Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Extracción, Transformación y Carga (ETL) en Plataformas de Integración

Informe de Proyecto de Grado

Andrés Pintos Faccio
Marcos Morales Iribarnegaray
Victoria Lavella Curbelo

Docentes Supervisores: Laura González, Marcelo Belén
Usuario Responsable: Marcelo Belén

Junio, 2015

Resumen

Las arquitecturas orientadas a servicios (Service Oriented Architecture, SOA) plantean un enfoque prometedor, que permite el desarrollo ágil y de bajo costo de los sistemas basados en servicios. La construcción de estos sistemas, se apoya en general en plataformas de integración, las cuales brindan un medio a través del cual se realizan invocaciones a los servicios. En este contexto puede resultar de interés analizar en tiempo real la información de los mensajes que circulan a través de una plataforma de integración, extrayendo información relevante sobre el uso del sistema que ayude a la toma de decisiones.

Por otro lado, los sistemas de Data Warehousing (DWs) surgen como un mecanismo de apoyo para la toma de decisiones, en el que los datos de una organización se transforman en información estratégica, que además puede ser accedida de manera sencilla en cualquier momento.

En este proyecto, se incorporaron mecanismos a una plataforma de integración, basada en un Enterprise Service Bus (ESB), que permiten capturar información de interés para el usuario de los mensajes que circulan a través del ESB, almacenando dicha información en un Data Warehouse.

En primer lugar, se realizó un análisis de la problemática planteada, identificando una serie de requerimientos, y se hizo un relevamiento de soluciones existentes que resolvieran dicha problemática. En este relevamiento se encontraron dos enfoques relacionados a la carga de un Data Warehouse en tiempo real, de los cuales se pudieron aprovechar algunas ideas.

En base a este análisis se propuso una solución que incorpora un conjunto de componentes a una plataforma de integración basada en un ESB, con el fin de alcanzar los objetivos planteados. Las principales características de la solución propuesta son la construcción de un Data Warehouse de forma automática a partir de configuraciones que puede realizar un usuario y la obtención de información tanto de los mensajes que circulan por el ESB, como aplicando técnicas de Complex Event Processing (CEP).

A partir de esta propuesta se desarrolló un prototipo basado principalmente en la utilización de Switchyard como plataforma ESB y Esper como motor CEP. En la etapa final del proyecto, se definieron tres casos de estudio que fueron utilizados durante el período de pruebas, para comprobar el correcto funcionamiento del sistema y evaluar el rendimiento de la plataforma.

Como conclusión general podemos decir que este tipo de solución puede ser de mucha utilidad para resolver la problemática planteada dentro de un ambiente controlado, por ejemplo dentro de una organización.

Palabras clave: Service Oriented Architecture, Web Services, Data Warehouse, Enterprise Service Bus, Complex Event Processing.

Contenido

1 Introducción 7
1.1 Objetivos 8
1.2 Aportes del proyecto 9
1.3 Organización del Documento 9
2 Marco Teórico 11
2.1 Extensible Markup Language 11
2.2 Service Oriented Architecture 13
2.3 Web Services 14
2.4 Enterprise Service Bus 16
2.5 Complex Event Processing 18
2.6 Data Warehouse 20
3 Análisis 27
3.1 Análisis conceptual 27
3.2 Relevamiento de requerimientos 30
3.3 Trabajo Relacionado 31
4 Solución propuesta 35
4.1 Descripción general 35
4.2 Flujo de mensajes 37
4.3 Diseño del Data Warehouse 38
4.4 Diseño general 46
4.5 Decisiones de diseño 47
5 Detalles de implementación 49
5.1 Selección de tecnologías 49
5.2 Implementación 53
5.3 Limitaciones de la implementación 68
6 Casos de estudio 71
6.1 Servicios utilizados 71
6.2 Caso de estudio 1: Monitoreo de un único servicio. 73
6.3 Caso de estudio 2: Monitoreo en dos servicios relacionados entre sí. 76
6.4 Caso de estudio 3: Monitoreo de tres servicios relacionados entre sí. 80
7. Conclusiones y trabajo a futuro 83

7.1 Resumen y conclusiones finales....83
7.2 Trabajo a futuro....84
8 Referencias....87
Apéndice 1. Primer diseño del Data Warehouse....91
Apéndice 2 Glosario....97

1 Introducción

En la actualidad, los sistemas empresariales tienden cada vez más a estar distribuidos, tanto físicamente en distintos centros de cómputo como lógicamente en diferentes departamentos pertenecientes a las organizaciones.

Esta tendencia a la distribución, hace que surjan nuevos enfoques sobre las arquitecturas de los sistemas, como es el caso de las arquitecturas orientadas a servicios (Service Oriented Architecture, SOA). En una arquitectura SOA se plantea un escenario en el cual las diferentes aplicaciones de las organizaciones exponen sus funcionalidades a través de servicios -los cuales son invocados por otras aplicaciones- para dar soporte a requisitos de negocio.

A su vez el Enterprise Service Bus (ESB) es reconocido como una infraestructura para dar soporte a la implementación de una SOA. Un ESB consiste en una plataforma de integración que provee herramientas para lograr la interoperabilidad entre sistemas heterogéneos, utilizando por ejemplo ruteo, transformación y adaptadores, entre otros. En este contexto los servicios no se comunican directamente entre ellos, sino que lo hacen por medio del envío de mensajes a través del ESB como puede verse gráficamente en la Figura 1.

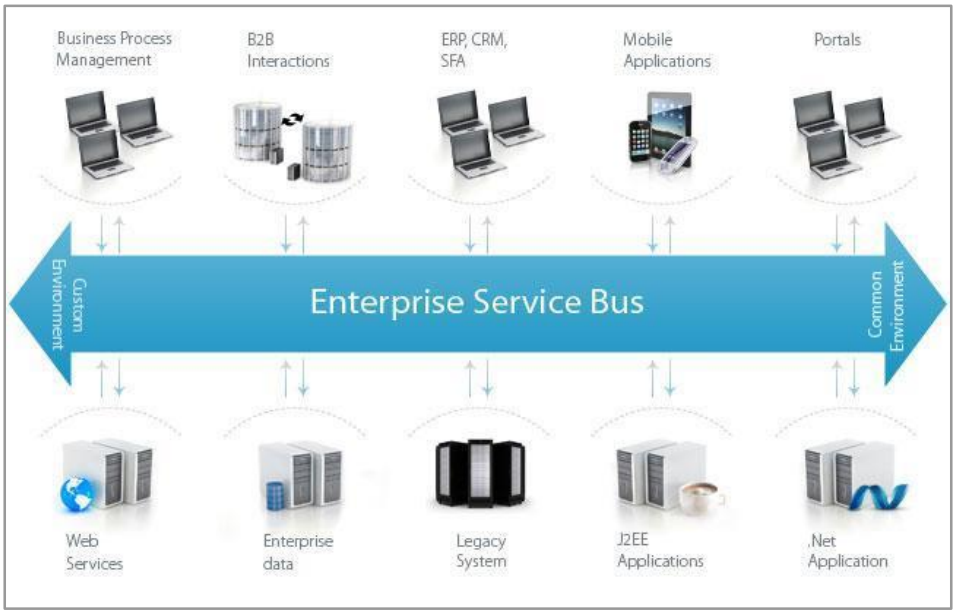


Figura 1: Comunicación en un ESB [1].

Por otro lado, los sistemas de Data Warehousing (DW) son un tipo de sistema de información orientado a la toma de decisiones. Este tipo de sistema toma datos de los sistemas operacionales de las organizaciones (CRM, manejo de clientes, entre otros) y brinda diferentes resúmenes de los mismos. Los procesos de Extracción, Transformación y Carga (ETL) para la construcción de un Data Warehouse se realizan, en general, a partir de las bases de datos de dichos sistemas operacionales.

Sin embargo, existen situaciones donde puede resultar de interés contar con resúmenes de datos en base a las interacciones que se dan a través de las plataformas de integración. Un buen ejemplo de esto es cuando se requiere extraer información relevante del negocio, como pueden ser métricas asociadas a las interacciones que se realizan en la plataforma, para luego utilizarlas en la toma de decisiones.

En estos casos, los procesos de ETL deberían tomar como fuente de datos los mensajes enviados a través de la plataforma para invocar a los servicios. La motivación del proyecto entonces, apunta a proveer soluciones que permitan analizar y obtener información de dicha fuente de datos, con el fin de realizar la carga de un Data Warehouse. Si bien las plataformas de integración de tipo ESB, cuentan en la actualidad con herramientas de monitoreo y análisis, no proveen una solución para este tipo de problema.

1.1 Objetivos

Dada la problemática planteada, el objetivo general del proyecto es proponer mecanismos a incorporar en una plataforma de integración, basada en un ESB, que permitan analizar y obtener información en tiempo real de los mensajes que circulan a través de la misma, generando un Data Warehouse relacional con el fin de almacenar dicha información.

Para llevar a cabo el objetivo principal se plantean los siguientes objetivos más específicos:

- Análisis detallado de la problemática planteada, identificando escenarios de interés.
- Relevamiento de soluciones existentes que resuelvan algún problema similar y de otra información que sea de ayuda para el planteo de una nueva solución.
- Propuesta de solución que contenga entre otras cosas una descripción conceptual y la arquitectura general de la misma.
- Implementación de un prototipo de la solución propuesta, teniendo como única restricción que debe ser desarrollado en Java y debe utilizar JBoss como servidor de aplicaciones.
- Definición de al menos dos casos de uso que permitan realizar pruebas para verificar el correcto funcionamiento del prototipo implementado.

1.2 Aportes del proyecto

Los principales aportes del proyecto son:

- **Relevamiento de soluciones que resuelven el problema de realizar la carga de un Data Warehouse en tiempo real.** En particular se analizaron dos enfoques: *"On the fly*" y "*Near* real *time Data Warehouse*".
- **Propuesta de solución para soportar la construcción del Data Warehouse a partir de los datos de interés indicados por el usuario, y la carga dinámica del mismo a partir de los mensajes que circulan a través del ESB.** La propuesta consta de un diseño conceptual, y del diseño de una arquitectura para soportar dicha propuesta.
- **Implementación de un prototipo que utiliza la arquitectura diseñada**. El prototipo fue desarrollado en Java utilizando JBoss como servidor de aplicaciones, y resuelve correctamente tres casos de uso definidos para comprobar el correcto funcionamiento del mismo.
- **Evaluación sobre las herramientas utilizadas y limitaciones del prototipo.** Se realizó un breve análisis sobre las herramientas utilizadas durante la implementación del prototipo, detallando los problemas encontrados sobre su uso, así como las limitaciones del mismo.

Si bien se buscó obtener una solución genérica para el problema descrito, se consideraron requerimientos específicos del Ministerio de Trabajo y Seguridad Social que ofició como cliente del proyecto.

1.3 Organización del Documento

Luego de este capítulo de introducción, se ha organizado el documento de la siguiente forma:

En el Capítulo 2, se profundiza sobre el marco teórico correspondiente a la problemática planteada en el proyecto. En este punto también se incluye la importancia que tiene contar con un Data Warehouse en tiempo real.

En el Capítulo 3, se analiza en profundidad la problemática planteada, se presentan los requerimientos del cliente, así como también se definen conceptos que serán utilizados en la solución propuesta. Además se muestra el relevamiento de trabajo relacionado.

En el Capítulo 4, se presenta la solución propuesta, comenzando con una descripción conceptual, luego se realiza una descripción de la interacción de los principales componentes de la misma. Además se presenta el proceso de construcción del Data Warehouse y se detallan las principales decisiones de diseño.

En el Capítulo 5, se brindan los detalles de implementación de las principales funcionalidades de la solución, así como las limitaciones presentes en el prototipo construido.

En el Capítulo 6, se definen los tres casos de estudio que se tomaron como referencia para evaluar el prototipo implementado. Se realizaron pruebas sobre el sistema, presentando el resultado obtenido en forma de reportes.

Finalmente en el Capítulo 7 se exponen las conclusiones obtenidas luego de finalizado el proyecto junto con los posibles trabajos a futuro.

2 Marco Teórico

En este capítulo se detallan con más profundidad los fundamentos teóricos que aplican a la problemática planteada para este proyecto y que serán de utilidad para comprender de mejor forma la solución planteada. Dichos fundamentos son: Extensible Markup Language, Service Oriented Architecture, Web Services, Enterprise Service Bus, Complex Event Processing y Data Warehouse.

2.1 Extensible Markup Language (XML)

Es un lenguaje de etiquetado cuya principal función es describir datos. El lenguaje es simple, pero a la vez estricto ya que juega un papel fundamental en el intercambio de una gran variedad de datos [2]. Un ejemplo de documento XML es el que se muestra en la Figura 2, donde se describe un objeto llamado *BankAccount* que tiene cinco atributos: *Number*, *Type*, *OpenDate*, *Balance* y *AccountHolder*.

```
<?xml version="1.0" standalone="yes"?>
<BankAccount>
      <Number>1234</Number>
      <Type>Checking</Type>
      <OpenDate>11/04/1974</OpenDate>
      <Balance>25382.20</Balance>
      <AccountHolder>
            <LastName>Singh</LastName>
            <FirstName>Darshan</FirstName>
      </AccountHolder>
</BankAccount>
```

Figura 2: Ejemplo de XML [3].

Las tecnologías XML son un conjunto de módulos que brindan diferentes facilidades a la hora de manipular documentos XML. Entre algunas de estas tecnologías se encuentra por ejemplo XML Path Language (XPath).

2.1.1 XML Namespace

Cuando se escriben datos en un documento XML es posible utilizar un vocabulario previamente definido para realizar la descripción de dichos datos. Para utilizar un vocabulario externo, es necesario realizar una importación de dicho vocabulario mediante la etiqueta "*xmlns*". Si en un documento XML fuera necesario importar más de un vocabulario, existe la posibilidad de que dichas importaciones introduzcan algún conflicto de nombres, dado que un elemento puede ser definido de manera diferente en más de un vocabulario.

XML namespace es un método que surge como forma de evitar el conflicto de nombres en un documento XML. Existen dos formas de definir un XML namespace: por prefijo o por defecto. Para definir un namespace por prefijo basta con incluir el atributo xmlns:prefijo="URI" en la etiqueta inicial de cualquier elemento. Luego se deberá utilizar en cada elemento hijo alguno de los prefijos definidos. Si por el contrario sólo se incluye el atributo xmlns="URI", el namespace es definido por

defecto. Esto evita incluir prefijos en los elementos hijos [2]. Un ejemplo de dichas definiciones se puede encontrar en la Figura 3, en la que se muestra un ejemplo en donde se busca diferenciar los elementos de una tabla, tanto mediante el uso de namespace con prefijos -a la izquierda de la imagen- como namespace por defecto -a la derecha de la imagen-.

Figura 3: Definición de namespace por prefijo (izquierda) y por defecto (derecha).

2.1.2 XML Path Language (XPath)

Este estándar define un lenguaje de consulta que permite buscar información de interés dentro de un documento XML. Su denominación se debe a que hace uso de una notación de caminos para navegar a través de la estructura jerárquica del mismo. XPath opera sobre la estructura lógica de un documento XML, modelando el mismo como un árbol de nodos [4]. En la Figura 4 se ilustra el mapeo de un documento XML a un árbol de nodos. Como puede verse dicha estructura comienza con un nodo raíz, despliega una serie de elementos intermedios y finaliza con varios elementos finales.

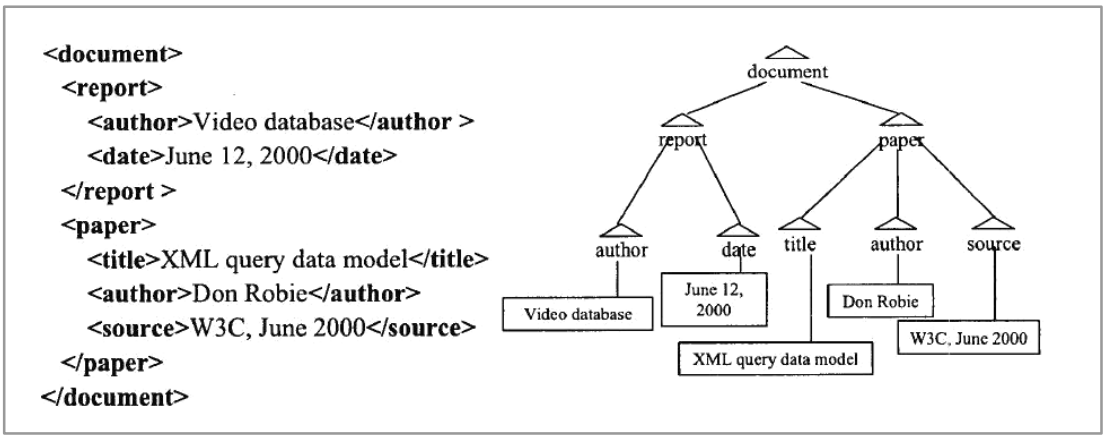


Figura 4: Mapeo de un documento XML a árbol de nodos [3].

En XPath se utilizan expresiones para navegar a través de los nodos. Al ejecutar estas expresiones es posible obtener tanto un nodo como un conjunto de nodos. A su vez cada nodo tiene asociado un valor que puede ser una cadena de caracteres, un

número o un valor que indica verdadero o falso. Algunos ejemplos de expresiones válidas en XPath se pueden ver en la Figura 5.

Expresión	Descripción
nodename	Selecciona todos los nodos de nombre nodename
/	Selecciona a partir de la raíz del documento
//	Selecciona nodos en cualquier lugar del documento
.	Selecciona el nodo actual
..	Selecciona el nodo padre del nodo actual
@	Selecciona atributos

Figura 5: Tabla de expresiones en XPath [5]

En la Figura 6 se muestra un ejemplo de diferentes resultados que se pueden obtener utilizando expresiones XPath sobre el XML presentado en la Figura 3. Se incluye del lado izquierdo la expresión utilizada y a la derecha el resultado correspondiente a su ejecución.

/document/report/author	<author>Video database</author>
//author	<author>Video database</author> <author>Don Robie</author>
/document/paper/*	<title>XML query data model</title> <author>Don Robie</author> <source>W3C,June 2000</source>

Figura 6: Ejemplo de expresiones utilizando XPath.

2.2 Service Oriented Architecture (SOA)

Es un estilo de Arquitectura de Software basado en la definición de servicios reutilizables, con interfaces públicas bien definidas, donde los proveedores y consumidores de servicios interactúan en forma desacoplada para realizar procesos de negocio [6].

La implementación de una SOA es posible, entre otras cosas, gracias a la encapsulación de servicios brindada por las tecnologías de web services, así como al papel que ocupan los ESB como plataformas de integración. Esto se logra debido a que las tecnologías de web services permiten encapsular los servicios mediante un estándar aceptado por todos los fabricantes y proveedores. Por otro lado, los ESB brindan la posibilidad de tener una red de sistemas dispares interactuando como un

sistema unificado corporativo y resolviendo las diferencias de hardware, software, redes y localización [7].

Dentro de los beneficios obtenidos en la implementación de una SOA se puede destacar que permite reutilizar aplicaciones existentes mediante encapsulación en servicios. Por otro lado, simplifica la adaptación de otros sistemas existentes y aumenta la interoperabilidad entre sistemas, permitiendo tanto la externalización como la prestación de servicios. Por último, ayuda a desacoplar el desarrollo de servicios y procesos [7].

2.3 Web Services

Un Web Service es un software diseñado para brindar mecanismos que faciliten la interoperabilidad entre dos o más sistemas en una red. Se identifica por una URI especificada a través de *XML Namespace*, y expone la lógica de negocio, datos y procesos por medio de una interfaz pública bien definida descrita mediante artefactos XML. Esto permite a distintas aplicaciones de diferentes orígenes, comunicarse entre sí de forma sencilla mediante el intercambio de documentos XML [8].

Los Web Services proveen entonces una forma de integrar aplicaciones basándose en estándares como XML, Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) entre otros, sobre los protocolos de Internet [9]. En las siguientes sub secciones se brindan más detalles de los mismos.

2.3.1 Web Services Description Language (WSDL)

Es un formato XML que se utiliza para describir Web Services. WSDL describe la interfaz pública de los mismos que incluye la forma de comunicación, es decir los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con las operaciones que expone el servicio al cual hace referencia [10].

Las operaciones y mensajes que soporta se describen primero de manera abstracta mediante las etiquetas *types*, *message* y *portType*. Esta información no depende de la tecnología utilizada. Luego se agrega la información concreta de la implementación mediante las etiquetas *bindings* y *service*. En la Figura 7 se muestra cómo se compone la estructura de un WSDL.

```
<definitions>

    <types>
        Aquí se definen los data types usados por el servicio.
    </types>

    <message>
        Aquí se definen los elementos del mensaje. Cada mensaje puede contener una
        serie de data types que fueron definidos en la etiqueta anterior.
    </message>

    <portType>
        Aquí se definen las operaciones permitidas y los mensajes intercambiados por
        las mismas.
    </portType>

    <binding>
        Aquí se especifican los protocolos de comunicación usados.
    </binding>

    <service>
        Aquí se especifica el conjunto de puertos y dirección de los mismos.
    </service>

</definitions>
```

Figura 7: Estructura de un WSDL

2.3.2 Simple Object Access Protocol (SOAP)

Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de un intercambio de datos XML [11]. En la Figura 8 se muestra la estructura de un mensaje SOAP.

```
<soap:Envelope xmlns:soap="..." soap:encodingStyle="...">

    <soap:Header>
    </soap:Header>

    <soap:Body>
    </soap:Body>

</soap:Envelope>
```

Figura 8: Estructura de un mensaje SOAP

La etiqueta Envelope es obligatoria. La misma debe contener el atributo xmlns:soap (es un namespace que define el XML como un mensaje SOAP) junto con el atributo soap:encodingStyle. Además el envelope puede contener dos etiquetas más: la etiqueta Header que es opcional y la etiqueta Body que es requerida.

