b)` retorna el conjunto de elementos $a_i \in A$ que tienen el prefijo b.

18. `ResolverAtr_Atr: Mapeo x string x Operando x Operando x Operador → ResP`$_{ALG}$

```
ResolverAtr_Atr(M,E,o1,o2,op) =
 let c = GenCond(
                  Atributos(M,ObtAtrib(o1)),
                  Atributos(M,ObtAtrib(o2)),
                  Op
                 )
 return Cons(Appσ(E,c),false)
```

19. `ResolverAtr_Cam : Mapeo x SecExprAlg x string x Operando x Operando x Operador x Entero→ ResP`$_{ALG}$

```
ResolverAtr_Cam(M,S,E,o1,o2,op,N) =
 let A = Atr(E) – Atributos(M,ObtAtrib(o1))
 let Ectx= Appρ(E,A,FuncRen(A))
 let O2 = OALG[o2](S,N,M,V)
 if ¬ EsError(O2)  then
    let S2 = SelSec(O2)
    let R = Generar(S2,S,E)
    return Cons(
                ResolverComp(
                             Ectx,
                             R,
                             Atributos(M,ObtAtrib(o1)),
                             SelAtr(O2),
                             Op
                            ),
                false
           )
 else
    return Cons(⊥,true)
 end if
```

20. `ResolverAtr_Val: Mapeo x string x Operando x Operando x Operador → ResP`$_{ALG}$

```
ResolverAtr_Val(M,E,o1,o2,op) =
 let c = GenCond(Atributos(M,ObtAtrib(o1)),{o2},op)
 return Cons(Appσ(E,c),false)
```

21. ResolverCam: string → string
ResolverCam(E) asegura que la expresión algebraica E tenga como columnas únicamente aquellas que corresponden a la expresión algebraica contexto (que son las que tienen el prefijo "ctx_") con el nombre correcto (es decir sin el prefijo "ctx_").

```
ResolverCam(E) =
 let A = ObtConPrefijo(Atr(E),"ctx_")
 let R = Appρ(AppΠ(E,A),A,FuncRen^-1(A))
 return R
```

22. ResolverCam_Cam: Mapeo x SecExprAlg x string x Operando x Operando x Operador x Entero→ $ResP_{ALG}$

```
ResolverCam_Cam(M,S,E,o1,o2,op,N) =
 let O1 = O_ALG[o1](S,N,M,V)
 let O2 = O_ALG[o2](S,N,M,V)
 if ¬ EsError(O1) ∧ ¬ EsError(O2)  then
  let S1 = SelSec(O1)
  let S2 = SelSec(O2)
  let R1 = Generar(S1,S,E)
  let R2 = Generar(S2,S,E)
  returnCons(ResolverComp(R1,R2,SelAtr(O1),SelAtr(O2),op),false)
 else
   return Cons(⊥,true)
 end if
```

23. ResolverComp: string x string x Conjunto(string) x Conjunto(string) x Operador → string

ResolverComp(R_1,R_2,a,b,op)

Dadas R_1 y R_2, expresiones algebraicas que corresponden a los operandos de un predicado, que incluyen las columnas de la expresión algebraica contexto prefijadas con "ctx_", esta función es utilizada para generar todas las combinaciones posibles entre los candidatos denotados por las expresiones algebraicas R_1 y R_2 de forma de encontrar si existe un par que cumpla la condición determinada por el predicado (recordar la definición de la semántica en la sección 4.3).

Para lograr esto se aplica el producto cartesiano y una selección. Finalmente esta función asegura que la expresión algebraica que se retorna tenga como columnas únicamente aquellas que corresponden a la expresión contexto (que son las que tienen el prefijo "ctx_") con el nombre correcto (es decir sin el prefjijo "ctx_").

```
ResolverComp(R1,R2,a,b,op) =
 let A1 = ObtConPrefijo(Atr(R1),"ctx_")
 let A2 = ObtConPrefijo(Atr(R2),"ctx_")
 let cond1 =  GenCond(A1,A2,"=")
 let cond2 =  GenCond(a,b,op)
 let cond = Concat(Concat(cond1,"∧"), cond2)
 let R= Appρ(AppΠ(
                  Appσ(Appx(R1,R2),cond),
                  A1
                 ),
             A1,FuncRen^-1(A1)
        )
 return R
```