2.3.2.1 SOAP Header

Es un elemento opcional que en caso de estar presente debe ser el primer hijo del elemento Envelope [12]. Es una forma de agregar información adicional a la que se encuentra en el elemento Body, por ejemplo información que hace referencia a detalles específicos de la aplicación como seguridad o contexto transaccional. Un ejemplo de Header es el que se muestra en la Figura 9.

```
<soap:Header>
    <m:Trans xmlns:m="URItrans">234</m:Trans>
</soap:Header>
```

Figura 9: Ejemplo de Header en un mensaje SOAP [12]

2.3.2.2 SOAP Body

Este elemento actúa como el contenedor de información que el emisor le envía al receptor. Debe estar obligatoriamente en el mensaje y su ubicación es después del Header en caso de que éste exista, o en caso contrario como primer elemento del Envelope [13]. Es el encargado de mantener la información relativa a la invocación o respuesta del servicio. En la Figura 10 se muestran dos ejemplos de Body correspondientes a una petición y su correspondiente respuesta.

```
<soap:Body>
    <m:getPrice xmlns:m="URI1">
        <m:Item>Manzanas</m:Item>
    </m:getPrice>
</soap:Body>

<soap:Body>
    <m:getPriceResponse xmlns:m="URI1">
        <m:Price>18.0</m:Price>
    </m:getPriceResponse>
</soap:Body>
```

Figura 10: Ejemplos de Body en mensajes SOAP [15]

2.4 Enterprise Service Bus (ESB)

Un ESB es una plataforma de integración basada en estándares que combina mensajería, web services, transformación de datos y ruteo inteligente para conectar y coordinar de forma confiable la interacción de un gran número de aplicaciones con integridad transaccional [14].

Un ESB no implementa en sí mismo una arquitectura orientada a servicios (SOA), sino que proporciona características mediantes las cuales sí se puede implementar [14]. Si bien es cierto que la utilización de un ESB implica ajustarse a una arquitectura determinada, el término “ESB” casi siempre se refiere a la infraestructura de software que hace posible tal arquitectura y, en esencia se considera al ESB como una plataforma para realizar una arquitectura orientada a servicios [15]. En la Figura 11 se muestra cómo se posiciona un ESB como plataforma de integración, dentro de un sistema de información.

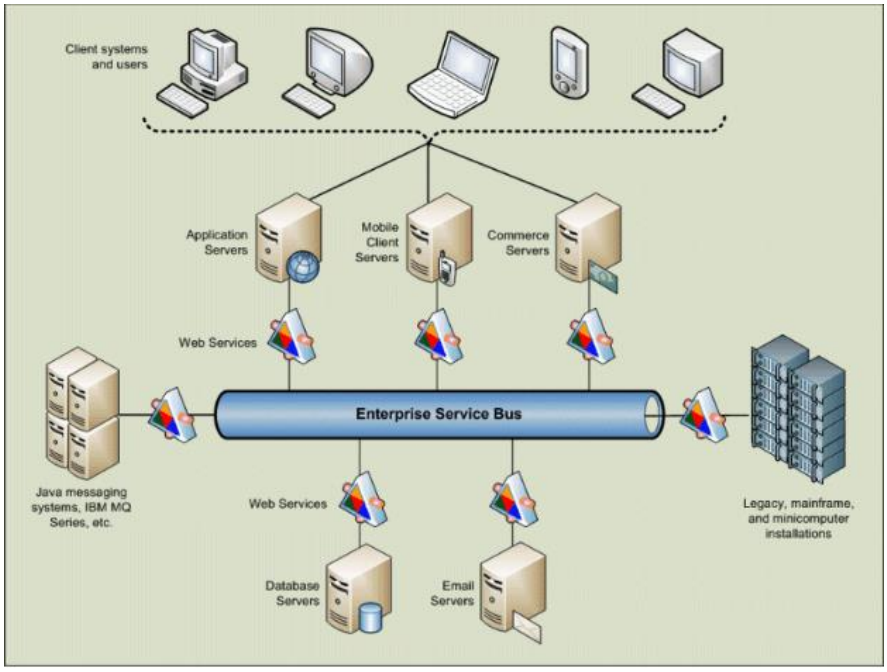


Figura 11: Representación gráfica de un ESB [16]

2.4.1 Características de un ESB

Un ESB facilita la interoperabilidad entre aplicaciones que utilizan diferentes lenguajes de programación, como por ejemplo Java y .Net. Da soporte a las comunicaciones entre diferentes aplicaciones utilizando XML como lenguaje de comunicación normalizado. Cuenta también con patrones de intercambio de mensajes y presenta modelos de seguridad que permiten autorizar, autenticar y auditar dichos intercambios. Brinda la posibilidad de manipular las comunicaciones mediante herramientas de enrutamiento o transformación de mensajes de modo condicional, basándose en reglas definidas. Además permite encolar y retener mensajes si las aplicaciones no están temporalmente disponibles, entre otras características [17].

2.4.2 Capacidades de interés

Entre todas las capacidades presentes en un ESB, se han detectado algunas que pueden ser de interés a la hora de resolver la problemática planteada. Este es el caso de la auditoría, ruteo de mensajes y virtualización de servicios, que serán detallados a continuación.

2.4.2.1 Auditoría de mensajes

Los ESB pueden registrar peticiones y respuestas con propósitos de auditoría, enviando alertas cuando se produzcan determinadas condiciones. Esta capacidad se corresponde con el patrón de monitorización de sistema denominado *wiretap*, definido en [18]. La importancia de dicho patrón, es que es posible eventualmente almacenar los mensajes de petición y respuesta, de manera de poder procesar dicha información de forma independiente sin afectar el tiempo de respuesta del servicio.

2.4.2.2 Ruteo de mensajes

Los ESB implementan varios patrones sobre el ruteo de mensajes, pero particularmente el que resulta de mayor interés es el denominado ruteo en base a contenido, también definido en [18]. Como lo dice su nombre, este patrón rutea los mensajes hacia diferentes destinos dependiendo del contenido de los mismos. Dos ejemplos sobre criterios de ruteo son el chequeo de la existencia de ciertos campos, y la búsqueda de campos con valores específicos.

2.4.2.3 Virtualización de servicios

Se refiere a la abstracción de servicios físicos a través de un servicio intermedio, como puede verse en la Figura 12 (Servicio Virtual). Con la virtualización de servicios, el servicio intermedio interactúa con el consumidor del servicio (la aplicación cliente que invoca al servicio), y oculta la dirección física del proveedor del mismo (el web service que provee las funcionalidades) [19].

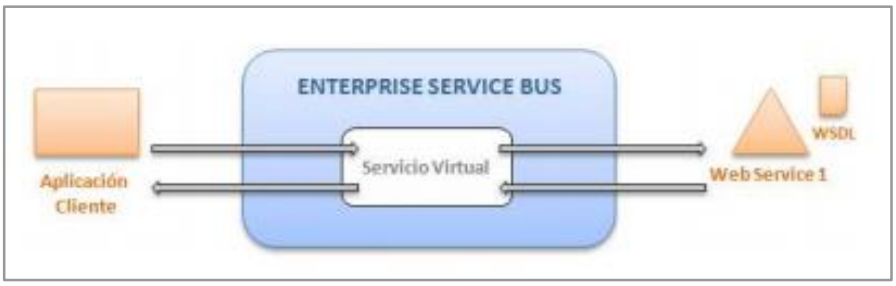


Figura 12: Representación gráfica de la virtualización de servicios [20].

2.5 Complex Event Processing (CEP)

El procesamiento de eventos complejos hace referencia al conjunto de técnicas utilizadas para detectar patrones complejos, dentro de una serie de eventos simples que ocurren en tiempo real [21].

En los Sistemas de Información se entiende como evento simple a un registro de un hecho que ocurre en determinado sistema o dominio. A su vez un evento complejo se define como un evento que resume, representa o denota un conjunto de eventos. Particularmente un evento complejo puede ser complejo para un sistema pero simple para otro [22].

El objetivo de CEP es realizar operaciones sobre dichos eventos, incluyendo la creación, captura, descarte y transformación de los mismos. La principal ventaja es que facilita la obtención de información procedente de diferentes sistemas (ambiente distribuido) y es capaz de aplicar reglas en tiempo real.

Dentro del procesamiento de eventos complejos se distinguen tres elementos. Estos son los productores de eventos, el procesamiento intermedio, y los consumidores de eventos [22].

En la Figura 13 se muestra un diagrama que explica el funcionamiento de los motores CEP. A continuación se detallan los elementos e interacciones que forman parte de este diagrama.

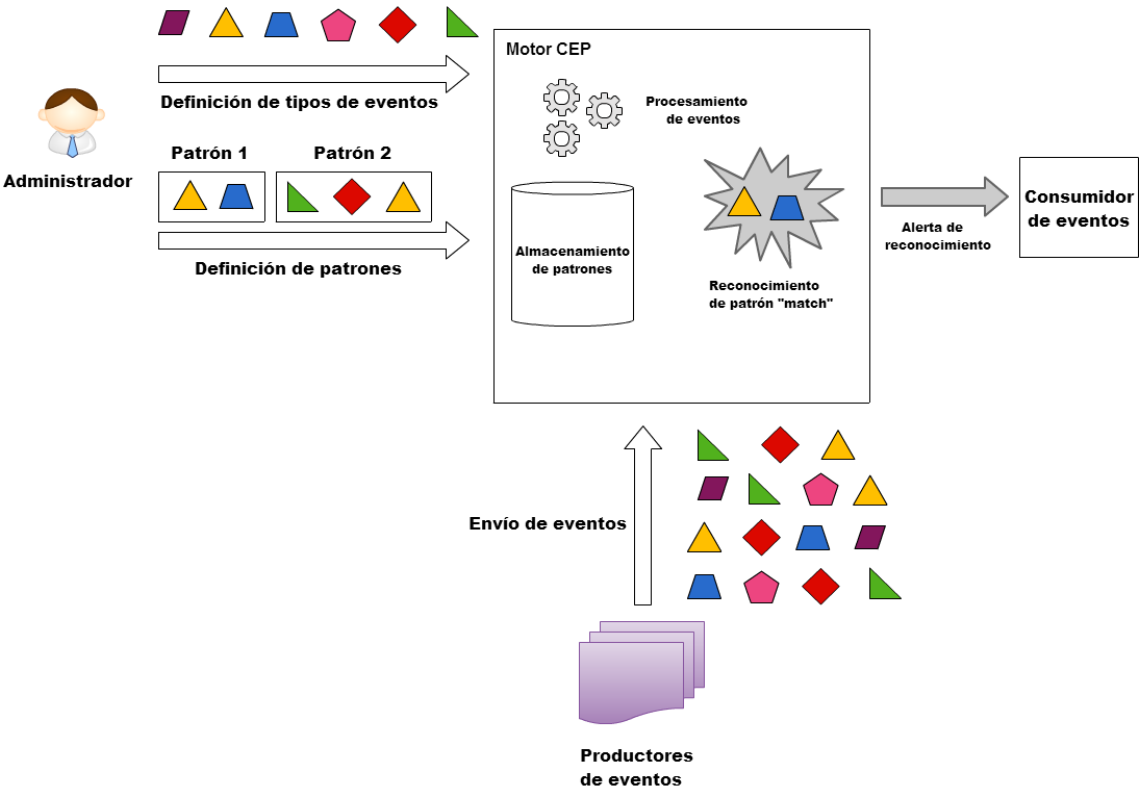


Figura 13: Diagrama de arquitectura de CEP.

Administrador: Denota al responsable de definir los eventos y patrones de interés que el motor debe monitorear, por ejemplo un usuario que hace uso de una aplicación que interactúa con el motor.

Definición de tipo de eventos: Como primer paso es necesario definir los eventos que utilizará el motor. Los eventos pueden ser representados en diferentes formatos de datos, por ejemplo tuplas, documentos XML u objetos de un lenguaje. Como los mismos ocurren en un momento de tiempo, será necesario guardar dicho instante de tiempo o un intervalo en el que ocurrió, además de poseer un identificador de evento [23].

Definición de patrones: Para poder detectar patrones que ocurren sobre los eventos que llegan al motor, se realizan consultas sobre los mismos. Dichas consultas son especificadas en un Event Query Language (EQL), que es un lenguaje de consulta de alto nivel [24]. Este lenguaje provee operadores lógicos (como por ejemplo or, and, not), y operadores temporales que permiten especificar tanto intervalos de tiempo, como el orden en que ocurren los eventos. Cada uno de los patrones definidos es almacenado dentro del motor CEP.

Consumidor de eventos: Denota a un componente que realiza una acción cuando el motor reconoce alguno de los patrones almacenados.

Productores de eventos: Representa a las fuentes que proveen los eventos que serán enviados al motor, por ejemplo una base de datos en donde cada tupla representa un evento, o un componente que envía eventos representados como un objeto de un lenguaje.

Procesamiento de eventos y "match": El motor se encarga de chequear si cada evento que le llega satisface alguno de los patrones almacenados en el mismo. En caso de que esto ocurra se produce lo que se denomina *match*, se toma alguna acción como imprimir un mensaje en pantalla o alertar a los consumidores de eventos.

Un ejemplo para ilustrar el funcionamiento descrito es el caso en que se desea controlar una alarma anti incendio a través de un sensor que mide la temperatura. En caso de que la temperatura detectada en dos mediciones consecutivas supere los 50°C se debe activar la alarma junto con los rociadores de agua dispersos por toda la casa.

En la Figura 14 se muestra un pseudocódigo de cómo se debe especificar el patrón que se utilizará en el ejemplo, junto con la acción a tomar en caso de detectarse un *match*. El operador "->" denota un operador temporal que indica precedencia en el tiempo. En este caso se traduce a un *EventoTemperatura* con medición superior a 50°C, seguido de otro con la misma característica.

```
Patrón

     QueryCEP = SELECT *
                FROM EventoTemperatura e1,e2
                WHERE (e1.medicion > 50) -> (e2.medicion > 50));
```

```
Acción

     if (QueryCEP) then
         encenderRociadores();
```

Figura 14: Pseudocódigo del patrón y acción a tomar para el ejemplo utilizando CEP.

2.6 Data Warehouse (DW)

Un Data Warehouse es una base de datos corporativa que se caracteriza por integrar y depurar información de una o más fuentes, para luego procesarla permitiendo su análisis desde diversas perspectivas. Este concepto puede verse ilustrativamente en la Figura 15. La creación de un Data Warehouse representa en la mayoría de las ocasiones el primer paso desde el punto de vista técnico, para implantar una solución completa y fiable de Business Intelligence [25].

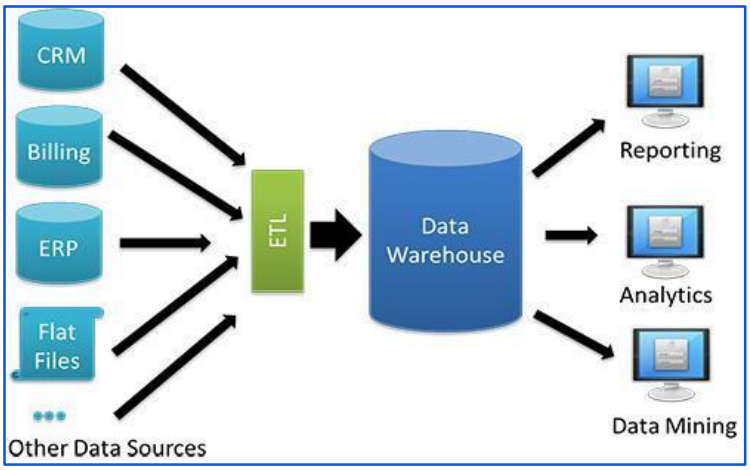


Figura 15: Ejemplo gráfico de un DW [26].

El término Data Warehouse fue acuñado por primera vez por Bill Inmon, y se traduce literalmente como almacén de datos. No obstante según definió el propio Bill Inmon, un Data Warehouse además se caracteriza por ser:

- **Integrado**: los datos almacenados en el Data Warehouse deben integrarse en una estructura consistente. Esto implica que deben ser eliminadas las inconsistencias existentes entre los datos de las diversas fuentes utilizadas.
- **Temático**: para facilitar el acceso y entendimiento a los datos, los mismos se organizan por temas en torno a los contenidos principales de la organización.
- **Histórico**: el tiempo es parte implícita de la información contenida en un Data Warehouse. Por lo tanto, el Data Warehouse se carga con los distintos valores que toma una variable en el tiempo. Esto permite entre otras cosas realizar análisis de tendencias.
- **No volátil**: la información almacenada en un Data Warehouse existe para ser leída, no modificada. Esto implica que la actualización del Data Warehouse refiere únicamente a la incorporación de nuevos datos.

Otra característica de un Data Warehouse es que contiene metadatos. Los metadatos permiten saber la procedencia de la información, su periodicidad de carga, su forma de cálculo, entre otros. Además sirven para dar soporte a los diferentes actores del Data Warehouse, ya sea en el acceso a los datos, la generación de reportes o la gestión de la información [27].

Por último, es importante destacar la forma en que la información almacenada en un Data Warehouse es presentada al usuario. Existen varias herramientas para realizar el análisis de los datos de un Data Warehouse entre ellas las herramientas OLAP (Online Analytical Processing). En un análisis OLAP la información es considerada en un contexto multidimensional. Esto implica que la información no es presentada al usuario a través de tablas, sino que es presentada en una forma matricial denominada *Cubo.* Esto se muestra más en detalle en las siguientes secciones. Algunos ejemplos de herramientas OLAP son las herramientas brindadas por la suite Pentaho [28].

2.6.1 Proceso de construcción de un Data Warehouse

El proceso de construcción de un Data Warehouse consta de una serie de etapas que se muestran en la Figura 16. Dicha figura fue construida en base al esquema presentado en [29].

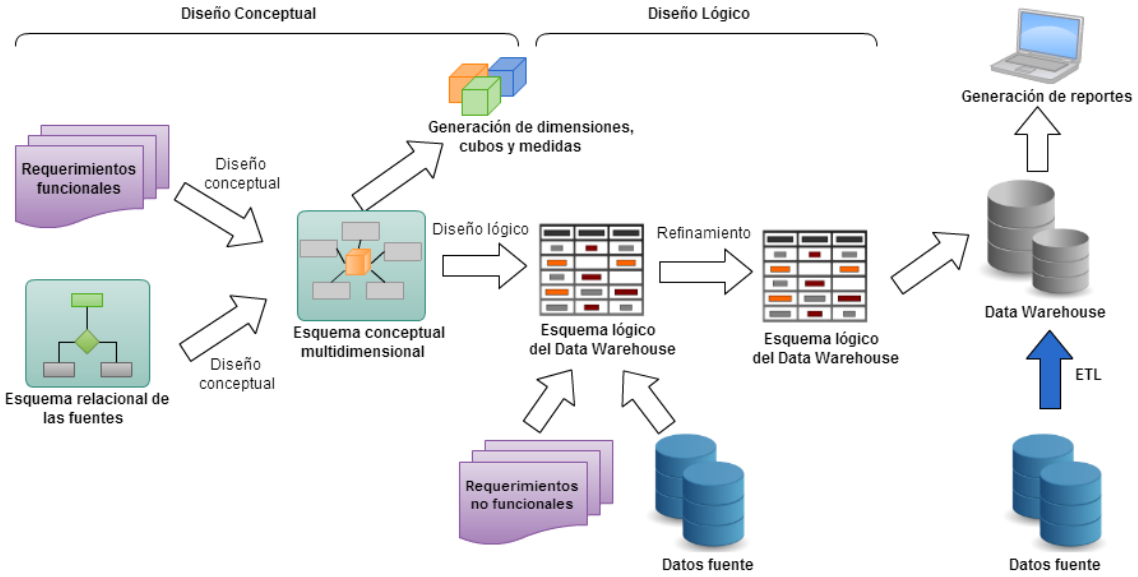


Figura 16: Proceso de construcción de un Data Warehouse.

El primer paso consiste en realizar un diseño conceptual de la solución. A partir de los requerimientos funcionales iniciales, y las fuentes de datos (bases, archivos, esquema relacional de las fuentes, etc.) se construye un esquema conceptual de las estructuras multidimensionales que formarán parte del Data Warehouse. En esta etapa, luego de múltiples refinamientos, se genera el diseño de las dimensiones, cubos y medidas del mismo -estos artefactos se detallan en la siguiente sección-.

Luego se pasa a realizar el diseño lógico de la solución. Esta etapa parte del diseño conceptual generado y de los requerimientos no funcionales, y tiene por objetivo la generación del esquema lógico del Data Warehouse. Esto significa que una vez finalizada esta etapa se contará con el diseño de la base de datos que almacenará la información del Data Warehouse. Se comienza realizando un mapeo del esquema conceptual multidimensional a un primer esquema lógico tomando en cuenta los requerimientos no funcionales y las fuentes de datos. Posteriormente se realiza un refinamiento de dicho esquema teniendo en cuenta entre otras cosas los volúmenes de datos que se van manipular.

Finalmente se realiza la carga del Data Warehouse a través de procesos de ETL y se procede a la generación de reportes para el usuario final [29] y [30].

2.6.2 Modelos multidimensionales

Como se detalló anteriormente, la información en un Data Warehouse se presenta siguiendo un enfoque multidimensional. Esto se debe a que se busca representar datos en una forma cercana a la intuición de un usuario. En un contexto

multidimensional la información es presentada al usuario en una forma matricial denominada *Cubo* o Hipercubo [29].

Los ejes de cada cubo son denotados como *Dimensiones*. Una dimensión consiste en un criterio de agrupamiento de datos en base a un criterio analítico. A su vez los valores de la información perteneciente a una dimensión se organizan en *Jerarquías* que presentan diferentes niveles de navegación. Esta organización es requerida para definir el agrupamiento y la navegación de los datos para dicha dimensión. En la Figura 17 se muestra un ejemplo de representación de una dimensión *Tiempo* que presenta una única jerarquía, con los diferentes niveles *fecha*, *hora*, *día*, *mes*, *año*.

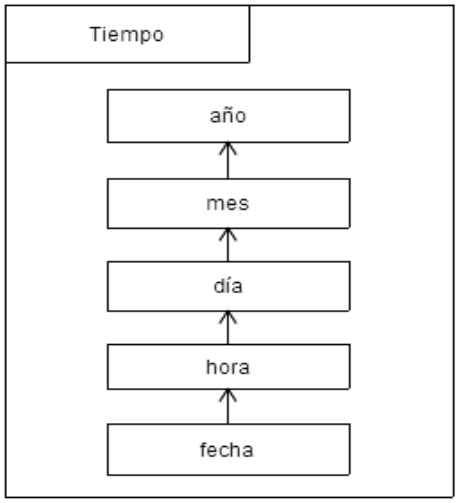


Figura 17: Ejemplo de Dimensión Tiempo.

Por otro lado, los valores de las celdas de un *Cubo* son denominados *Medidas.* Las medidas son los valores que serán analizados y representan a los datos asociados a cruces de las dimensiones de la realidad que se esté modelando [29].

En la Figura 18 se presenta un ejemplo de modelo multidimensional. En este ejemplo se tiene una dimensión *Modelo* que mantiene información sobre modelos de autos y otra dimensión *Color* que mantiene información sobre colores de los autos. Las medidas en este modelo se corresponden con la cantidad de autos disponibles de cada color para los modelos presentados.

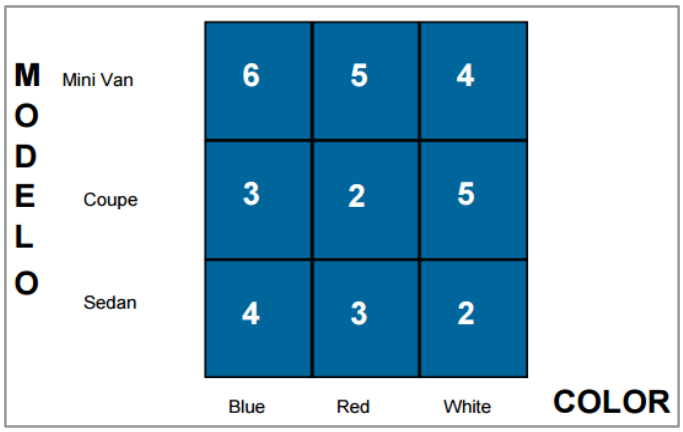


Figura 18: Cubo formado por las dimensiones Modelo y Color [29].

Una forma de consultar información sobre un modelo multidimensional es a través de consultas MDX (Multidimensional Expresión). MDX es un lenguaje que permite realizar definiciones, consultas y cálculos sobre los datos de un modelo multidimensional, proporcionando a un usuario instruido en el mismo, una forma de interactuar con una herramienta OLAP [29].

2.6.3 Extracción, Transformación y Carga (ETL)

Es una parte del proceso de construcción de un Data Warehouse. Consiste en las siguientes tres etapas:

- **Extracción:** obtención de información de las distintas fuentes tanto internas como externas.
- **Transformación:** filtrado, limpieza, depuración, homogeneización y agrupación de la información.
- **Carga:** organización y actualización de los datos y metadatos en la base de datos.

2.6.4 Esquemas de un Data Warehouse

En un Data Warehouse la información puede ser físicamente representada siguiendo diferentes esquemas. Los más populares son el esquema estrella y el esquema copo de nieve [31].

En ambos esquemas las dimensiones son presentadas como tablas -tablas de dimensión- en una base de datos y los cruces entre las diferentes dimensiones son representados en tablas denominadas tablas de hechos.

En un esquema estrella las diferentes tablas de dimensión son representadas en torno a una única tabla de hechos. En la tabla de hechos se agregan las referencias a las diferentes tablas de dimensión y los atributos destinados a representar las diferentes métricas relacionadas con el objeto de análisis. Por otro lado, en las tablas de dimensión se almacena la información relacionada a los diferentes niveles definidos en las jerarquías de la dimensión. Se mantiene toda la información en un único nivel de representación física.

Un esquema copo de nieve es similar a un esquema estrella, con la excepción de que la información de las tablas de dimensión se encuentra normalizada en múltiples tablas. Es decir la información de las jerarquías de la dimensión, se presenta en varios niveles a través de tablas relacionadas [32].

Ambos esquemas deben su nombre a la forma que presentan en su representación gráfica.

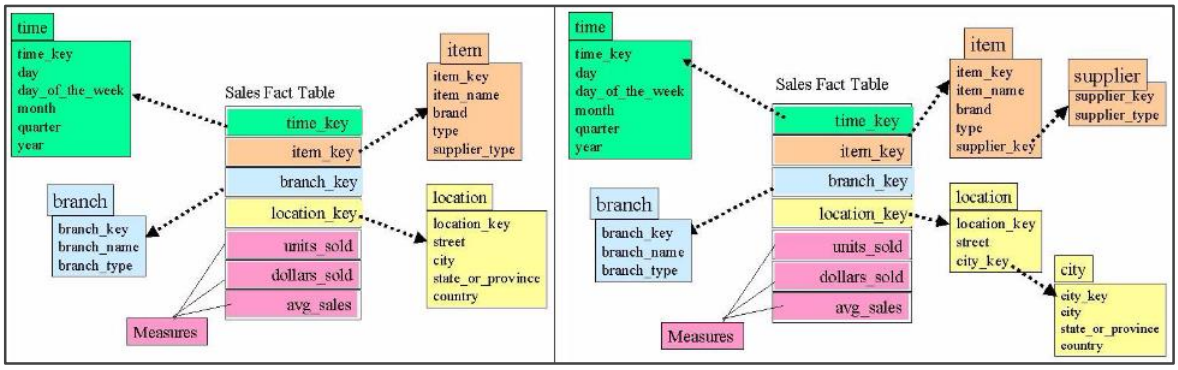


Figura 19: Comparación entre esquema estrella (izquierda) y esquema copo de nieve (derecha).

En la Figura 19 se tiene una representación para cada esquema de un modelo que cuenta con una dimensión *time*, una dimensión *ítem*, una dimensión *branch* y una dimensión *location*. Se puede ver cómo la tabla de hechos -en la imagen fac_table - es igual en ambos esquemas, sin embargo es claro visualizar que lo mismo no sucede para las tablas dimensión. Esto es debido a que para el esquema estrella -izquierda- toda la información de las diferentes dimensiones se presenta en un mismo nivel, mientras que para el esquema copo de nieve -derecha- las tablas de dimensión se encuentran *normalizadas*. Es decir que en el esquema copo de nieve, se mantienen diferentes tablas para los distintos niveles de las jerarquías de la dimensión. Así por ejemplo, para la dimensión *location* se mantiene una tabla para el nivel *location* y otra para el nivel *city*.

2.6.5 Data Warehouse en tiempo real

Tradicionalmente la actualización de la información en los Data Warehouses se realiza periódicamente -al final de un día, semana o mes- y proporcionando una *captura* de los datos de la organización en un determinado momento en el tiempo.

El proceso de carga tradicional ha prevalecido en gran parte debido a las dificultades inherentes a los procesos de integración entre los sistemas en producción y los Data Warehouses. Por ejemplo los datos pueden no estar disponibles todo el tiempo lo cual imposibilita la carga en tiempo real. No obstante este proceso presenta el inconveniente de que la toma de decisiones se realiza sobre datos que pueden ser potencialmente obsoletos.

De aquí radica la importancia de tener la información actualizada hasta la fecha, ya que los datos son un producto perecedero: mientras más antiguos sean menos relevantes son.

Data Warehouse en tiempo real, es un término que hace referencia a la velocidad con la que se actualiza la información en el Data Warehouse.

Esta técnica pretende eliminar la brecha de disponibilidad de datos para mejorar el proceso de toma de decisiones. Se intenta que la nueva información que ingresa quede disponible para ser consultada de forma casi inmediata [33].

3 Análisis

En este capítulo se realiza un análisis en profundidad de la problemática planteada. Esta problemática gira en torno a un escenario propuesto por el Ministerio de Trabajo y Seguridad Social (MTSS). El análisis realizado se presenta como resultado de múltiples reuniones con un representante del MTSS. El capítulo comienza por un análisis conceptual del escenario planteado, luego un análisis de requerimientos y para finalizar un relevamiento sobre trabajos relacionados.

3.1 Análisis conceptual

Como se muestra en la Figura 20 se parte de un escenario compuesto por un conjunto de aplicaciones que exponen su funcionalidad a través de un conjunto de servicios virtuales alojados en un Enterprise Service Bus. Por otro lado existen en dicho escenario aplicaciones que simplemente utilizan los servicios virtuales para comunicarse con otras aplicaciones.

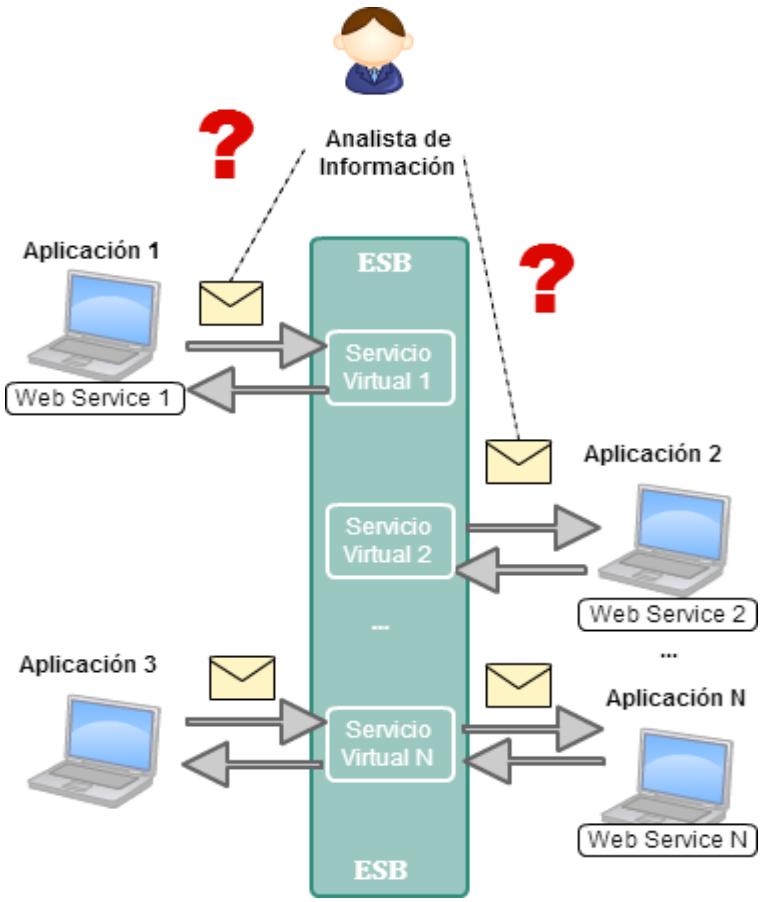


Figura 20: Escenario planteado

Se tiene además un usuario *analista de información* cuya intención es obtener información sobre algunos de los mensajes que intercambian dichas aplicaciones.

Se deberá proveer al usuario una forma de indicar cuáles de los servicios del sistema le interesan.

A partir de aquí se pueden diferenciar dos casos: el primero es cuando interesa analizar sólo los mensajes de un servicio, y el segundo cuando se quieren analizar los mensajes de varios servicios en conjunto que estén relacionados entre sí.

Como resultado del análisis de ambos casos se introducen los siguientes conceptos que son de importancia para comprender tanto la solución como el resto de la documentación:

- **DataSet**: se define como un conjunto de operaciones de un servicio. Un DataSet permite almacenar información para los mensajes correspondientes a cada una de dichas operaciones dentro de un servicio. Cada uno de esos mensajes son recibidos en las diferentes invocaciones a las operaciones mencionadas.
- **Vista**: se define como un conjunto de DataSets. Debe ser posible definir condiciones (en base a operadores lógicos *AND* y *OR*), que relacionan los DataSets pertenecientes a la misma. Dicha relación se da a través de los diferentes atributos de los mensajes correspondientes a las operaciones indicadas para cada DataSet.

Se debe entonces, brindar a un usuario la posibilidad de definir diferentes DataSets a partir de los servicios de interés. Además definir Vistas a partir de dichos DataSets y de condiciones que relacionen los mismos.

Por último se deben implementar mecanismos que permitan crear las estructuras del Data Warehouse en base a la información brindada por el usuario, y además realizar la carga del mismo en tiempo real utilizando la información de los mensajes intercambiados y capturados.

En la Figura 21 se muestra el diagrama conceptual correspondiente al análisis presentado.

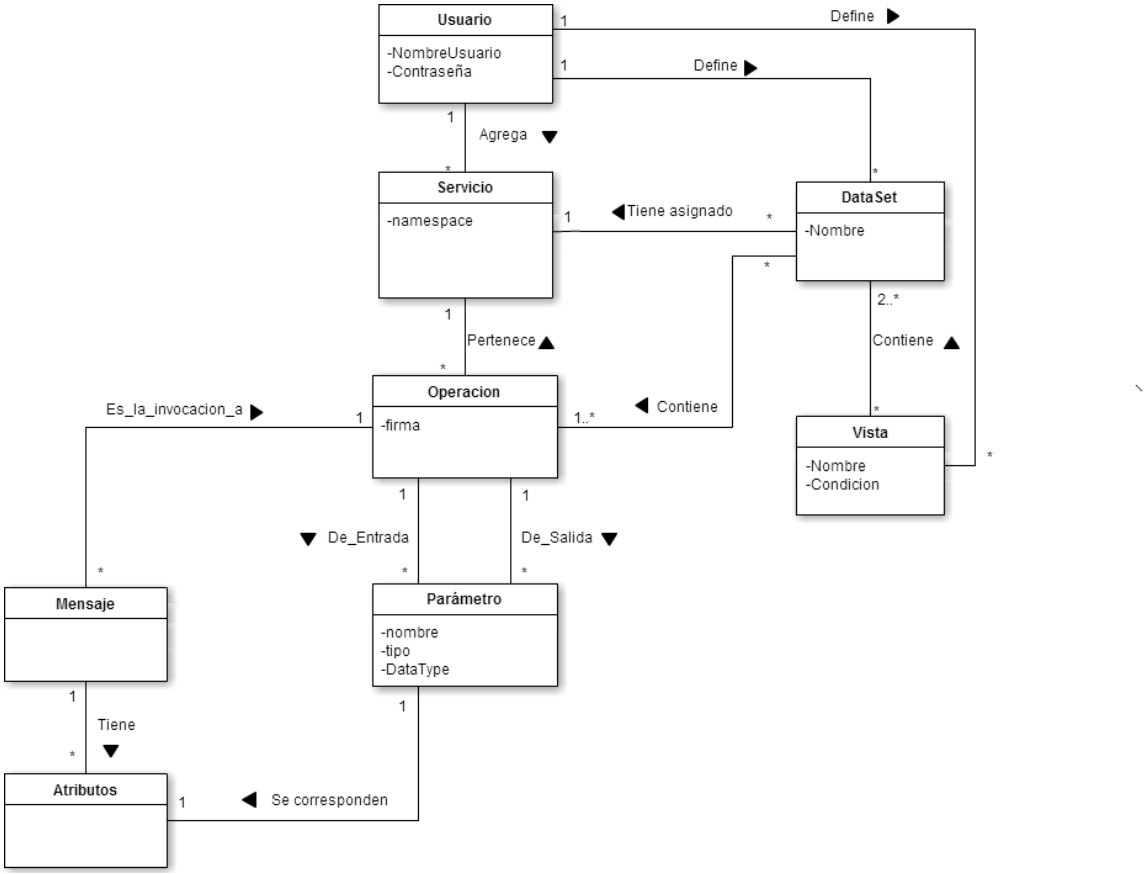


Figura 21: Diagrama conceptual de la realidad a modelar

De dicho análisis se desprenden las siguientes restricciones en lenguaje natural:

1. Un DataSet solo puede contener operaciones pertenecientes al servicio al que está asociado.
2. Todos los atributos presentes en el mensaje se corresponden a parámetros de la operación que es invocada por medio del mensaje.
3. Se considera al mensaje de invocación de una operación y su correspondiente respuesta como una única entidad mensaje.

A continuación se presenta un ejemplo para facilitar la comprensión del análisis conceptual presentado. Supongamos que un usuario indica que un servicio *ServicioAutorizacion* es de interés. Dicho servicio expone una operación *Autorizar* destinada a generar permisos para realizar determinadas acciones. El usuario define un DataSet correspondiente al servicio que incluye la operación *Autorizar*. Esto significa que al usuario le interesan los valores de los parámetros -de entrada y salida- de las invocaciones de la operación *Autorizar*. Los valores de estos parámetros ingresarán a la plataforma contenidos en los mensajes de invocación a dicha operación. Supongamos ahora que el usuario indica que otro servicio *ServicioAccesoADatos* es de interés. Dicho servicio expone una operación *ObtenerDatos.* Al usuario le interesa analizar si se cumple que para cada acceso del servicio *ServicioAccesoADatos* se invocó previamente al servicio

ServicioAutorizacion. Para lograr esto, el usuario debe definir un DataSet para el servicio *ServicioAccesoADatos* con la operación *ObtenerDatos*. Y posteriormente debe definir una Vista que contenga ambos DataSets generados.

Dada esta problemática, los principales desafíos a resolver son los siguientes:

- Implementar un mecanismo que brinde al usuario la posibilidad de indicar servicios de interés así como definir DataSets y Vistas.
- Implementar mecanismos que permitan crear la estructura del Data Warehouse a partir del ingreso de DataSets y Vistas.
- Utilizar técnicas de CEP para enviar y capturar eventos, que permitan detectar instancias de las Vistas.
- Implementar mecanismos para realizar la carga de los Data Warehouse en tiempo real.

3.2 Relevamiento de requerimientos

Con el fin de entender con mayor exactitud el problema planteado, se definieron un conjunto de requerimientos que deben cumplir los componentes a agregar en el escenario planteado. Estos requerimientos se dividen en funcionales y no funcionales, y los mismos se explicitan en las siguientes secciones.

3.2.1 Requerimientos funcionales

RQ1: Se debe brindar al usuario la posibilidad de agregar nuevos servicios al sistema a través de la selección de su correspondiente archivo WSDL.

RQ2: Se debe brindar al usuario la posibilidad de agregar nuevos DataSets a través de la selección de un servicio y de las respectivas operaciones de interés para dicho servicio.

RQ3: Se debe brindar al usuario la posibilidad de agregar nuevas Vistas a través de la selección de dos o más DataSets. Se debe además indicar las condiciones que relacionan dichos DataSets. Estas condiciones deben soportar los operadores lógicos *AND* y *OR*.

RQ4: Se debe brindar al usuario la posibilidad de visualizar la información sobre sus servicios, DataSets y Vistas agregadas.

RQ5: La solución debe garantizar la creación y carga del Data Warehouse dinámicamente.

3.2.2 Requerimientos no funcionales

Estos requerimientos refieren a pedidos realizados por el cliente.

RQ1: Se deben utilizar técnicas de Complex Event Processing, para resolver las condiciones de relación entre los mensajes correspondientes a las operaciones pertenecientes a los DataSets de una Vista.

RQ2: Los servicios no deben aumentar su tiempo de respuesta debido al procesamiento de los mensajes.

RQ3: Se debe poder agregar nuevos servicios sin tener que dar de baja la aplicación.

RQ4: La solución debe ser implementada en el lenguaje de programación Java.

RQ5: Se debe utilizar la plataforma JBoss para el despliegue de la aplicación.

RQ6: La solución debe ser integrada a un ESB a libre consideración. De todas formas se recomienda utilizar la plataforma Switchyard de JBoss.

RQ7: La solución debe garantizar la posibilidad de generar reportes en alguna herramienta OLAP. Se recomienda considerar la suite de herramientas brindada por Pentaho.

3.3 Trabajo Relacionado

De los problemas encontrados, existe uno que particularmente presenta la mayor dificultad, esto es realizar la carga de un Data Warehouse en tiempo real. Es por ello que se relevaron varios enfoques para resolverlo. No fue posible encontrar ningún enfoque que adaptara algún mecanismo que interactúe con un ESB para realizar el proceso de construcción y carga de un Data Warehouse. No obstante se relevaron dos posibles enfoques para resolver esta problemática prescindiendo de un ESB, ellos son: On the fly y Near Real Time.