24. ResolverVal_Cam: Mapeo x SecExprAlg x string x Operando x Operando x Operador x Entero→ $ResP_{ALG}$

```
ResolverVal_Cam(M,S,E,o1,o2,op,N) =
 let A = Atr(E)
 let Ectx= Appρ(E,A,FuncRen(A))
 let R1 = Appx(Ectx,CrearRel(o1,"valor","valor"))
 let O2 = O_ALG[o2](S,N,M,V)
 if ¬ EsError(O2)  then
    let S2 = SelSec(O2)
    let R2 = Generar(S2,S,E)
    return Cons(ResolverComp(R1,R2,"valor",SelAtr(O2),op),false)
 else
    return Cons(⊥,true)
 end if
```

25. ResolverVal_Val: Mapeo x string x Operando x Operando x Operador→ $ResP_{ALG}$

```
ResolverVal_Val(M,E,o1,o2,op) =
 let c = GenCond({o1},{o2},op)
 return Cons(Appσ(E,c),false)
```

26. Tabla: Mapeo x string → string

Tabla(M,n) retorna el nombre de la tabla que corresponde al elemento n en el mapeo M.

27. Ult : string → string

Ult(c) retorna el último paso de localización de la consulta $Xpath_{Sub}$ c.

Anexo 3 - Ejemplos de aplicación del algoritmo

10.1 Introducción

En este anexo se muestran dos ejemplos de cómo se aplica el algoritmo de conversión definido para generar una expresión algebraica a partir de una consulta $Xpath_{Sub}$, aplicando la mejora del algoritmo definida en la sección 4.6.1.

10.2 Consulta 1

La consulta: "devolver los números de producto que fueron vendidos a un precio mayor que 200 pesos" se expresa en $Xpath_{Sub}$ como:

```
/fabricante/venta[@precio > 200]/@numero_producto
```

para realizar la conversión a álgebra relacional se realiza la invocación:

$\boldsymbol{XPath_{Sub}ToAlg}$`[/fabricante/venta[@precio > 200]/@numero_producto](M,V)`

siendo `M` el mapeo y `V` la vista *XML* (*XML Schema*) de la base de datos, que a su vez realiza la invocación:

$\boldsymbol{(1)X_{ALG}}$`[/fabricante/venta[@precio > 200]/@numero_producto](∅,0,M,V)`

se aplica la regla $\boldsymbol{X_{ALG}}$`[/ca]` y se realiza la invocación:

$\boldsymbol{(2)X_{ca\_ALG}}$`[fabricante/venta[@precio>200]/@numero_producto](Ins(∅,Oj(M,V)),1,M,V)`

se aplica la regla $\boldsymbol{X_{ca\_ALG}}$`[cc/@n]` y se realiza la invocación:

$\boldsymbol{(3)X_{cc\_ALG}}$`[fabricante/venta @precio>200](Ins(∅,Oj(M,V)),1,M,V)`

se aplica la regla $\boldsymbol{X_{cc\_ALG}}$`[cc p]` y se realiza la invocación:

$\boldsymbol{(4)X_{cc\_ALG}}$`[fabricante/venta](Ins(∅,Oj(M,V)),1,M,V)`

se aplica la regla $\boldsymbol{X_{cc\_ALG}}$`[cc/n]` y se realiza la invocación:

$\boldsymbol{(5)X_{cc\_ALG}}$`[fabricante](Ins(∅,Oj(M,V),1,M,V)`

se aplica la regla $\boldsymbol{X_{cc\_ALG}}$`[n]` y se inserta en la secuencia de expresiones algebraicas `Ins(∅,Oj(M,V)` la expresión algebraica[26]:

$$\Pi_{\texttt{numero\_fabricante,nombre,direccion}}(\texttt{Oj(M,V)* Fabricantes})$$

obteniéndose como resultado de $\boldsymbol{(5)}$:

$$\texttt{Cons(}\ \texttt{Ins(}\ \texttt{Ins(∅,Oj(M,V),}\ \Pi_{\texttt{numero\_fabricante,nombre,direccion}}(\texttt{Oj(M,V)* Fabricantes})\ \texttt{),}\ \texttt{2,}\ \texttt{false}\ \texttt{)}$$

[26] Se utiliza `Oj(M,V)` por razones de simplicidad, aunque debería sustituirse por la expresión algebráica correspondiente.