3.3.1 Enfoque On the fly

La información sobre este enfoque es realmente escasa, no obstante en [34] se plantea la generación on-the-fly de cubos multidimensionales en un contexto de Web of Things (WoT es una visión donde los dispositivos son integrados a la web reutilizando principios arquitectónicos de la misma, para interactuar con otros dispositivos [35]).

Para esto se define un componente que en la referencia se denomina como Event Data Warehouse Agent (EDWH Agent), que es el encargado de procesar la información, generar los cubos e interactuar con tres bases de datos distintas:

- Una llamada Registered Events Knowledge Base (REKB) en donde se almacena la información de cada nuevo evento que se agregue al sistema.
- Una para el Event Data Warehouse (EDWH).
- Una base auxiliar para almacenar temporalmente los eventos.

Cada vez que se recibe un evento se notifica al EDWH Agent vía JMS (Java Message Service). Una vez notificado, este componente obtiene las dimensiones y medidas de interés para dicho evento de la base REKB. Siguiente a esto se pasa por un proceso de transformación en donde se agregan dichas medidas y dimensiones al propio evento, a las que además se le suma la dimensión Tiempo. Para finalizar se almacena dicho evento transformado en la base de datos auxiliar.

Por otro lado, existe un servicio llamado "Aggregates Manager", que chequea la existencia de eventos en la base de datos auxiliar cada cierto tiempo definido (por ejemplo un segundo para que sea prácticamente tiempo real). En caso que exista alguno, ejecuta funciones de agregación en base a las medidas definidas, genera el cubo y finalmente lo almacena en el EDWH además de remover dicho evento de la base de datos auxiliar.

3.3.2 Enfoque Near Real Time

Este enfoque aplica a estructuras de Data Warehouses fijas, es decir que no cambian dinámicamente en el tiempo, sino que cada vez que se desee realizar cambios en el Data Warehouse se debe rediseñar el mismo.

Se apunta a incrementar considerablemente la periodicidad con que se ejecuta el proceso de carga, y de esta forma lograr aproximarse al tiempo real. La principal ventaja de este enfoque sobre una solución en tiempo real "verdadera", es que no es necesario re implementar la lógica del proceso de extracción, transformación y carga (ETL).

A pesar de que parece una solución sencilla existen varios desafíos a resolver. El mayor desafío es que los procesos de ETL no pueden asumir que los datos se van a mantener estables durante el proceso de carga. Es por esto que surgen inconsistencias entre los sistemas fuente y el Data Warehouse. No obstante en [36] se proponen algunos enfoques para prevenir este tipo de inconvenientes, debatiendo además las ventajas y desventajas de cada uno de ellos.

3.3.3 Resumen

Del análisis realizado para la realidad planteada se detectan para ambos enfoques características que pueden ser utilizadas para el diseño de la solución.

Del enfoque "On the fly" se destaca la generación de los cubos del Data Warehouse dinámicamente y la utilización de una dimensión implícita "Tiempo". No obstante este enfoque se centra en extraer información de dispositivos físicos, y no se adapta enteramente a las características de la realidad que se presenta en el contexto de este proyecto.

Del enfoque "Near Real Time" se destaca la periodicidad con que se ejecuta el proceso de ETL. El mismo podría ser aproximado -en el contexto de este proyecto- mediante el almacenamiento de la información en un contenedor intermedio a modo de garantizar el cumplimiento del requerimiento no funcional RQ2. No obstante se desprende de los requerimientos de este proyecto que la creación del

Data Warehouse debe realizarse dinámicamente, lo cual se contrapone con lo propuesto por este enfoque.

4 Solución propuesta

En este capítulo se presenta la solución propuesta. Se brinda primero una descripción general de la misma, luego se presenta el proceso de construcción del Data Warehouse, y finalmente se muestra el diseño del sistema desarrollado, junto con las decisiones de diseño que se tomaron en el desarrollo de la solución.

4.1 Descripción general

Se propone incorporar algunos componentes al escenario analizado en el Capítulo 3 -ver Figura 20- con el fin de satisfacer los requerimientos que se detallan en dicho capítulo. En la Figura 22 se muestra un diagrama con los componentes agregados al escenario presentado en el Capítulo 3 y cómo interactúan entre sí. Los nuevos componentes son identificados con una trama en la figura.

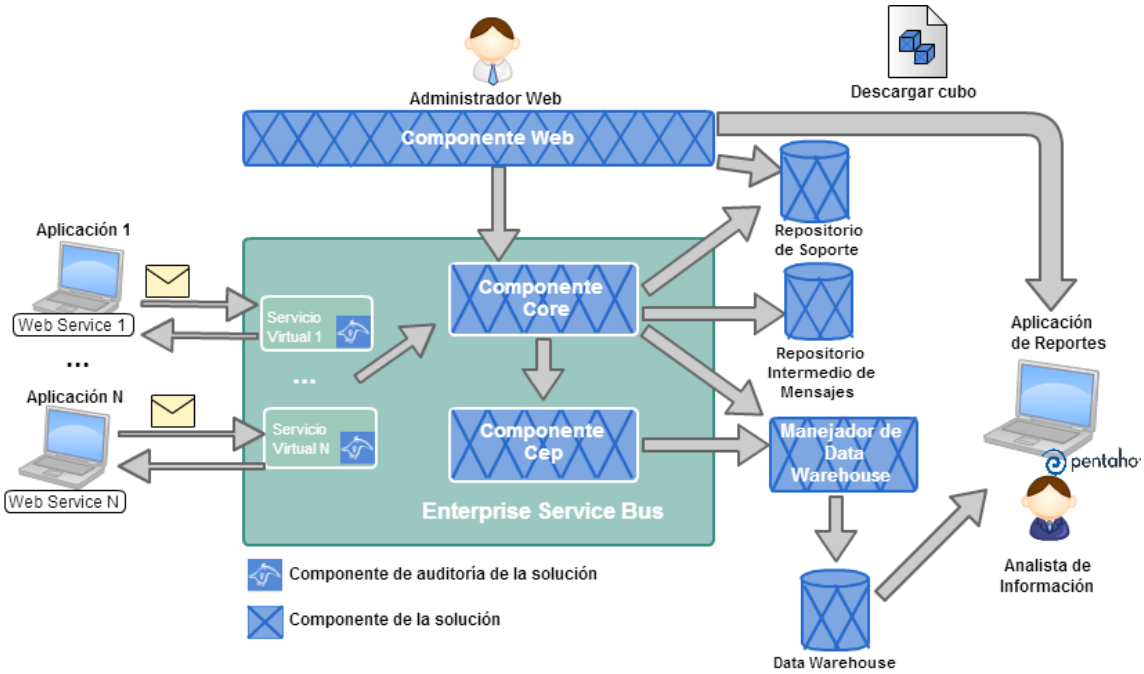


Figura 22: Diagrama de componentes del sistema.

Se plantea una solución que integrada a este contexto es capaz de extraer información de los mensajes que circulan por la plataforma en tiempo mínimo, y la almacena en un Data Warehouse en un tiempo cercano a "Real Time". Dicho Data Warehouse es construido dinámicamente en base a la definición de DataSets y Vistas que realizan los usuarios.

La propuesta se enfoca en capturar todos los mensajes pertenecientes a invocaciones de los servicios virtuales del ESB, para extraer información de interés que los usuarios del sistema indiquen.

Aprovechando las capacidades que poseen los ESB se utiliza el patrón de ruteo de mensajes en base a contenido, para direccionar los mensajes al servicio virtual que corresponda. Los servicios son identificados por su *namespace* en el sistema. Por

otro lado, dado que no se deben perjudicar los tiempos de respuesta de los servicios, se implementa un patrón auditor dentro de cada servicio virtual del ESB que sea capaz de capturar los mensajes y almacenarlos en una base de datos intermedia.

Para capturar los mensajes correspondientes a los DataSets el patrón auditor es suficiente, sin embargo para las Vistas es provechoso utilizar técnicas de CEP.

Además se le brinda al usuario la posibilidad de descargar del componente web, cubos generados a partir de los elementos de interés que seleccionó, para que luego pueda visualizar la información utilizando una aplicación de reportes.

Como se ha desarrollado en el Capítulo 3, el sistema requiere resolver diversas dificultades, por lo que se construyeron varios componentes que se centran en ciertas funcionalidades o se focalizan en brindar una solución a cada característica del sistema.

En un nivel macro los componentes de la solución son:

- **Componente web:** que gestiona el alta de servicios, la creación de DataSets y Vistas en el sistema.
- **Componente de auditoría:** es un componente que realiza la tarea de proveer mecanismos para auditar los mensajes que pasan por el ESB. Los componentes de auditoría se utilizan en los servicios virtuales.
- **Repositorio intermedio de Mensajes:** utilizado como almacenamiento intermedio para guardar los mensajes de interés que son auditados. Se guardan en este repositorio tanto el mensaje recibido junto con su correspondiente respuesta, así como también otros datos de interés que permiten determinar la prioridad del mensaje en cuanto al procesamiento, y otro tipo de información administrativa.
- **Repositorio de Soporte:** para almacenar la información gestionada por el sistema.
- **Manejador de Data Warehouse:** es un componente que realiza la interacción con el Data Warehouse.
- **Componente CEP:** es un componente dedicado a trabajar con eventos complejos. Es el encargado de contener las condiciones definidas en las Vistas y además es quien detecta cuando se produce un *match* de estas condiciones.
- **Componente Core:** es un componente que puede verse como núcleo del sistema. Contiene un manejador de carga, el cual es un planificador que se activa cuando se recibe un nuevo mensaje, y se detiene cuando no quedan mensajes por procesar.

En las siguientes secciones se brinda más detalle de cada uno de ellos.

A continuación se procede a explicar la interacción entre componentes a través del flujo de los mensajes que circulan por la plataforma. Luego se procederá a explicar

los pasos necesarios para realizar la construcción del Data Warehouse. Finalmente se presenta el diseño de los componentes a bajo nivel.

4.2 Flujo de mensajes

Los mensajes enviados al sistema tienen un flujo determinado, el cual recorre los distintos componentes del mismo, que realizan diversas tareas sobre estos mensajes para lograr el objetivo planteado.

En la Figura 23 se muestra un diagrama de secuencia de los mensajes, describiendo a continuación de la misma cada uno de los pasos.

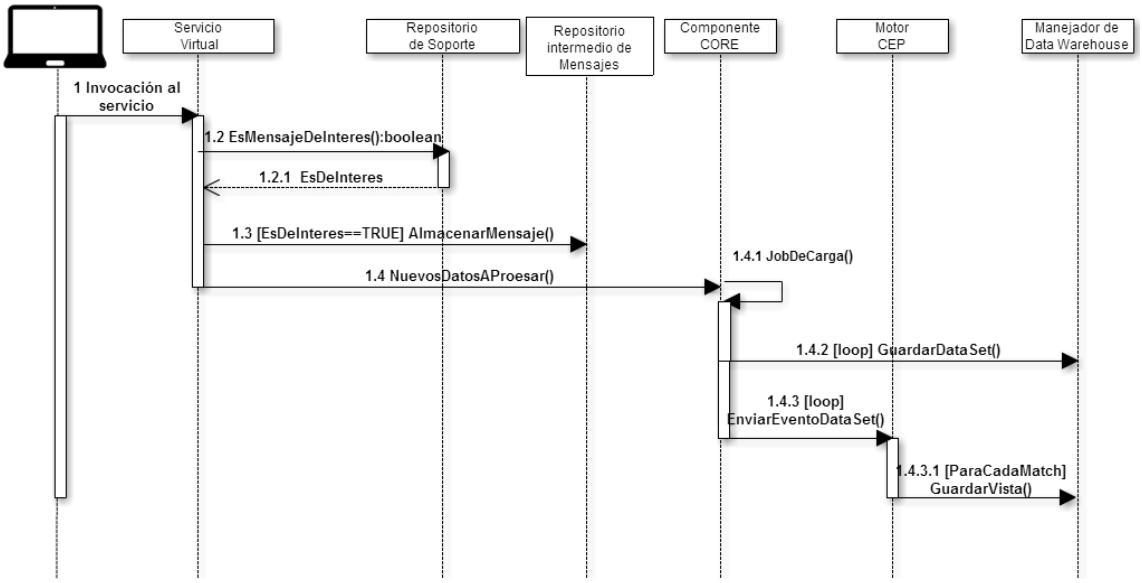


Figura 23: Diagrama de secuencia del ciclo de un mensaje.

1 Invocación al servicio: La secuencia comienza una vez que un servicio virtual es invocado por parte de un servicio externo. Esta invocación se realiza a través de un mensaje SOAP.

1.2 Es mensaje de interés: Una vez que se recibe el mensaje, el componente de auditoría del servicio virtual consulta al repositorio de soporte utilizando el *namespace* del servicio al que el mensaje pertenece junto con un identificador de la operación. Esta acción es realizada para determinar si existe o no un DataSet en el sistema que posea la operación de la que proviene el mensaje.

1.3 Almacenamiento en base intermedia: En caso de que la respuesta sea afirmativa, el componente de auditoría se comunica con el repositorio intermedio de mensajes para almacenar el mensaje recibido junto con su correspondiente respuesta y demás datos mencionados en la descripción de este componente.

1.4 Nuevos datos a procesar: Además el componente de auditoría notifica al componente core sobre la llegada de un nuevo mensaje.
1.4.1 Job de carga: Una vez activado el job de carga, el componente core obtendrá del repositorio intermedio de mensajes el mensaje más viejo sin procesar. A continuación obtendrán los datos de mensaje.
1.4.2 Guardar DataSet: Para cada DataSet relacionado con el mensaje, el componente core se comunicará con el manejador de Data Warehouse para almacenar los datos del DataSet correspondiente.
1.4.3 Enviar evento Cep: Además se envía al componente CEP un nuevo evento para cada Vista que está compuesta por esos DataSet.
1.4.3.1 Guardar Vista: Finalmente para cada evento que satisface un *match* de alguna condición guardada en el componente CEP, se solicita al manejador de Data Warehouse que almacene la información de la Vista correspondiente a dicho *match*.

4.3 Diseño del Data Warehouse

En esta sección se explica el proceso de diseño del Data Warehouse, así como su resultado final.

4.3.1 Análisis conceptual

Se realizó un análisis conceptual de la realidad planteada intentando identificar las etapas a seguir en el proceso de construcción del Data Warehouse. Este análisis tiene como objetivo identificar: *Dimensiones, Cubos y Medidas.*

Escenario

De los requerimientos relevados en el Capítulo 3 se desprende que se desea almacenar información de los mensajes que circulen por la plataforma siempre que sean indicados de interés por un usuario. Esto implica que las fuentes de información se obtienen dinámicamente en diversas etapas a partir de:

1. Los servicios que el usuario indica de interés.
2. Los mensajes que llegan a las operaciones que exponen dichos servicios.

Además la información de las invocaciones a operaciones de interés, se guardarán en el sistema en agrupaciones de datos básicas denominadas DataSets y complejas denominadas Vistas.

Diseño conceptual

Dado este contexto se identifican las Dimensiones: *DataSet* y *Tiempo*. La primera para mantener información sobre los servicios que un usuario desee monitorear, y la segunda para preservar el contexto temporal de la información.

Se identifica también la necesidad de generar un *Cubo* resultado de cruzar cada dimensión DataSet con la dimensión Tiempo. Para satisfacer el requerimiento no

funcional *RQ7* se decidió utilizar como herramienta de visualización de datos el servidor OLAP de la suite Pentaho. Una consecuencia de esta decisión es que para cada cubo se debe generar al menos una medida. Se decide que puede resultar de interés determinar los tiempos de respuesta para los servicios, por lo que este cubo presentará la medida: tiempo de procesamiento de la respuesta por parte del servicio.

En la Figura 24 se muestran las *Dimensiones*, *Cubo* y *Medida* identificados en esta etapa del análisis.

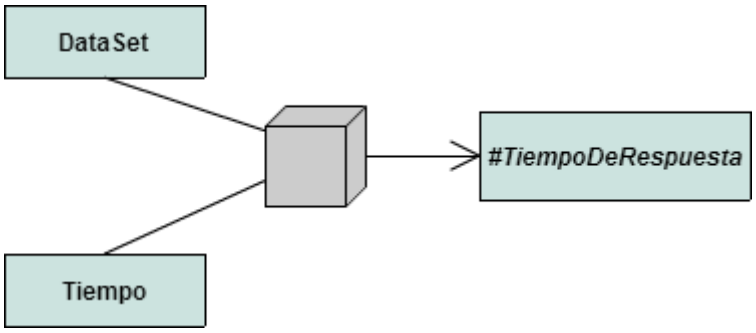


Figura 24: Esquema conceptual de las dimensiones, cubo y medidas identificados.

Por otro lado, para representar las Vistas se define un cubo en el Data Warehouse resultado de cruzar cada una de las dimensiones DataSet que pertenecen a la Vista y la dimensión Tiempo. Para este cubo la medida predefinida será el tiempo transcurrido entre el primer y el último mensaje correspondientes a la Vista en el sistema. Este modelo se presenta gráficamente en la Figura 25.

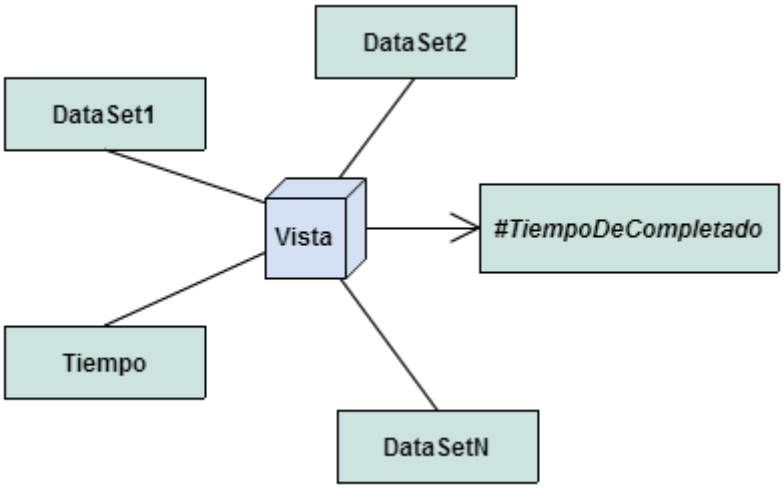


Figura 25: Representación conceptual de una Vista.

Se presenta finalmente un diseño conceptual a través de un ejemplo para un servicio expuesto por la DNIC (Dirección Nacional de Identificación Civil). Dicho servicio brinda información sobre una persona a partir de su documento de identidad.

El proceso de construcción comienza cuando un usuario indica que el servicio es de interés a través del ingreso de su WSDL en el sistema.

Este servicio expone dos operaciones: *ObtPersonaPorDoc* y *ProductDesc*. El usuario debe indicar qué operaciones le son de interés. Para este caso resuelve elegir: *ObtPersonaPorDoc.* Se asume en esta solución que todos los atributos asociados a dichas operaciones son de interés para el usuario.

En la Figura 26 se presenta una imagen de la información indicada por un usuario como de interés a partir del WSDL de un servicio.

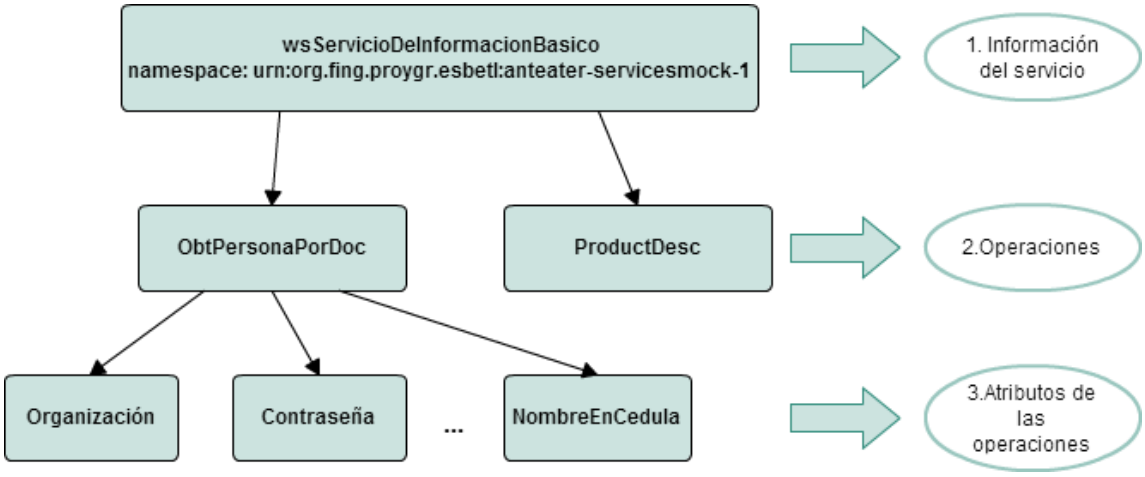


Figura 26: Análisis de un WSDL para la construcción del Data Warehouse.

A partir de esta información el usuario pretende obtener reportes sobre el contenido de los mensajes que circulen por la plataforma, producto de la invocación a la operación *ObtPersonaPorDoc.* Esto puede representarse con los niveles de jerarquías que se muestran en la Figura 27.