se retorna a ***(4)*** obteniéndose como resultado:

```
Cons(
     Ins(
         Ins(
             Ins(∅,Oj(M,V),
             Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
             ),
             Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas
     ),
     3,
     false
)
```

se retorna a ***(3)*** y se realiza la invocación:

```
(6)P ALG[@precio > 200](
     Ins(
         Ins(
             Ins(∅,Oj(M,V),
             Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
         ),
         Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas
     ),
     3,
     M,
     V
  )
```

se aplica la regla $P_{ALG}[o_1\ op\ o_2]$, como el operando izquierdo es un atributo y el operando derecho es un valor se invoca a la función:

```
(7)ResolverAtr_Val(
     M,
     Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas,
     @precio,
     200,
     ">"
  )
```

la función `ResolverAtr_Val` retorna:

```
  Cons(
       σ precio>200(Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas),
       false
  )
```

se retorna a ***(3)*** obteniéndose como resultado:

```
 Cons(
      Ins(
          Ins(
              Ins(∅,Oj(M,V),
              Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
              ),
              Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas *
              σ precio>200(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes) * Ventas)
      ),
      3,
      false
 )
```

se retorna a ***(2)*** y se obtiene:

```
Cons(
      Ins(
          Ins(
              Ins(
                  Ins(∅,Oj(M,V),
                  Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
              ),
              Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas *
              σ precio>200(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes)*Ventas)
          ),
          Π numero_producto(
            Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas *
            σ precio>200(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes) * Ventas)
          )
      ),
      4,
      false
)
```

se retorna a ***(1)*** y a la función ***XPath$_{Sub}$ToAlg***. Se invoca a la función `Descendientes(@numero_producto)`, que retorna el conjunto vacío y se devuelve, finalmente, la expresión algebraica:

```
Π numero_producto(
   Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas *
      σ precio>200(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes) * Ventas)
)
```

Si se aplica el proceso de optimización descripto en la sección 4.7, se obtiene:

```
Π numero_producto(Fabricantes * Ventas * σ precio>200(Fabricantes * Ventas))
```

10.3 Consulta 2

La consulta: “devolver los nombres de fabricantes que realizaron ventas tales que la fecha de venta de alguna de sus ventas es igual a la fecha de envío de algunas de sus ventas” (ejemplo 4 de la sección 4.4) se expresa en $Xpath_{Sub}$ como:

```
/fabricante[venta/@fecha_venta=venta/@fecha_envio]/@nombre
```

para realizar la conversión a álgebra relacional se realiza la invocación:

$\boldsymbol{XPath_{Sub}ToAlg}$[/fabricante[venta/@fecha_venta=venta/@fecha_envio]/@nombre](M,V)

siendo M el mapeo y V la vista *XML* (*XML Schema*) de la base de datos, que a su vez realiza la invocación:

$\boldsymbol{(1)X}_{ALG}$[/fabricante[venta/@fecha_venta=venta/@fecha_envio]/@nombre](∅,0,M,V)

se aplica la regla $\boldsymbol{X}_{ALG}[/ca]$ y se realiza la invocación:

$\boldsymbol{(2)X}_{ca\_ALG}$[fabricante[venta/@fecha_venta=venta/@fecha_envio]/@nombre]
(Ins(∅,Oj(M,V)),1,M,V)

se aplica la regla $\boldsymbol{X}_{ca\_ALG}[cc/@n]$ y se realiza la invocación:

$\boldsymbol{(3)}$
$\boldsymbol{X}_{cc\_ALG}$[fabricante venta/@fecha_venta=venta/@fecha_envio](Ins(∅,Oj(M,V)),1,M,V)

se aplica la regla $\boldsymbol{X}_{cc\_ALG}[cc\ p]$ y se realiza la invocación:

$\boldsymbol{(4)X}_{cc\_ALG}$[fabricante](Ins(∅,Oj(M,V)),1,M,V)

se aplica la regla $\boldsymbol{X}_{cc\_ALG}[n]$ y se inserta en la secuencia de expresiones algebraicas Ins(∅,Oj(M,V)) la expresión algebraica:

$$\Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)*\ Fabricantes)$$

obteniéndose como resultado de $\boldsymbol{(4)}$:

```
Cons(
     Ins(
         Ins(∅,Oj(M,V),
         Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
     ),
     2,
     false
)
```

se retorna a $\boldsymbol{(3)}$ y se realiza la invocación:

$\boldsymbol{(5)P}_{ALG}$[venta/@fecha_venta=venta/@fecha_envio](
Ins(Ins(∅,Oj(M,V),$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)),
2,
M,
V
)

se aplica la regla $\boldsymbol{P}_{ALG}[o_1\ op\ o_2]$ y como ambos operandos son caminos se invoca a la función:

(6)ResolverCam_Cam(
M,
Ins(Ins(∅,Oj(M,V),$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)),
$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes),
venta/@fecha_venta,
venta/@fecha_envio,
"=",
2
)

la función ResolverCam_Cam realiza las invocaciones ***(7)*** y ***(8)***:

(7) O_{ALG}[venta/@fecha_venta](
Ins(Ins(∅,Oj(M,V),$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)),
2,
M,
V
)

(8) O_{ALG}[venta/@fecha_envio](
Ins(Ins(∅,Oj(M,V),$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)),
2,
M,
V
)

(7) invoca a:

(9) X_{ca_ALG}[venta/@fecha_venta](
Ins(Ins(∅,Oj(M,V),$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)),
2,
M,
V
)

se aplica la regla X_{ca_ALG}[*cc/@n*] y se realiza la invocación:

(10) X_{cc_ALG}[venta](
Ins(Ins(∅,Oj(M,V),$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)),
2,
M,
V
)

se aplica la regla X_{cc_ALG}[*n*] y se inserta en la secuencia de expresiones algebraicas la expresión algebraica:

$$\Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes) * Ventas$$

obteniéndose como resultado de ***(10)***:

Cons(
Ins(
Ins(
Ins(∅,Oj(M,V),
$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)
),
$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)* Ventas
),
3,
false
)

se retorna a ***(9)*** y se obtiene como resultado:

```
Cons(
      Ins(
          Ins(
              Ins(
                  Ins(∅,Oj(M,V),
                  Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
              ),
              Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes) * Ventas
          ),
          Π fecha_venta(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes)*Ventas)
      ),
      4,
      false
)
```

se retorna a **(7)** y se obtiene como resultado:

```
Cons(
      Ins(
          Ins(
              Ins(
                  Ins(∅,Oj(M,V),
                  Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
              ),
              Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas
          ),
          Π fecha_venta(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes)*Ventas)
      ),
      fecha_venta,
      false
)
```

como resultado de ***(8)*** (que es análoga a ***(7)***) se obtiene:

```
Cons(
      Ins(
          Ins(
              Ins(
                  Ins(∅,Oj(M,V),
                  Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
              ),
              Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes) * Ventas
          ),
          Π fecha_envio(Π numero_fabricante,nombre,direccion(Oj(M,V)*Fabricantes)*Ventas)
      ),
      fecha_envio,
      false
)
```

se retorna a ***(6)*** y se invoca a ***(11)*** y ***(12)*** :

```
(11)Generar(
      Ins(
          Ins(
              Ins(
                  Ins(∅,Oj(M,V),
                      Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
                  ),
                  Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas
              ),
              Π fecha_venta(Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas)
      ),
      Ins(
          Ins(∅,Oj(M,V),
          Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
      ),
      Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
   )

(12)Generar(
      Ins(
          Ins(
              Ins(
                  Ins(∅,Oj(M,V),
                      Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
                  ),
                  Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas
              ),
              Π fecha_envio(Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas)
      ),
      Ins(
          Ins(∅,Oj(M,V),
          Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
      ),
      Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
   )
```

obteniéndose como resultado de ***(11)***:

```
Π fecha_venta,ctx_numero_fabricante,ctx_nombre,ctx_direccion
   (Π numero_fabricante,nombre,direccion,numero_producto,precio,fecha_venta,fecha_envio,
      ctx_numero_fabricante,ctx_nombre,ctx_direccion
      (σ ctx_numero_fabricante=numero_fabricante ∧ ctx_mombre=nombre ∧ ctx_direccion=direccion
         (ρ numero_fabricante,nombre,direccion →ctx_numero_fabricante,ctx_nombre,ctx_direccion(
            Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes)
         ) x Π numero_fabricante,nombre,direccion(Oj(M,V)* Fabricantes) * Ventas
      )
   )
)
```

como resultado de ***(12)*** (que es análoga a ***(11)***) se obtiene:

$\Pi_{fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$
(
$\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$
$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$
($\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$
($\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$(
$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)
) x $\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes) * Ventas
)
)
)

se retorna a ***(6)*** y se invoca a:

(13)ResolverComp(
$\Pi_{fecha\_venta,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$
(
$\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$
$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$
($\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$
($\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$(
$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)
) x $\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes) * Ventas
)
)
),
$\Pi_{fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$
(
$\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$
$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$
($\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$
($\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$(
$\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes)
) x $\Pi_{numero\_fabricante,nombre,direccion}$(Oj(M,V)* Fabricantes) * Ventas
)
)
),
fecha_venta,
fecha_envio,
"="
)

que retorna:

$\rho_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion \rightarrow numero\_fabricante,nombre,direccion}$(
$\Pi_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$(
$\sigma_{ctx\_numero\_fabricante= ctx\_numero\_fabricante \wedge ctx\_nombre=ctx\_nombre \wedge ctx\_direccion=ctx\_direccion}$
$\wedge$ fecha_venta=fecha_envio
(R_1 x R_2)
)
)

siendo R_1:

$$\Pi_{fecha\_venta,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$$
$$(\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$$
$$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$$
$$(\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$$
$$(\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}($$
$$\Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes)$$
$$) \times \Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes) * Ventas$$
$$)$$
$$)$$
$$)$$

y siendo R_2:

$$\Pi_{fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$$
$$(\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$$
$$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$$
$$(\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$$
$$(\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}($$
$$\Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes)$$
$$) \times \Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes) * Ventas$$
$$)$$
$$)$$
$$)$$

se retorna a ***(6)***, obteniéndose como resultado (y en consecuencia para ***(5)***):

$$Cons($$
$$\rho_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion \rightarrow numero\_fabricante,nombre,direccion}($$
$$\Pi_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}($$
$$\sigma_{ctx\_numero\_fabricante=ctx\_numero\_fabricante \wedge ctx\_nombre=ctx\_nombre \wedge}$$
$$ctx\_direccion=ctx\_direccion \wedge fecha\_venta=fecha\_envio$$
$$(R_1 \times R_2)$$
$$)$$
$$),$$
$$false$$
$$)$$

se retorna a ***(3)*** y se obtiene:

$$Cons($$
$$Ins($$
$$Ins(\emptyset,Oj(M,V),$$
$$\Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes) *$$
$$\rho_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion \rightarrow numero\_fabricante,nombre,dirccion}($$
$$\Pi_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}($$
$$\sigma_{ctx\_numero\_fabricante=ctx\_numero\_fabricante \wedge ctx\_nombre=ctx\_nombre \wedge}$$
$$ctx\_direccion=ctx\_direccion \wedge fecha\_venta=fecha\_envio$$
$$(R_1 \times R_2)$$
$$)$$
$$)$$
$$),$$
$$2,$$
$$false$$
$$)$$

se retorna a ***(2)*** y se obtiene:

$$
\begin{aligned}
&\text{Cons}(\\
&\quad \text{Ins}(\\
&\quad\quad \text{Ins}(\\
&\quad\quad\quad \text{Ins}(\\
&\quad\quad\quad\quad \text{Ins}(\emptyset, \text{Oj(M,V)},\\
&\quad\quad\quad\quad \Pi_{\text{numero\_fabricante,nombre,direccion}}(\text{Oj(M,V)* Fabricantes}) *\\
&\quad\quad\quad\quad \rho_{\text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion} \rightarrow \text{numero\_fabricante,nombre,dirccion}}(\\
&\quad\quad\quad\quad\quad \Pi_{\text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\\
&\quad\quad\quad\quad\quad\quad \sigma_{\text{ctx\_numero\_fabricante=ctx\_numero\_fabricante} \wedge \text{ctx\_nombre=ctx\_nombre} \wedge}\\
&\quad\quad\quad\quad\quad\quad\quad \text{ctx\_direccion=ctx\_direccion} \wedge \text{fecha\_venta=fecha\_envio}\\
&\quad\quad\quad\quad\quad\quad\quad (R_1 \times R_2)\\
&\quad\quad\quad\quad\quad)\\
&\quad\quad\quad\quad)\\
&\quad\quad\quad),\\
&\quad\quad\quad \Pi_{\text{nombre}}(\\
&\quad\quad\quad\quad \Pi_{\text{numero\_fabricante,nombre,direccion}}(\text{Oj(M,V)* Fabricantes}) *\\
&\quad\quad\quad\quad\quad \rho_{\text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion} \rightarrow \text{numero\_fabricante,nombre,dirccion}}(\\
&\quad\quad\quad\quad\quad\quad \Pi_{\text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\\
&\quad\quad\quad\quad\quad\quad\quad \sigma_{\text{ctx\_numero\_fabricante=ctx\_numero\_fabricante} \wedge}\\
&\quad\quad\quad\quad\quad\quad\quad \text{ctx\_nombre=ctx\_nombre} \wedge \text{ctx\_direccion=ctx\_direccion} \wedge\\
&\quad\quad\quad\quad\quad\quad\quad \text{fecha\_venta=fecha\_envio}\\
&\quad\quad\quad\quad\quad\quad\quad (R_1 \times R_2)\\
&\quad\quad\quad\quad\quad\quad)\\
&\quad\quad\quad\quad\quad)\\
&\quad\quad\quad)\\
&\quad\quad),\\
&\quad\quad 3,\\
&\quad\quad \text{false}\\
&)
\end{aligned}
$$