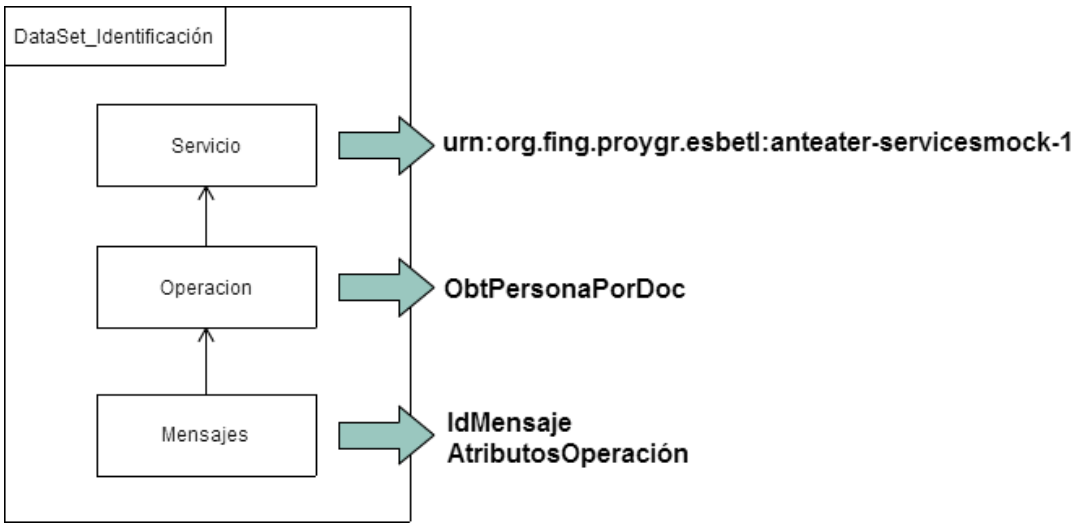


Figura 27: Dimensión generada a partir del DataSet correspondiente al WSDL de la Figura 26.

Dado que el interés del usuario radica en el contenido de los mensajes, se desea inferir medidas -en la futura generación de reportes- a partir de dicha información. Para satisfacer este requerimiento se resuelve que los atributos de los mensajes se almacenen como propiedades del nivel mensaje de dicha jerarquía [37]. En la Figura 28 se ilustra un ejemplo de las propiedades de un mensaje.

Figura 28: Definición de las propiedades del nivel mensaje de la jerarquía a partir de un SOAP request y su correspondiente response

La dimensión Tiempo anteriormente mencionada se presenta en la Figura 29.

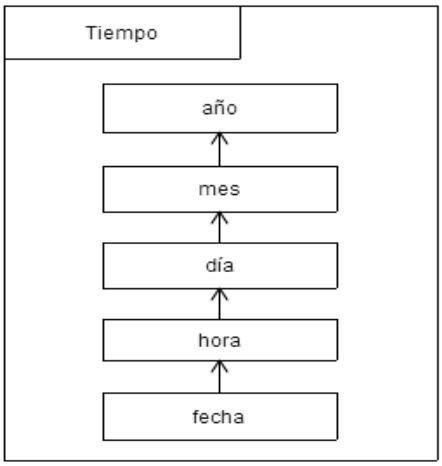


Figura 29: Representación dimensión Tiempo.

Luego para realizar el cruce de la información, la solución deberá generar en el Data Warehouse un cubo cuya representación conceptual se presenta en la Figura 30.

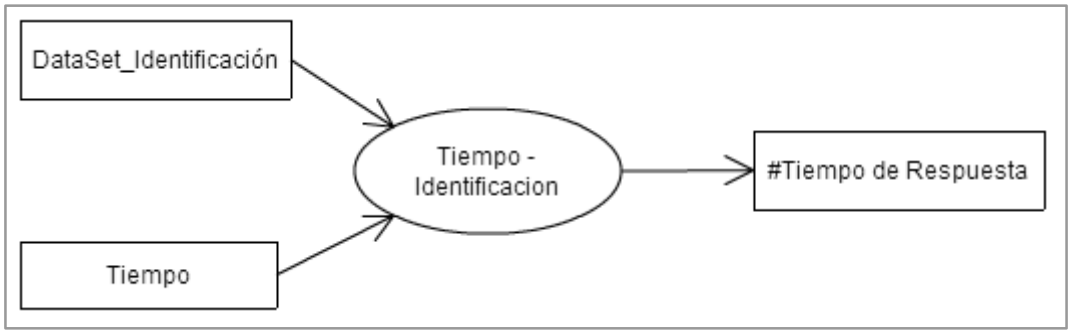


Figura 30: Diseño conceptual del cubo generado por DataSet Identificación y la dimensión Tiempo.

Y cuya representación gráfica se presenta en la Figura 31.

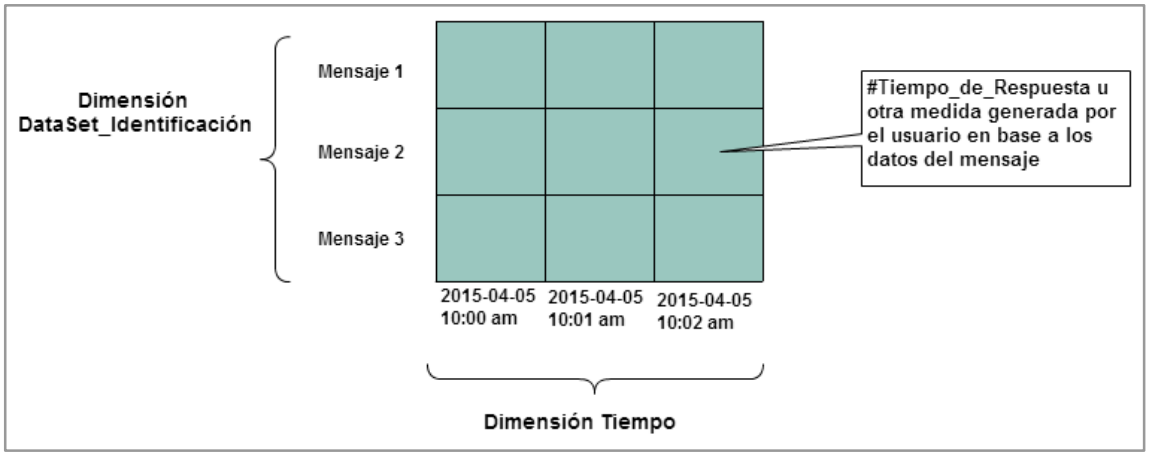
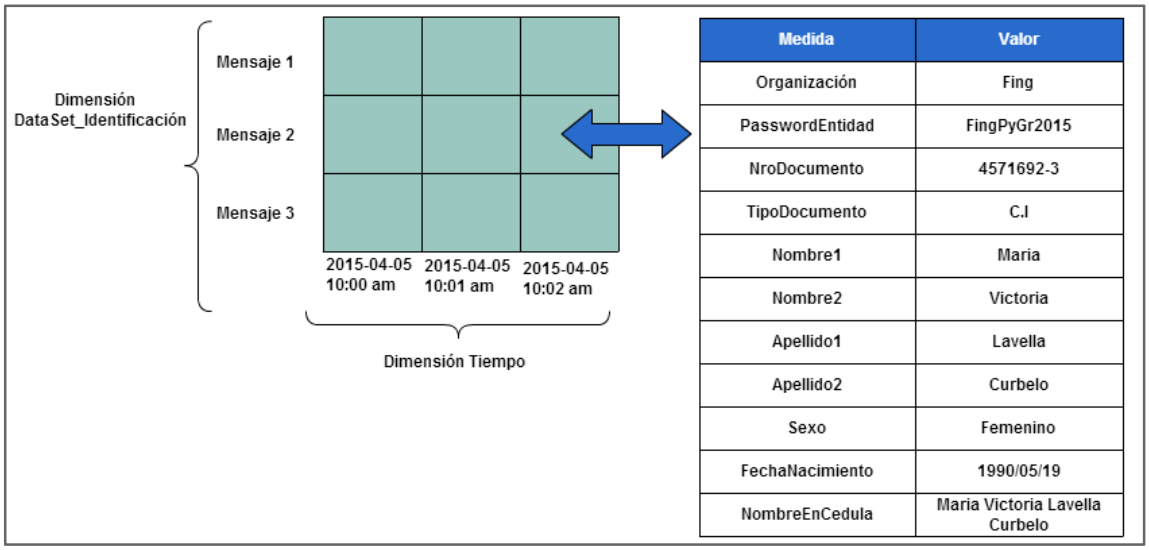


Figura 31: Representación gráfica del cubo generado por el DataSet Identificación y la dimensión Tiempo.

Para el caso en que el usuario indique una medida por parámetro de cada mensaje, el esquema gráfico del cubo se indica en la Figura 32 con los valores definidos en la

Figura 28, donde se pueden ver los valores que tomarían las medidas para cada atributo. Por ejemplo para el caso en que la medida sea el atributo *Organización* su valor será *Fing*.



Medida	Valor
Organización	Fing
PasswordEntidad	FingPyGr2015
NroDocumento	4571692-3
TipoDocumento	C.I
Nombre1	Maria
Nombre2	Victoria
Apellido1	Lavella
Apellido2	Curbelo
Sexo	Femenino
FechaNacimiento	1990/05/19
NombreEnCedula	Maria Victoria Lavella Curbelo

Figura 32: Representación gráfica de los atributos de un mensaje como medidas en el cubo.

Para finalizar el ejemplo se indica que el usuario selecciona otro servicio de interés. Dicho servicio es un servicio que es utilizado por una entidad para exponer métodos relativos a la facturación. El usuario seleccionará para este servicio la operación *agregarFactura*. En la Figura 33 se muestra la representación de la dimensión DataSet generada para esta interacción.

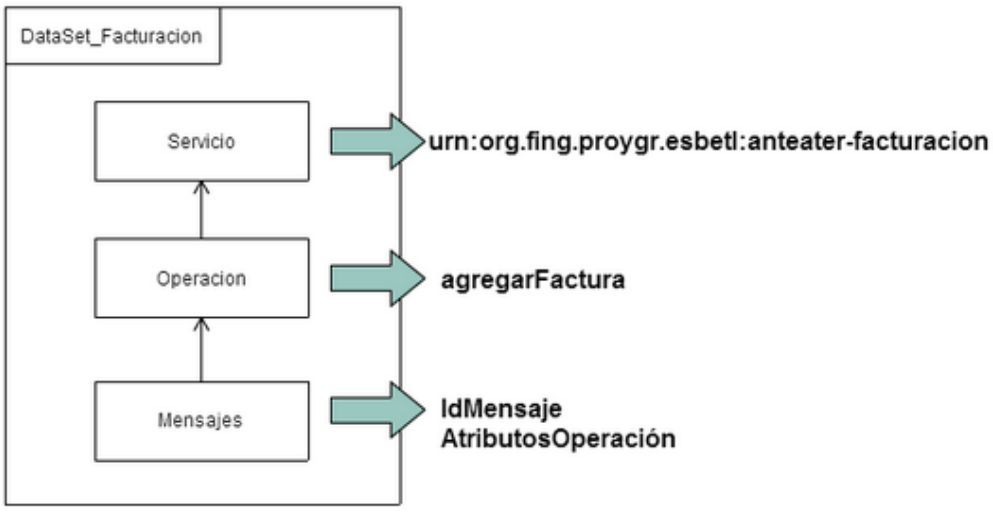


Figura 33: Representación dimensión generada a partir de un servicio de facturación

El usuario además indica que le interesa generar una Vista a partir de la unión de ambos DataSets, es decir el generado a partir del servicio de Identificación y el generado a partir del servicio de Facturación.

Para dicha Vista se presenta en la Figura 34 el diagrama conceptual del cubo generado.

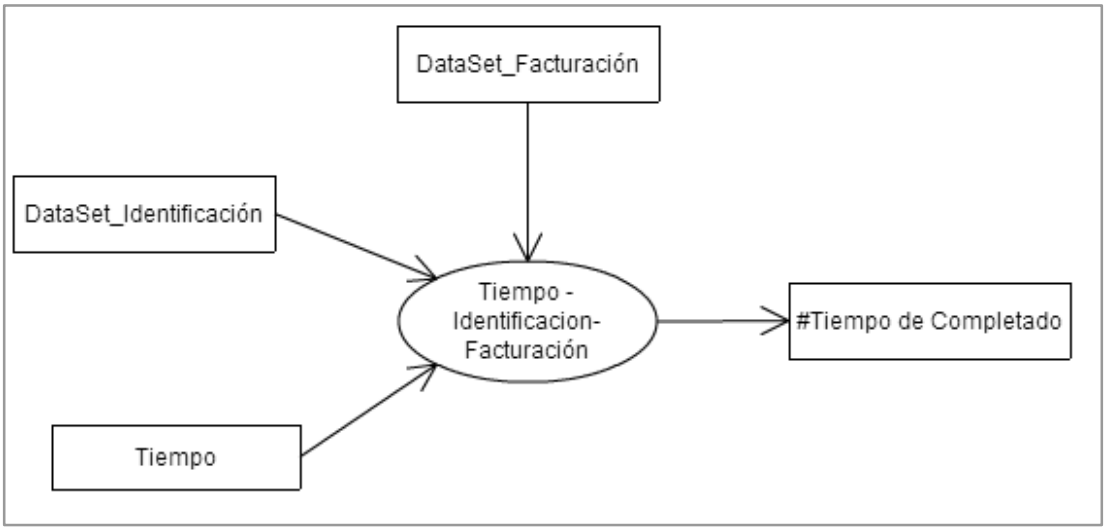


Figura 34: Diseño conceptual del cubo generado para la Vista descrita en el ejemplo.

En la Figura 35 se muestra una representación gráfica de dicho cubo.

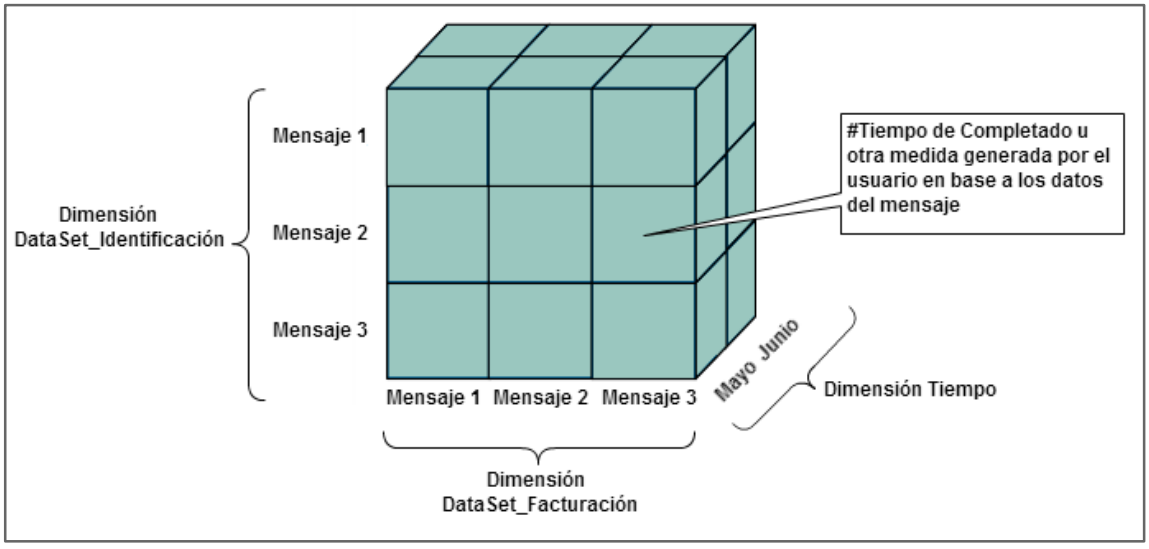


Figura 35: Representación gráfica del cubo generado para la Vista descrita en el ejemplo.

Para el cubo representado en la Figura 35 se tiene que por ejemplo si llega un mensaje a la operación *ObtPersonaPorDoc* en la fecha "*20-04-2015 16:45:00*" y luego se recibe un mensaje para la operación *agregarFactura* en la fecha "*20-04-2015 16:45:01*" el valor de la medida *tiempo_de_completado* será de un segundo.

4.3.2 Diseño lógico

Se presenta en esta sección el diseño lógico realizado para la solución planteada. Aquí se muestra cómo el resultado del análisis conceptual es mapeado a una base de datos física. El diseño de la base de datos fue realizado siguiendo el esquema estrella.

A grandes rasgos se destaca para las tablas generadas por dicho mapeo:

- Cada DataSet se presenta como una tabla de dimensión en la base de datos.
- Todas las tablas de dimensión correspondientes a los DataSets almacenan información sobre el servicio, las operaciones y los parámetros que pertenezcan a dichos DataSets, manteniendo una columna para cada uno de los parámetros.
- Para evitar problemas con los nombres de las columnas de los parámetros, se decide que cada nombre será la concatenación de tipos del parámetro junto con su nombre. Por ejemplo para el atributo *Organizacion* de tipo *ObjParamObtPersonaPorDoc* la columna correspondiente se nombrará *ObjParamObtPersonaPorDoc_Organizacion*.
- Las Vistas se presentan como una tabla de hechos en la base de datos.
- Se genera además una tabla de hechos resultado del cruce de cada dimensión DataSet con la dimensión Tiempo.

Para finalizar se retoma el ejemplo planteado en la sección 4.3.1, y se detalla gráficamente el diseño lógico para el mismo.

En las Figuras 36 y 37 se muestra el diagrama del diseño lógico para el diseño conceptual presentado en las Figuras 34 y 30.

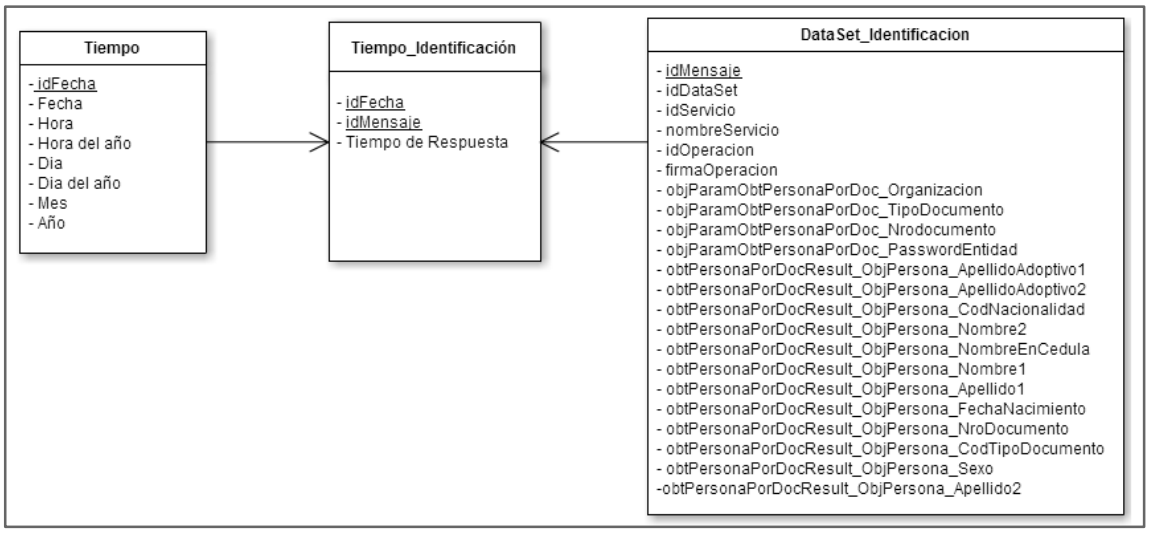


Figura 36: Diseño lógico del Data Warehouse cruce dimensión generada con dimensión precargada Tiempo.

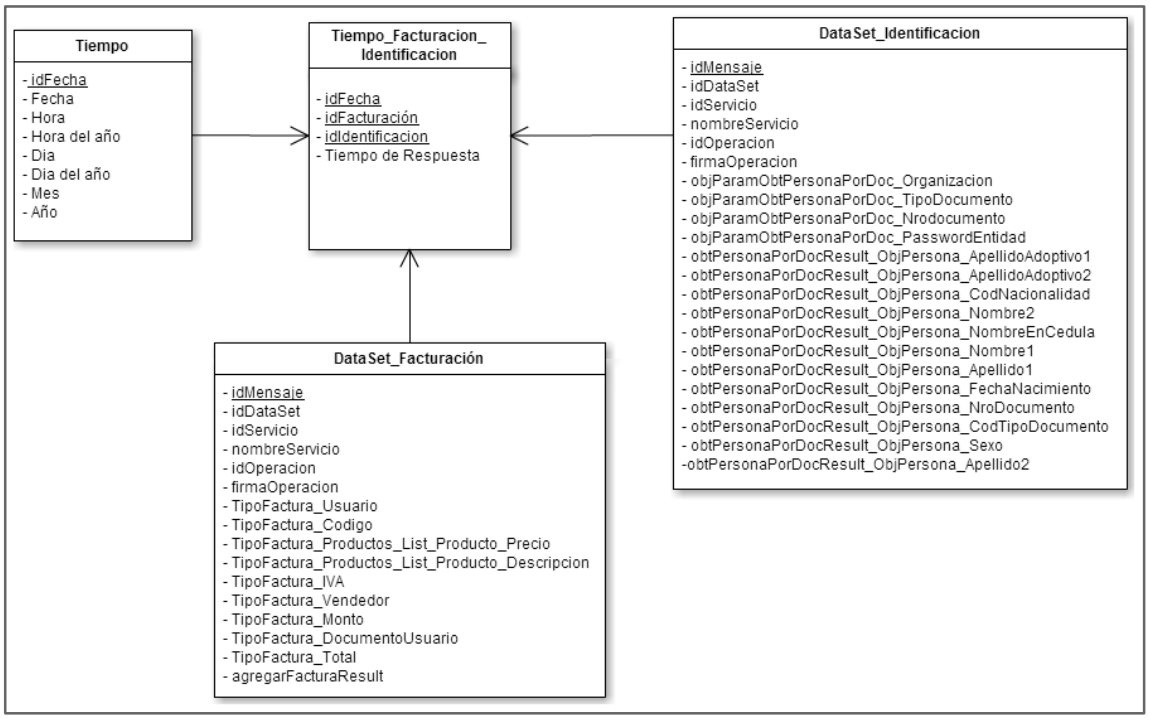


Figura 37: Diseño lógico del Data Warehouse para una Vista.

4.4 Diseño general

A partir del diseño presentado en las secciones 4.1 y 4.2 y del proceso de construcción del Data Warehouse indicado en la sección 4.3 se procede a diseñar la solución a bajo nivel. En la Figura 38 se presenta un diagrama del diseño general del sistema, mostrando los diferentes componentes necesarios para alcanzar los requerimientos presentados en el Capítulo 3.

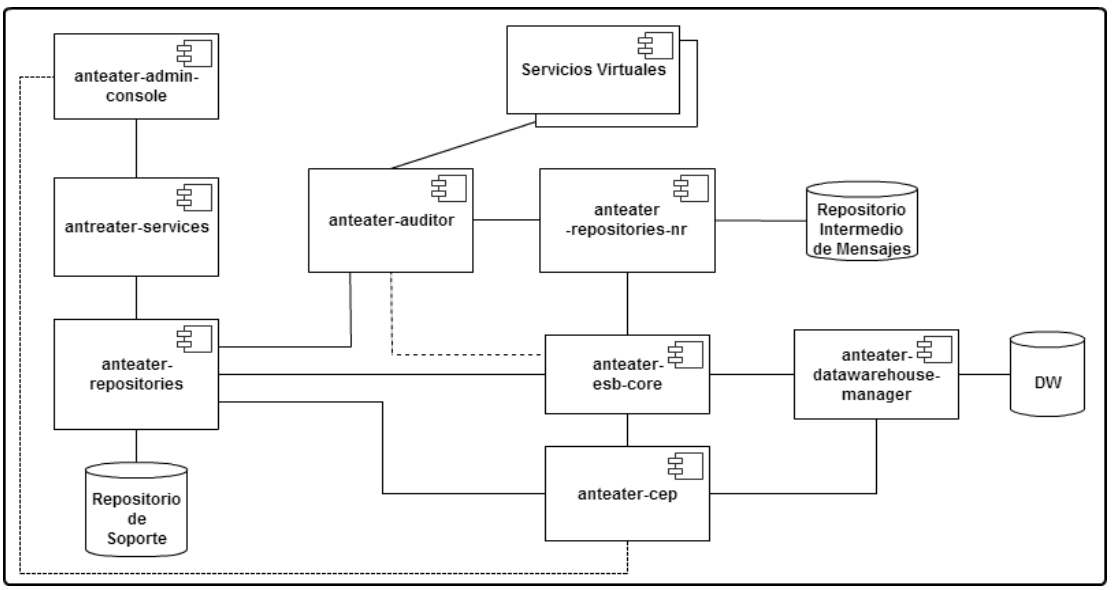


Figura 38: Diseño general del Sistema

Se propone contar con los siguientes módulos para llevar a cabo las funcionalidades a implementar:

- **anteater-admin-console:** este módulo se encarga de la consola de administración web, en donde los usuarios tendrán interacción con la plataforma.
- **anteater-auditor:** módulo que provee la lógica a implementar en los auditores. En este componente se realiza un análisis del mensaje y se decide su siguiente paso en el procesamiento del mismo.
- **anteater-services:** módulo que provee la lógica de servicio para la consola de administración web.
- **anteater-repositories:** este módulo provee los mecanismos necesarios para comunicarse con el repositorio de soporte, en donde se almacena información de las diferentes entidades del sistema.
- **anteater-repositories-nr:** al igual que el módulo anteater-repositories, este módulo brinda los mecanismos necesarios para comunicarse con el repositorio intermedio de mensajes.
- **anteater-datawarehouse-manager:** este módulo permite la comunicación con la base de datos cuya estructura de almacenamiento permitirá volcar la información a un Data Warehouse.
- **anteater-esbcore:** módulo núcleo del sistema, realiza la extracción de la información contenida en los mensajes que circulan por la plataforma y maneja los posteriores eventos que se producen luego de la misma.
- **anteater-cep:** este módulo se encarga del procesamiento de eventos complejos.

4.5 Decisiones de diseño

En esta sección se describen algunas decisiones de diseño que se tomaron para la construcción del sistema.

- **Base intermedia no relacional:** se toma la decisión de utilizar una base de datos intermedia, para almacenar los mensajes que llegan al sistema. Esta decisión busca asegurar no perder mensajes que deban ser procesados, debido a posibles demoras en el procesamiento de los mensajes, o a la eventual baja del sistema. Se busca tener un lugar de almacenamiento físico de los mensajes para evitar que los mismos sean almacenados en memoria. Se selecciona una base no relacional por sus ventajas de rendimiento en grandes volúmenes de datos por sobre una base relacional.
- **Medidas del Data Warehouse:** en la solución presentada se brindan algunas medidas por defecto cuyo valor se relaciona con los tiempos de procesamiento de los mensajes. Sin embargo, no se brinda la posibilidad de definir nuevas medidas sobre la información de los mensajes al momento de construir el Data Warehouse. De todas formas, se brinda un diseño del Data Warehouse que da al usuario la posibilidad de generar sus propias medidas a la hora de visualizar los datos. Esto último se consigue por medio de la inclusión de propiedades en el nivel mensaje de la jerarquía como se definió en la sección 4.3.

- **Auditoría de mensajes:** se decidió utilizar el patrón auditor sobre los mensajes que llegan a los servicios virtuales para no degradar los tiempos de respuesta. Esto es porque antes de almacenar un mensaje en la base intermedia, se analiza el contenido del mismo, para determinar si dicho mensaje pertenece a algún DataSet definido en el sistema.
- **Identificación de los servicios:** con la intención de evitar restricciones más estrictas -como la utilización del estándar *WS-Addressing* -que exige adicionar información específica en los mensajes SOAP [38]-, el sistema propuesto identifica los servicios que se dan de alta por su *namespace*, por lo que se introduce la limitación de que no se pueden repetir los mismos.
- **Diseño del Data Warehouse:** inicialmente se había realizado otro diseño para el Data Warehouse que es especificado en el Apéndice 1. Este diseño debió ser descartado debido a que no permitían dar soporte a la definición de medidas por parte de los usuarios, y no presentaba una forma correcta de visualizar la información de las Vistas. Finalmente se decidió realizar un nuevo diseño que es el presentado en esta solución.

5 Detalles de implementación

En este capítulo se presentan las tecnologías utilizadas, se brindan los detalles de implementación para cada uno de los componentes de la solución, y finalmente se comentan los problemas encontrados.

5.1 Selección de tecnologías

En esta sección se detallan las tecnologías que fueron seleccionadas con el fin de resolver la problemática planteada. Se ha tenido en cuenta que el ambiente de desarrollo debe ser Java/JBoss. Para los componentes del ESB se utilizó Switchyard 1.1. En la Figura 39 se muestra un diagrama de componentes del sistema, indicando para cada módulo las tecnologías que se utilizaron.

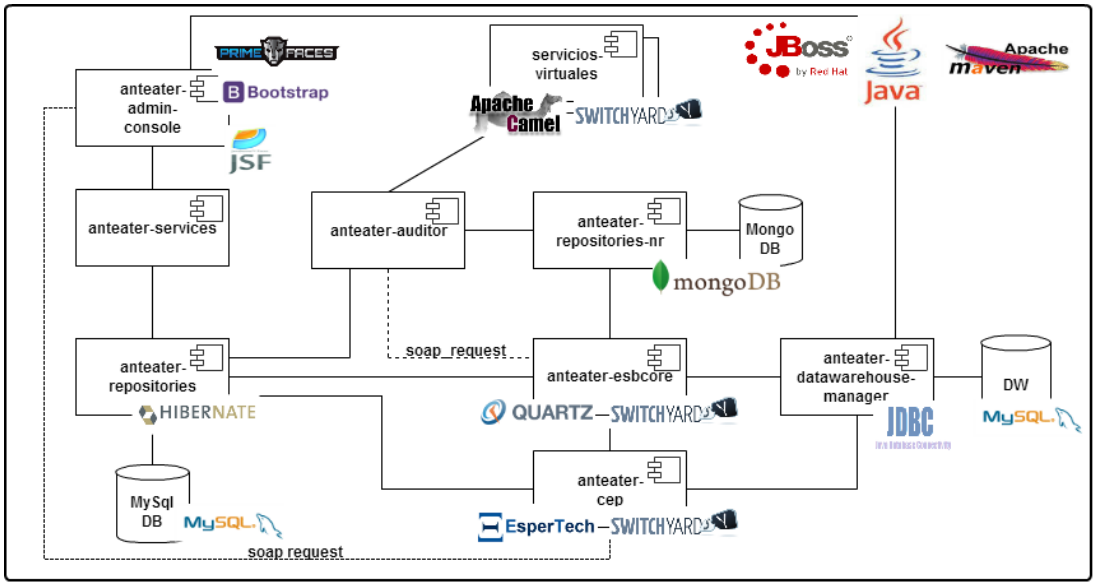


Figura 39: Diagrama de componentes del sistema con tecnologías.

5.1.1 Apache Maven

Es un software de administración de proyectos, que utiliza archivos de configuración POM (Project Object Model) y permite manejar varios aspectos de los mismos. Dentro de los aspectos que maneja se encuentran la compilación del proyecto, documentación, dependencias y librerías. Además permite construir una estructura modular en la cual organizar los proyectos, existiendo normalmente un proyecto padre [39].

Para nuestra solución se utilizó una estructura modular para encapsular todos los proyectos con excepción de la consola web que se construyó como un proyecto maven separado del resto.

Además se utilizó esta tecnología para realizar toda la gestión de dependencias de los diferentes proyectos.

5.1.2 MySQL

MySQL es un sistema de gestión de base de datos relacional, que posee un diseño multi hilo que le permite soportar una gran carga de forma muy eficiente. Este gestor es probablemente el gestor de base de datos más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. Esta gran aceptación es debida, en parte, a que existen infinidad de librerías y otras herramientas que permiten su uso a través de una gran cantidad de lenguajes de programación, además de su fácil instalación y configuración [40].

Se necesitaron utilizar dos bases de datos SQL, una para almacenar los datos ingresados a través de la consola de administración y otra para almacenar las estructuras del Data Warehouse. Se decidió utilizar MySQL porque se contaba con experiencia previa en su utilización. Además las características del motor mencionadas en su descripción pesaron en el motivo de su elección.

5.1.3 Hibernate

Es una herramienta para entornos Java que permite trazar correspondencias entre dos representaciones de datos: un modelo basado en objetos y un modelo de datos basado en SQL. Se ocupa entonces de corresponder clases Java a tablas pertenecientes a una base de datos (y viceversa).

Hibernate también facilita la consulta y recuperación de datos. Esto puede reducir de manera importante el tiempo de desarrollo que se tomaría con el manejo de datos de forma manual en SQL y JDBC [41].

Se evaluó también la posibilidad de utilizar EclipseLink [42], pero se ha decidido utilizar Hibernate para aprovechar el módulo nativo de esta herramienta que provee Jboss EAP (Enterprise Application Platform) [43].

El acceso a datos del módulo *anteater-repositories* fue desarrollado utilizando esta herramienta.

5.1.4 Java Database Connectivity (JDBC)

Es una API de Java para ejecutar sentencias SQL [44]. Se decidió utilizar JDBC para la base de datos que contiene las estructuras del Data Warehouse, debido a que su configuración presenta una menor complejidad, y provee una forma sencilla de ejecutar comandos en forma directa a la base de datos.

Resulta más adecuado utilizar este enfoque y no el que presentaban otros frameworks más complejos como Hibernate o EclipseLink, porque el módulo que interactúa con dicha base implementa funcionalidades orientadas a la creación de tablas e inserción de datos, por lo tanto no necesita de las funcionalidades orientadas a mapeo de objetos que brindan dichos frameworks.

5.1.5 MongoDB

Debido al diseño de la solución que fue planteado, es necesario contar con una base de datos intermedia para almacenar los mensajes que atraviesan el ESB. Dado que los mensajes utilizan un formato XML surgieron dos posibilidades: utilizar una base de datos XML nativa o utilizar una base de datos NoSQL.

Se decidió utilizar MongoDB ya que los productos de las bases XML no brindan la posibilidad de almacenar parámetros extra además del mismo documento. Particularmente es de interés almacenar parámetros para indicar el estado de procesamiento de los mensajes, la fechas de arribo, la fechas de procesado, entre otros.

MongoDB es una base de datos no relacional orientada a documentos. Brinda facilidad a la hora de definir o modificar esquemas a medida que los procesos de desarrollo lo requieran. Además proporciona todas las funcionalidades tradicionales de las bases de datos relacionales como la posibilidad de adicionar índices y definir restricciones de consistencia sobre los datos. Proporciona también un lenguaje de búsqueda que permite manipular la información [45].

5.1.6 Switchyard

Es un framework de desarrollo basado en componentes enfocado a construir aplicaciones o servicios, estructurados y mantenibles utilizando las mejores prácticas de SOA (Service Oriented Architecture). Ofrece varios componentes que simplifican la implementación de los servicios, como por ejemplo Java Bean, Camel, Drools Rules, y Business Process Model and Notation (BPMN).

Por otro lado brinda diferentes *gateways*[1] que permiten a los desarrolladores comunicar el entorno Switchyard con el "mundo exterior", como por ejemplo HTTP, SOAP, SQL, REST, JMS, entre otros.

La lógica de negocio se mantiene aislada de otras funcionalidades como validación, transformación y definición de políticas, ya que estas pueden ser manipuladas de forma declarativa. Esto asegura una mayor consistencia y ofrece a los desarrolladores una visión clara de los servicios pertenecientes a la plataforma de integración [46].

Se decidió utilizar Switchyard (reemplazo de JBoss ESB), ya que fue la recomendada por el cliente, y además tomando en cuenta el análisis comparativo sobre los productos ESBs que utilizan la plataforma Java y tienen licencia Open Source presentado en el Capítulo 3 del Proyecto de Grado "Implementación de una Plataforma ESB Adaptativa" [47]. En el proyecto de grado mencionado se analizan diferentes productos ESB y se muestran las ventajas y desventajas de cada uno, por lo que se comparó Switchyard con los mismos para tener un fundamento más en la decisión de utilizarlo.

1 Aplicaciones que actúan como puente entre varios sistemas.

5.1.7 Apache Camel

Es un framework de código abierto que implementa los principales patrones de integración empresariales (EIP, Enterprise Integration Patterns) descritos en el libro [48]. Se considera que puede ser de mucha utilidad dentro de una arquitectura SOA, es por esto que viene incluido como un componente de Switchyard.

Se decidió utilizar Apache Camel para implementar de forma eficiente el patrón de ruteo en base a contenido [49].

5.1.8 Esper

Es un conjunto de librerías que permite definir, detectar y procesar eventos complejos. Estas librerías proveen de un motor que permite almacenar consultas, que se ejecutan constantemente sobre los eventos que suceden en el sistema. Cuando ocurren las condiciones descritas en alguna de las consultas, el motor responde en tiempo real con las acciones que se especifiquen para el cumplimiento de las condiciones de esas consultas. A diferencia de las bases de datos tradicionales, la ejecución de las consultas es continua.

Esper ofrece un lenguaje (Event Processing Language EPL) para especificar expresiones que detectan patrones basados en eventos, dicho lenguaje tiene mucha similaridad con SQL [50].

Se seleccionó este motor de eventos complejos para ser utilizado en el prototipo por estar modelado de una forma orientada a flujo de eventos, lo cual se ajusta mejor con nuestro sistema. Con orientado a flujo de eventos se quiere decir que los eventos van llegando al sistema y las condiciones de las consultas se basan particularmente en la forma y orden en que arriban estos eventos.

Además se evaluó Drools Fusion como otro motor de eventos complejos. Drools Fusion es un motor modelado orientado a reglas, las cuales marcan condiciones que deben cumplir los eventos y las acciones correspondientes a tomar. Se descartó su uso ya que no se encontró que Drools posea una funcionalidad que permite agregar reglas en tiempo de ejecución. Por lo que al tener que agregar nuevas condiciones para eventos del sistema, se debería volver a *deployar*[2] el módulo de eventos complejos en el servidor de aplicaciones. Esto podría provocar la pérdida de eventos que están ocurriendo en el sistema. Esto último no es admisible ya que si se realiza el *deploy* de dicho módulo se podrían perder mensajes, lo cual implicaría una pérdida de información.

5.1.9 Quartz

Es necesario contar con un planificador de tareas para chequear la presencia de nuevos mensajes a procesar, además de invocar a las operaciones que se encargan del procesamiento de los mismos.

[2] Realizar la instalación del módulo en el servidor.

Se decidió utilizar la librería Quartz como tecnología para implementar dicho planificador, debido a que es uno de los más populares en la actualidad y muchas entidades lo han utilizado embebido en sus aplicaciones, entre las cuales se incluye JBoss [51].

5.1.10 Java Server Faces (JSF)

La tecnología Java Server Faces establece un estándar para el desarrollo de interfaces de usuario del lado del servidor. Incluye un conjunto de APIs que permiten representar los componentes de la UI (user interface) y además una librería de etiquetas para poder expresar las distintas páginas que se construyan.

Propone mantener las capas de interfaz de usuario y la lógica de negocios bien definidas y separadas, y brinda elementos para hacer sencilla la comunicación de las mismas a través de elementos llamados *Beans.*

También permite otras funcionalidades como manejar la navegación entre páginas, el estado de los componentes de la UI, y poder definir eventos sobre la interfaz.

Fue utilizado para el desarrollo del componente web de nuestro sistema *anteater-admin-console*. Se contaba con conocimiento previo de la tecnología y los principios que marca el estándar ayudan a desarrollar un componente sencillo y flexible incluso para poder trabajar en conjunto con otros frameworks de interfaz de usuario.

5.1.11 Primefaces

Es una librería para desarrollo de interfaces de usuario. Se integra bien con JSF por basarse en este estándar. Es muy sencillo de configurar ya que solamente se requiere de incluir esta librería.

Fue utilizado para el desarrollo de algunos componentes de la interfaz web. En particular para el *file uploader*[3] del alta de servicios, para los modales del alta de Vistas, para las validaciones de los formularios y para los gráficos del dashboard.

5.1.12 Bootstrap

Es un framework HTML, CSS y JavaScript para desarrollar sitios web *responsive.* Pone a disposición algunos elementos que ayudan al desarrollo front-end y los sitios se adecuan según el dispositivo donde se esté desplegando. Fue utilizado para generar los estilos del componente de administración web.

5.2 Implementación

En esta sección se presentan los detalles más relevantes de la implementación del prototipo construido. Se presenta por un lado aquellas funcionalidades inherentes a la generación de estructuras, luego se hace mención al procesamiento de mensajes

[3] Componente que permite subir un archivo al sistema.

y carga del Data Warehouse y finalmente se detallan limitaciones y problemas del prototipo.

5.2.1 Estructura del Repositorio de Soporte

Se diseñó la base de datos que se muestra en la Figura 40 para almacenar la información administrativa del sistema. Dicha información refiere a los servicios, Vistas y DataSets que son generados por el administrador de la consola web.

Para manejar el acceso a esta base de datos se construyó el módulo de repositorios *anteater-repositories*. Dicho módulo expone un conjunto de interfaces cuya organización se provee por entidad del sistema.

Las principales responsabilidades de este módulo se detallan a continuación:

- Inserción de información de los servicios en la base de datos
- Inserción de información de los DataSets en la base de datos
- Inserción de información de las Vistas en la base de datos.

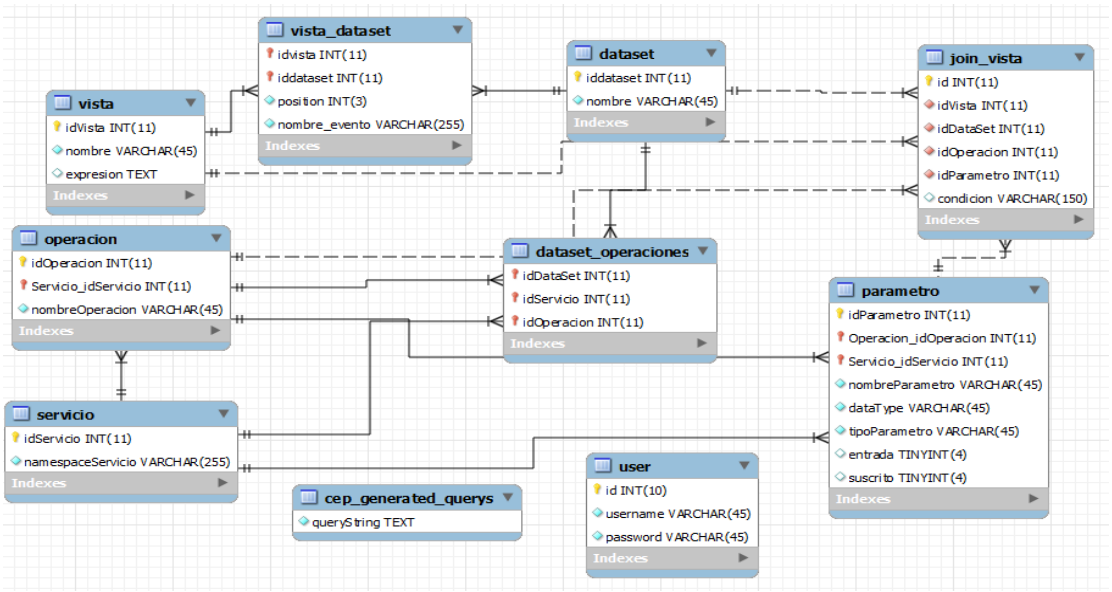


Figura 40: Diagrama de la base de datos *anteaterdb*.