se retorna a ***(1)*** y a la función $\mathcal{XPath}_{Sub}\mathcal{ToAlg}$. Se invoca a la función `Descendientes(@nombre)`, que retorna el conjunto vacío y se devuelve, finalmente, la expresión algebraica:

$$
\begin{aligned}
&\Pi_{\text{nombre}}(\\
&\quad \Pi_{\text{numero\_fabricante,nombre,direccion}}(\text{Oj(M,V)* Fabricantes}) *\\
&\quad \rho_{\text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion} \rightarrow \text{numero\_fabricante,nombre,dirccion}}(\\
&\quad\quad \Pi_{\text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\\
&\quad\quad\quad \sigma_{\text{ctx\_numero\_fabricante=ctx\_numero\_fabricante} \wedge \text{ctx\_nombre=ctx\_nombre} \wedge}\\
&\quad\quad\quad \text{ctx\_direccion=ctx\_direccion} \wedge \text{fecha\_venta=fecha\_envio}\\
&\quad\quad\quad (R_1 \times R_2)\\
&\quad\quad)\\
&\quad)\\
&)
\end{aligned}
$$

siendo R_1:

$$
\begin{aligned}
&\Pi_{\text{fecha\_venta,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}\\
&\quad (\Pi_{\text{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}}\\
&\quad\quad \text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}\\
&\quad\quad (\sigma_{\text{ctx\_numero\_fabricante=numero\_fabricante} \wedge \text{ctx\_mombre=nombre} \wedge \text{ctx\_direccion=direccion}}\\
&\quad\quad\quad (\rho_{\text{numero\_fabricante,nombre,direccion} \rightarrow \text{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}}(\\
&\quad\quad\quad\quad \Pi_{\text{numero\_fabricante,nombre,direccion}}(\text{Oj(M,V)* Fabricantes})\\
&\quad\quad\quad) \times \Pi_{\text{numero\_fabricante,nombre,direccion}}(\text{Oj(M,V)* Fabricantes}) * \text{Ventas}\\
&\quad\quad)\\
&\quad)\\
&)
\end{aligned}
$$

y siendo R_2:

$$\Pi_{fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$$
$$(\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$$
$$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$$
$$(\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$$
$$(\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}($$
$$\Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes)$$
$$) \times \Pi_{numero\_fabricante,nombre,direccion}(Oj(M,V)* Fabricantes) * Ventas$$
$$)$$
$$)$$
$$)$$

Si se aplica el proceso de optimización descripto en la sección 4.7, se obtiene:

$$\Pi_{nombre}($$
$$Fabricantes *$$
$$\rho_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion \rightarrow numero\_fabricante,nombre,dirccion}($$
$$\Pi_{ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}($$
$$\sigma_{ctx\_numero\_fabricante=ctx\_numero\_fabricante \wedge ctx\_nombre=ctx\_nombre \wedge}$$
$$ctx\_direccion=ctx\_direccion \wedge \ fecha\_venta=fecha\_envio$$
$$(R_1 \times R_2)$$
$$)$$
$$)$$
$$)$$

siendo R_1:

$$\Pi_{fecha\_venta,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$$
$$(\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$$
$$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$$
$$(\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$$
$$(\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}(Fabricantes)$$
$$\times Fabricantes * Ventas$$
$$)$$
$$)$$
$$)$$

y siendo R_2:

$$\Pi_{fecha\_envio,ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}$$
$$(\Pi_{numero\_fabricante,nombre,direccion,numero\_producto,precio,fecha\_venta,fecha\_envio,}$$
$$ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion$$
$$(\sigma_{ctx\_numero\_fabricante=numero\_fabricante \wedge ctx\_mombre=nombre \wedge ctx\_direccion=direccion}$$
$$(\rho_{numero\_fabricante,nombre,direccion \rightarrow ctx\_numero\_fabricante,ctx\_nombre,ctx\_direccion}(Fabricantes)$$
$$\times Fabricantes * Ventas$$
$$)$$
$$)$$
$$)$$