5.2.2 Alta de servicios

El alta de un nuevo servicio se produce a través de la carga del archivo WSDL que describe al mismo en el sistema. Se resuelve realizar el procesamiento del mismo de forma automática, para evitar que el usuario realice el ingreso de los datos del servicio, así como también para evitar posibles errores inducidos por la carga manual.

Cada servicio es identificado en el sistema por su namespace. Por lo tanto, se tiene como precondición que el namespace del servicio debe ser único en el sistema.

Una vez cargado el archivo se procede a procesar la información del mismo, mediante un *parser* para extraer su contenido utilizando para ello la librería *javax.wsdl*. El orden en que se procesa el archivo es el que se indica en la Figura 41.

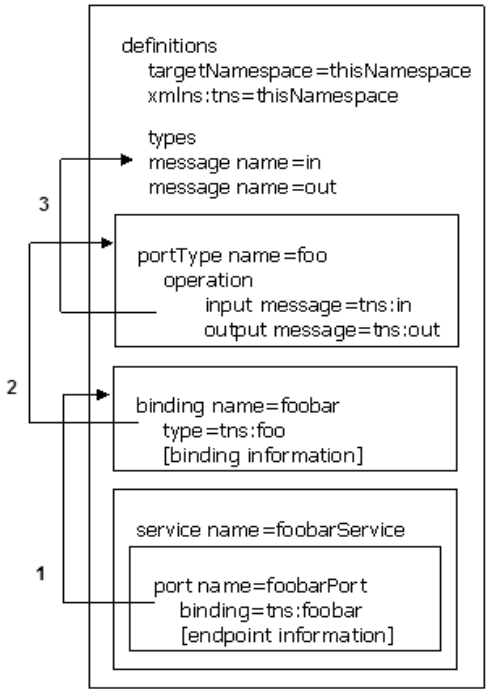


Figura 41: Ejemplo lectura WSDL.

Luego de obtenida toda la información inherente a las operaciones que el servicio expone, así como sus parámetros, la consola de administración *anteater-admin-console,* se comunica con el módulo de servicios para que se almacene la información en la base de datos.

5.2.3 Generación automática de artefactos

5.2.3.1 DataSets

Como se menciona en el Capítulo 3 un DataSet se genera a partir de un servicio, y está formado por una agrupación de operaciones del mismo. Es por lo tanto una precondición de esta funcionalidad que exista al menos un servicio dado de alta en el sistema.

Lo interesante detrás de esta funcionalidad es la creación de las estructuras necesarias para dar soporte a los mensajes que interesen almacenarse para dichos DataSets. Para ello la consola de administración debe encargarse de dos tareas relacionadas entre sí: almacenar la información administrativa del DataSet y crear las tablas que se desprendan del mismo en la base de datos del Data Warehouse.

El proceso de comunicación que se sigue para dicho fin se detalla en la Figura 42.

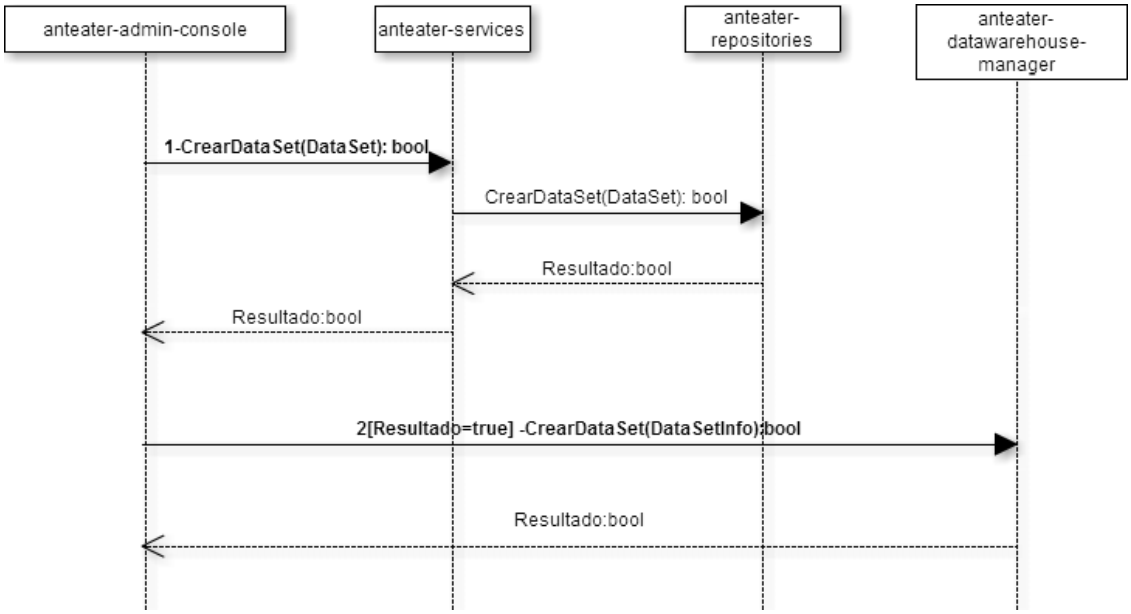


Figura 42: Proceso de comunicación de la creación de DataSets.

Como resultado de este proceso se almacena la información del DataSet en el repositorio de soporte. Además se almacena la información necesaria para relacionar el DataSet con las operaciones que le corresponden. Esta información se guarda en la tabla de nombre *DataSet_Operaciones*.

En el Data Warehouse por otra parte se generan dos nuevas tablas, una de ellas correspondiente a la dimensión que se acaba de crear y otra correspondiente al cruce de dicha dimensión con el tiempo. Ambas tablas se muestra en la Figuras 43 y 44 para el servicio de Identificación descrito en el Capítulo 4.

id	idDataSet	idServicio	nombreServicio	idOperacion	firmaOperacion	objParamObtPers	objParamObtPersonaPorI	objParamObtPersonaPorDc
1	3	6	urn:org.fing.proy...	10	ObtPersonaPo...	fing	CI	4571692-3
3	3	6	urn:org.fing.proy...	10	ObtPersonaPo...	fing	CI	4571692-3
5	3	6	urn:org.fing.proy...	10	ObtPersonaPo...	fing	CI	4571692-3
7	3	6	urn:org.fing.proy...	10	ObtPersonaPo...	fing	CI	4571692-3
9	3	6	urn:org.fing.proy...	10	ObtPersonaPo...	fing	CI	4571692-3

Figura 43: Tabla correspondiente al DataSet identificación. Generada dinámicamente por el módulo Data Warehouse manager

idtiempo	idIdentificacion	tiempo de respuesta
20150404...	1	63
20150404...	3	326
20150404...	5	553
20150404...	7	47
20150404...	9	47

Figura 44: Tabla de hechos correspondiente al cruce entre el DataSet Identificación y la dimensión tiempo. Generada dinámicamente por el módulo Data Warehouse manager.

Para cada tabla generada correspondiente a un DataSet se pone énfasis además en generar tipos de datos que se aproximen al tipo de dato que efectivamente se

corresponde con el mismo, intentando detectar los posibles tamaños de los mismos de modo de no desaprovechar espacio en la base de datos. Para ello la consola de administración provee operaciones dedicadas a generar dicho mapeo. Se indica como tipo de datos por defecto el *VARCHAR (255*), pero dependiendo de la información almacenada para cada parámetro en la creación del servicio en el sistema, el mismo se puede cambiar a *INT*, *DECIMAL*, *BINARY*, *CHAR*, *TEXT*, *BLOB*, entre otros.

5.2.3.2 Vistas

El proceso de generar una nueva Vista en el sistema, es bastante similar al del generar un DataSet descrito en la sección 5.2.3.2. No obstante en este proceso se agrega un nuevo componente a la interacción que se debe realizar desde la consola de administración: el manejador de eventos complejos.

Como se menciona en el Capítulo 3 una Vista se genera a partir de un conjunto de DataSets y una -o muchas-condición de *match* entre los mismos. Es por ende una precondición de esta funcionalidad que existan al menos dos DataSets dados de alta en el sistema.

Lo interesante detrás de esta funcionalidad es la creación de las estructuras necesarias para dar soporte a los *matchs* de las Vistas en el Data Warehouse. Para ello la consola de administración debe encargarse de tres tareas relacionadas entre sí: almacenar la información administrativa de la Vista en el repositorio, crear las tablas que se desprendan de la misma en la base de datos del Data Warehouse y enviar al manejador de eventos complejos la nueva *query*[4] generada a partir de la creación de la Vista. El proceso de comunicación que se sigue para dicho fin se detalla en la Figura 45.

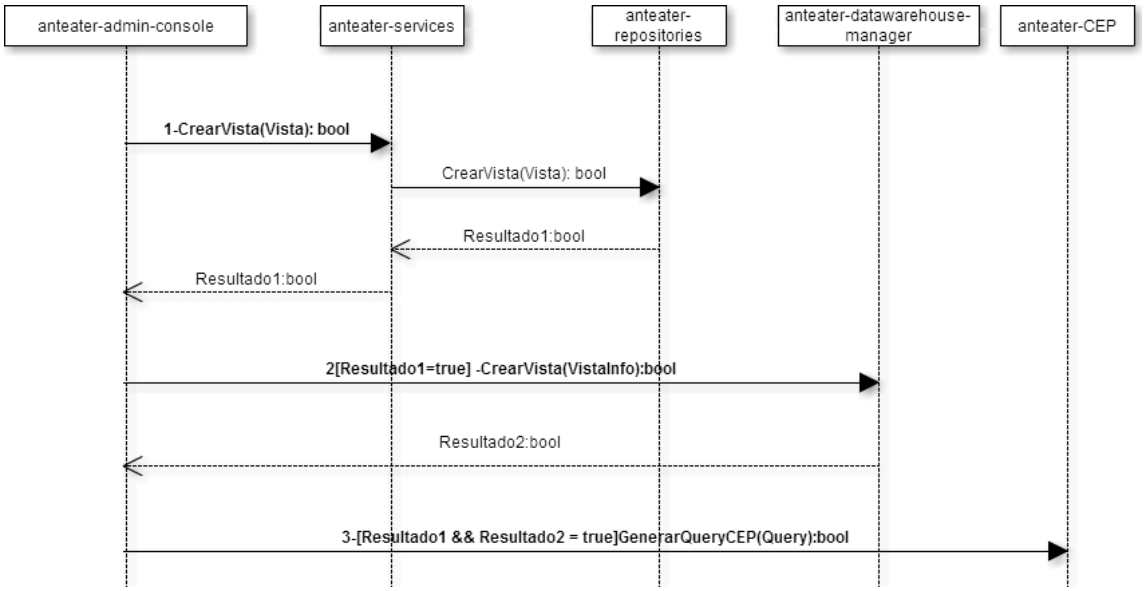


Figura 45: Creación de Vistas

4 Consulta en el motor de eventos complejos

Como resultado de este proceso, además de la información de la Vista, se almacena en el repositorio de soporte la información que relaciona a la Vista con sus DataSets correspondientes en una tabla de nombre *Vista_DataSets*.

En el Data Warehouse por otra parte se genera una nueva tabla que oficiará de tabla de hechos para las dimensiones involucradas en el cubo que define la Vista, es decir los DataSets y la dimensión precargada Tiempo. En la Figura 46 se muestra dicha tabla.

idTiempo	Facturacion	Identificacion	tiempo de completado
201504042...	3	1	711392
NULL	NULL	NULL	NULL

Figura 46: Tabla generada a partir de la creación de una Vista en el Data Warehouse.

5.2.3.3 Condiciones de match para una Vista

Se define una sintaxis para generar las condiciones de una Vista. La misma sigue las siguientes reglas:

1. Cada atributo debe ser descrito de la siguiente manera:
   NombreDataSet.NombreOperacion.TipoCompuestoAtributo_NombreAtributo
2. El operador igualdad debe ser *equal* en minúscula.
3. El operador de unión debe ser *or* en minúscula.
4. El operador de intersección debe ser *and* en minúscula.
5. Solo se permite un operador lógico por expresión.

En la Figura 47 se presenta un ejemplo de la sintaxis para una expresión.

Condición de join:

```
Login.login.username equal Facturacion.agregarFactura.TipoFactura_Vendedor and Facturacion.agregarFactura.Tipo
Factura_Usuario equal Identificacion.ObtPersonaPorDoc.obtPersonaPorDocResult_ObjPersona_NombreEnCedula
```

Figura 47: Ejemplo de la sintaxis para una expresión de *and* y una Vista compuesta por tres DataSets.

Luego en el proceso de creación de una Vista se traduce esa condición a una consulta de Esper que se muestra en la Figura 48 y se almacena dicha consulta en el repositorio de soporte.

Query Esper:

```
select * from
pattern[ every( eventLoginFacturacionIdentificacion_Login=Evento(nombreEvento='LoginFacturacionIdentificacion_Login') ->
eventLoginFacturacionIdentificacion_Facturacion=Evento(nombreEvento='LoginFacturacionIdentificacion_Facturacion',
(eventLoginFacturacionIdentificacion_Facturacion.valores('TipoFactura_Vendedor')?
=eventLoginFacturacionIdentificacion_Login.valores('username')?)) ->
eventLoginFacturacionIdentificacion_Identificacion=Evento(nombreEvento='LoginFacturacionIdentificacion_Identificacion',
(eventLoginFacturacionIdentificacion_Identificacion.valores('objParamObtPersonaPorDoc_Nrodocumento')?
=eventLoginFacturacionIdentificacion_Facturacion.valores('TipoFactura_DocumentoUsuario')?)) )]
```

Figura 48: Ejemplo de traducción de condición de join de Vista a consulta de Esper

5.2.3.4 Descarga de cubos de Mondrian

Se brinda al usuario la posibilidad de descargar los cubos para los diferentes DataSets o Vistas que se hayan generado. Dichos cubos consisten en documentos XML que se construyen siguiendo la especificación de la documentación de la herramienta Mondrian de la suite Pentaho [52].

Para la generación del archivo se utilizaron las librerías *javax.xml* y *org.w3c*. Una vez descargado el cubo el mismo puede visualizarse a través de la herramienta Mondrian.

5.2.4 Procesamiento y carga de datos

5.2.4.1 Auditores

Como se especificó en la Capítulo 4 se decidió que la mejor estrategia para interceptar los mensajes que circulaban por la plataforma era utilizando el patrón *auditor.* Esto se debe a que dicha estrategia tiene la ventaja de no alterar los tiempos de procesamiento de los mismos.

Con este fin se decidió utilizar el patrón ya implementado en el componente de Camel que viene por defecto en la versión de Switchyard utilizada. Uno de los principales problemas que se presentan en este enfoque es que dicho patrón debe ser implementado por cada una de las aplicaciones que se integren a la plataforma. Se resolvió entonces implementar una librería denominada *anteater-auditor* para proveerles a las diferentes aplicaciones integradas a la plataforma la lógica necesaria para utilizar la solución planteada.

Anteater-auditor expone las mismas dos operaciones que expone el patrón auditor implementado por Camel

- **void beforeCall(**Processors processor, String correlationId,String router, Message message**)**
- **void afterCall(**Processors processor, String correlationId,String router, Message message**)**

No obstante los parámetros que recibe no son los mismos, esto es debido a que no se puede referenciar el *exchange* como parámetro, pues se genera un nuevo *correlationId* por cada vez. Este atributo es lo que permite relacionar un *request* con su respuesta. Por ende, para usar la librería es necesario extraer los parámetros

correlationId, *router* **y** *message* pertenecientes a los exchanges proporcionados por el auditor de Camel.

Las responsabilidades de este componente son las siguientes:

1. **Detectar si un mensaje es de interés**: Para ello se consulta al *anteater-repositories* si existe al menos un DataSet definido para el servicio al que la operación invocada pertenece.
2. **Obtener el mensaje y la respuesta y almacenarlos en la base intermedia:** se definieron dos clases Singleton, una llamada *WorkingMemory* y otra *OperationMemory*. La primera con el fin de almacenar las peticiones y la segunda con el fin de almacenar información administrativa sobre el método invocado. Dicha información es luego extraída al momento de procesar las respuestas y junto con las respuestas constituye la información que se almacena en la base intermedia. Una vez procesado y almacenado el mensaje, se elimina su correspondiente Working/Operation Memory.
3. **Comunicarse con el *anteater-esbcore* para comenzar el procesamiento de dicho mensaje**: se envía un mensaje SOAP a una interfaz expuesta por el *anteater-esbcore* para este cometido.

Finalmente es importante destacar que una precondición del uso de este componente es que el mismo debe ser utilizado una sola vez, ya sea en el método *beforeCall* o en el método *afterCall*, y los auditores que lo importen auditen en los siguientes *providers*:*CONSUMER CALLBACK* y *PROVIDER CALLBACK.* En la Figura 49 se muestra un ejemplo del uso del auditor.

```
@Audit({Processors.CONSUMER_CALLBACK, Processors.PROVIDER_CALLBACK})
@Named("Anteater_Auditor")
public class Anteater_Auditor implements Auditor {

	@Override
	public void afterCall(Processors processor, Exchange exchange) {
		}

	@Override
	public void beforeCall(Processors processor, Exchange exchange) {
		String router = exchange.getFromRouteId();
		Message message = exchange.getIn();
		String correlationId = exchange.getExchangeId();
		auditor.beforeCall(processor, correlationId, router, message);
	}

}
```

Figura 49: Ejemplo de uso del módulo *anteater-auditor*

5.2.4.2 Anteater-esbcore

Anteater-esbcore es un proyecto Switchyard que consiste en el módulo núcleo del sistema, debido a las tareas que lleva a cabo. Este módulo realiza la mayor parte del flujo de procesamiento de los mensajes, por lo que brinda componentes implementados en distintas tecnologías que interactúan entre sí para dar soporte a este proceso. En la Figura 50 se muestra cómo es la representación gráfica de este módulo en Switchyard.

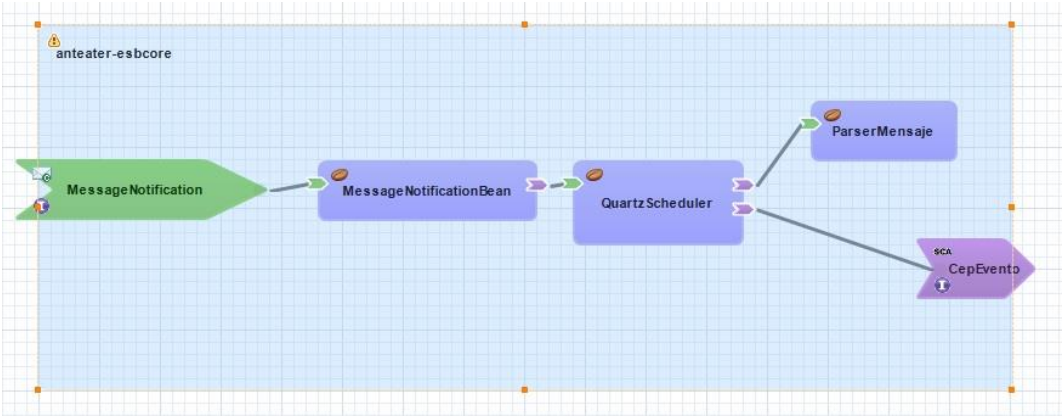


Figura 50: Canvas *anteater-esbcore.*

En la misma se puede ver un canvas que brinda Switchyard, en donde se puede modelar el flujo que van a tener los mensajes al llegar al ESB. Con esta herramienta se ven los módulos del flujo y las interfaces que utilizan para comunicarse.

Anteater-esbcore expone una interfaz *MessageNotification* por la cual los auditores notifican la llegada de un nuevo mensaje al sistema, y además utiliza una interfaz *CepEvento* expuesta por el módulo de eventos complejos para enviar los eventos que se generen en el procesamiento.

También cuenta con otros dos componentes que son el *QuartzScheduler* y el *Parser* de mensajes.

5.2.4.2.1 MessageNotification

Este componente expone una interfaz para ser consumida por los auditores del sistema. La misma cuenta con una operación llamada *MessageReceived* que es utilizada para avisar que se recibió un nuevo mensaje. Dicha operación consulta si el *job* perteneciente al *quartz* está detenido, y en caso de que esté detenido lo inicia. Esta acción desencadena el procesamiento del mensaje.

5.2.4.2.2 ParserMensaje

Otro de los componentes del módulo es el parser de mensajes. Este componente utiliza la tecnología XPath para obtener los valores de los parámetros enviados en el mensaje.

Para esta tarea se expone una interfaz que recibe como parámetros: el mensaje del cual se van a obtener los valores, y una lista con el tipo de parámetro concatenado

con el nombre de cada parámetro. Se define como tipo de parámetro a la concatenación de tipos de un atributo de la operación. Esto se muestra en la Figura 51, donde por ejemplo para el atributo *Codigo* el tipo de parámetro es *datosFactura_Codigo*.

Figura 51: Tipo de mensaje concatenado a nombre de parámetro para un mensaje dado.

La operación devuelve un diccionario conteniendo como clave el tipo de parámetro concatenado al nombre del mismo y el valor del parámetro. Un ejemplo de esto se puede ver en la Figura 52, donde para el caso del atributo *Codigo* se tiene que la clave en el diccionario es *datosFactura_Codigo* y el valor será *QWERTY1234*.

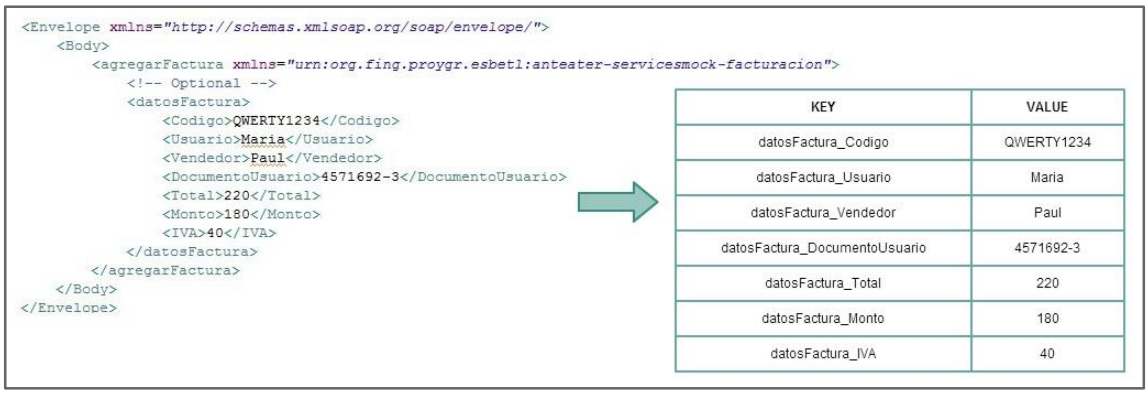


Figura 52: Ejemplo del resultado del parser para un mensaje dado.

Para el caso de las listas se devuelve como valor todos los valores de ese parámetro de la lista concatenados por una coma (“,”). Por ejemplo si tenemos una lista de productos de los cuales cada producto tiene precio y nombre, para el parámetro *Nombre* se devuelve como valor todos los nombres de los productos de la lista concatenados por una coma *nombre1,nombre2,nombre3* y lo mismo para los precios. Un ejemplo de esto se puede ver en la Figura 53.

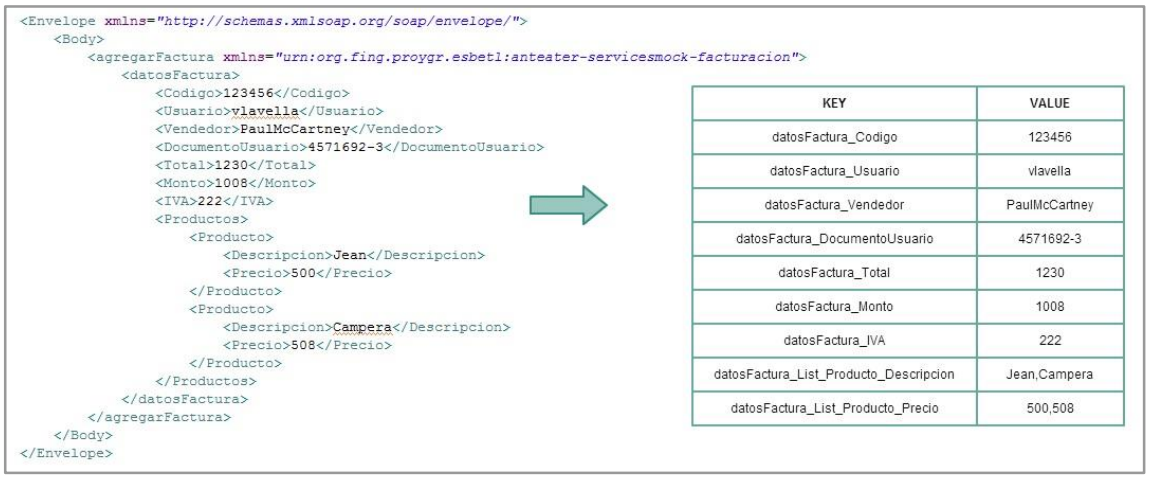


Figura 53: Ejemplo resultado parser de mensajes con listas, para un mensaje dado.

El parser se utiliza tanto para el mensaje de entrada como para el mensaje de respuesta al servicio web.

5.2.4.2.3 Quartz

Este es uno de los componentes más potentes del sistema ya que es quien tiene la responsabilidad de interactuar con los demás componentes para llevar a cabo el procesamiento del mensaje. Se puede dividir en las siguientes partes.

QuartzStartup

Es una clase que posee la anotación de java *@Startup* y además *@Singleton* y realiza un *override* del método *initialize* para que su lógica sea ejecutada en el momento en que el módulo es instalado en el servidor de aplicaciones.

Esta clase inicializa el manejador de MongoDB, utilizado para los accesos al repositorio no relacional, y además dispara el Job del Quartz.

Job

El Job es la tarea que inicia el Quartz para que procese el mensaje que fue notificado.

Como primer paso este Job obtiene el mensaje más antiguo sin procesar. Para ello se comunica con el repositorio mongo, el cual brinda una interfaz para realizar operaciones sobre el mismo. En este caso se utiliza la operación *buscarMensajeParaProcesar*, que devuelve el mensaje más antiguo sin procesar como fue indicado anteriormente.

Si no se obtuvo ningún mensaje entonces se pausa el Job, y se espera hasta que se notifique la llegada de un mensaje. De lo contrario, se procede con los pasos, detallados en la Figura 54.

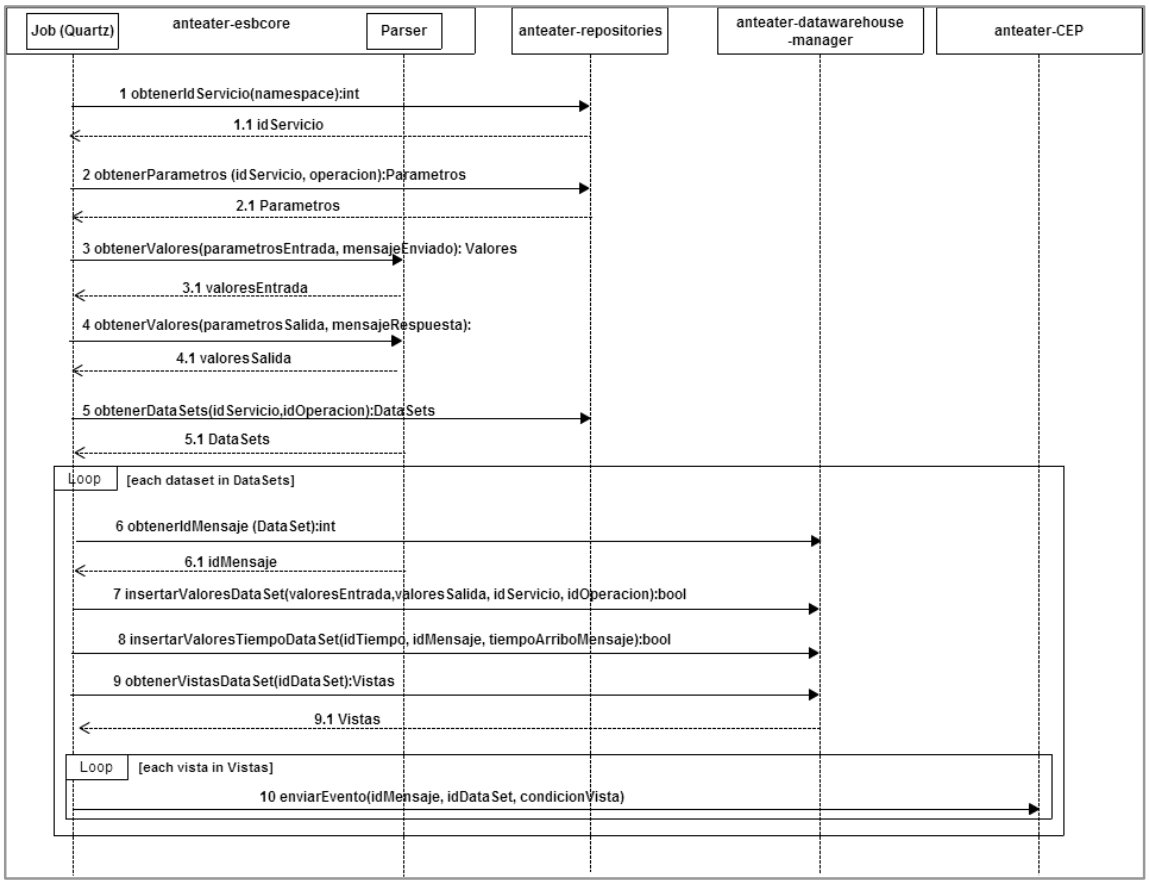


Figura 54: Diagrama de secuencia del Job de carga.

1. Se obtiene el identificador del servicio a partir del namespace del mismo, que está contenido en el mensaje.
2. Con el identificador del servicio y el nombre de la operación, que también está contenida en el mensaje, se obtienen los parámetros de la operación. Esto se hace porque se va a llamar al parser del mensaje quien necesita de ellos.
3. Se invoca el parser en dos oportunidades, la primera con los parámetros de entrada y el mensaje enviado, y la segunda con los parámetros de salida y el mensaje de respuesta del servicio.
4. Se obtienen los DataSets que se crearon en base al servicio y a la operación que obtuvimos del mensaje.
5. Para cada DataSet se generan las entradas correspondientes a ingresar luego en las tablas, utilizando los valores obtenidos del parser, la operación, el servicio, el tiempo de arribo del mensaje, y otros valores correspondientes a las columnas de las tablas de DataSets. En caso de haber un error en este proceso se activa un indicador.
6. En caso de no haber error se insertan en las tablas de DataSets, las entradas creadas en el paso anterior tanto con valores de parámetros de entrada como de salida. También se insertan las entradas correspondientes a las tablas de tiempo de los DataSets.

7. A continuación se obtienen las Vistas a las que pertenece el DataSet que se está procesando en ese momento (recordar que se está recorriendo cada DataSet).
8. Para esa Vista se obtienen las condiciones que se definieron al crearla. Luego se obtienen los parámetros que participan en esas condiciones y se genera una lista conteniendo al parámetro y su valor correspondiente.
9. Finalmente se crea un evento conteniendo esta lista de parámetros y sus valores, el identificador del DataSet, la fecha de arribo del mensaje y el identificador de la entrada en la tabla del DataSet que corresponde. Este evento se envía al motor de eventos complejos donde en caso de "match" de una *query* se insertará una entrada en la tabla de la Vista que corresponda a esa *query.*

5.2.4.2.4 CepEvento

En el Job una de las acciones que se realiza es enviar eventos al motor de eventos complejos, la cual se realiza utilizando esta interfaz expuesta por el motor. La interfaz expone una operación llamada *enviarEvento* que recibe un evento como parámetro. Este evento es construido en base a información manejada en el Job.

El evento cuenta con la fecha de arribo del mensaje, el identificador del DataSet, el nombre del DataSet -esto es a modo de identificar cuál es su tabla correspondiente en el Data Warehouse- y la lista con los parámetros y sus valores.

La comunicación entre los proyectos *anteater-esbcore y anteater-cep* se realiza mediante la tecnología *Service Component Architecture (SCA)*. Esta tecnología está formada por un conjunto de especificaciones que definen la interacción entre dos entidades. Switchyard brinda la posibilidad de utilizar *bindings*[5] *SCA*. Esto se ilustra en la Figura 55.

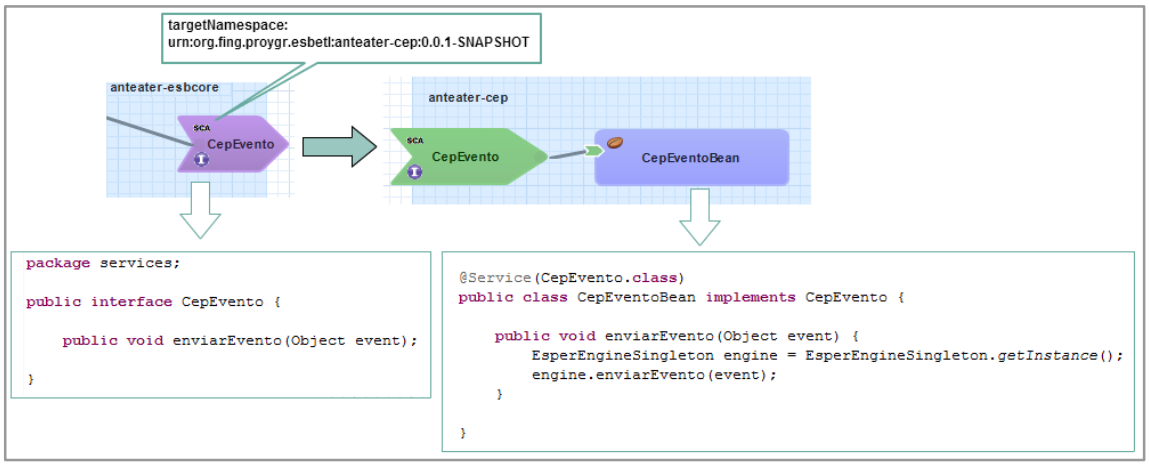


Figura 55: Se muestra la comunicación por SCA entre *anteater-esbcore* y *anteater-cep*.

[5] Un binding es un enlace utilizado para comunicar dos piezas de software.

5.2.4.3 Manejador de eventos complejos

El módulo *anteater-cep* es un proyecto Switchyard cuya principal función es encargarse del manejo de los eventos complejos. Para lograr este cometido expone dos interfaces como puede verse en la Figura 56.

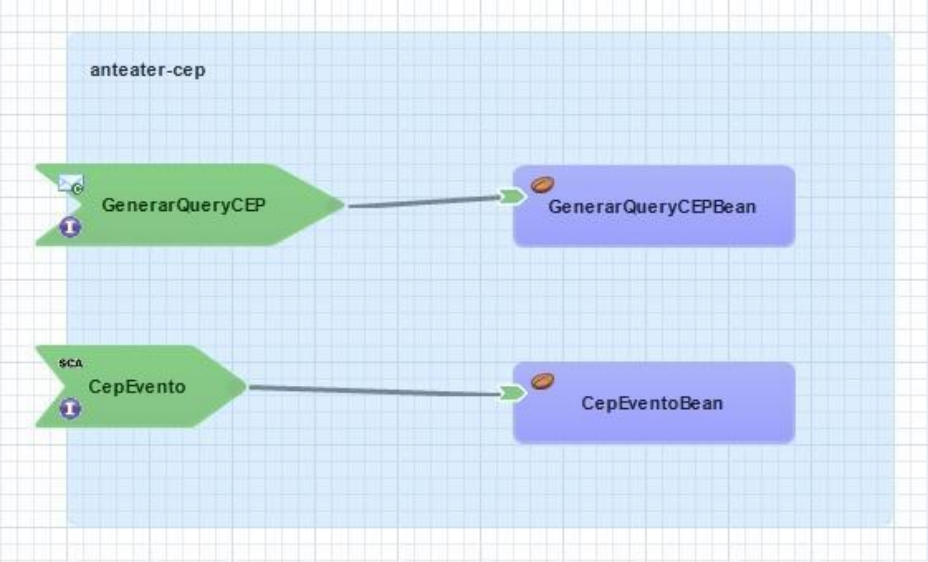


Figura 56: Canvas del módulo *anteater-cep*.

Este módulo implementa un patrón *Singleton* para utilizar una única instancia del motor Esper. Para esto se ha implementado la clase *EsperEngineSingleton*, que además de obtener la instancia incluye otras operaciones que interactúan con el motor.

Por otro lado se cuenta con una clase *CepStartup* que tiene las anotaciones *@Startup* y *@Singleton* que se ejecuta cada vez que el servidor es iniciado y cumple con las siguientes funciones:

- Obtener una instancia del motor Esper.
- Agregar al motor el único tipo de evento válido, que corresponde a la clase Evento perteneciente al módulo auxiliar *anteater-dataobjects*. Este tipo de evento fue definido según los parámetros con que necesitaba contar un evento.
- Verificar si existen consultas que hayan sido almacenadas en el repositorio, y en caso de existir se agregan dichas consultas al motor.

En la Figura 57 se muestra un ejemplo de del uso de dichas anotaciones.

```
@Startup
@Singleton
public class CepStartup {
    @PostConstruct
    public void inicialize() {
        EsperEngineSingleton engine = EsperEngineSingleton.getInstance();
        engine.agregarTipoEventoClase(Evento.class);
        try {
            List<String> generatedQuerys = DataAccessDAO.getAllGeneratedQuerys();
            for(String query : generatedQuerys){
                query=query.replaceAll("\\\\'", "\\'");
                System.out.println(query);
                engine.agregarQueryStartUp(query);
            }
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Figura 57: Ejemplo del uso de las anotaciones @Startup y @Singleton.

A continuación se detallan las funcionalidades de las interfaces indicadas en la Figura 56.

5.2.4.3.1 GenerarQueryCEP

Esta interfaz es utilizada por el módulo *anteater-admin-console* y se encarga de dos tareas:

- Generar la consulta que será agregada al motor, a partir de los identificadores de los DataSets pertenecientes a la Vista que se acaba de definir y opcionalmente a las reglas definidas para dicha Vista. Además la consulta es persistida en una base de datos, lo que asegura que no se pierdan las mismas ante una eventual falla del sistema.
- Agregar la consulta generada al motor y definir a la clase GenericEventSubscriber como listener para dicha consulta. Este listener entonces será notificado cuando alguna de las consultas agregadas al motor realiza un match. Adicionalmente se persisten las consultas generadas en el motor en la base de datos, de forma que ante alguna falla del sistema se puedan volver a agregar nuevamente.

En la Figura 58 se muestra la implementación del suscriptor de eventos.

```
public class GenericEventSubscriber implements UpdateListener {

    public void update(EventBean[] newE
        MapEventBean e = (MapEventBean)
        Map<String,Object> properties =
        SimpleDateFormat formatter = new SimpleDateFormat("EEEE, MMM dd, yyyy HH:mm:ss a");
```

Interfaz de Esper para notificar eventos. Cada "match" que se realice será manejado por esta clase.

Figura 58: Ejemplo de suscriptor de eventos.

Es importante destacar que existe un tiempo de espera entre el envío de la consulta al motor CEP y la activación de la misma. Esto implica que existe un rango mínimo

de tiempo en segundos hasta que los eventos comienzan a ser “detectados” por el motor.

5.2.4.3.2 CepEvento

Esta interfaz es utilizada por el módulo *anteater-esbcore*, lo que realiza simplemente es obtener la instancia del motor y enviar un evento al mismo.

5.3 Limitaciones de la implementación

En esta sección se detallan las limitaciones presentes en el prototipo construido, así como también los casos en donde dichas limitaciones pueden producir algún error.

5.3.1 Parser WSDL

El parser de archivos WSDL consulta los prefijos de los elementos para recorrerlos. En la implementación solo se consideran los prefijos pertenecientes a los siguientes *namespaces*:

- xmlns:xsd="http://www.w3.org/2001/XMLSchema"
- xmlns:s="http://www.w3.org/2001/XMLSchema"

Esto implica que en el sistema solo se soportan servicios que contengan los prefijos ‘xsd’ y ‘s’ en su archivo descriptor.

5.3.2 Mensajes

El parser de mensajes soporta parámetros con los mismos nombres, solo si estos tienen dos niveles de profundidad. Esto significa que para un mensaje en el cual se repite el nombre de un parámetro en tres niveles diferentes, no se garantiza que los parámetros sean detectados correctamente. En las Figuras 59 y 60 se detalla un ejemplo de cada caso.

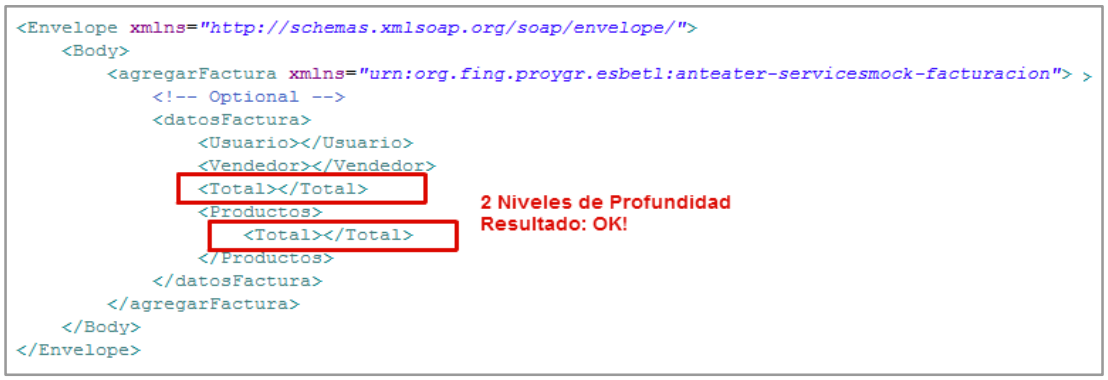


Figura 59: Caso de éxito parser de mensaje

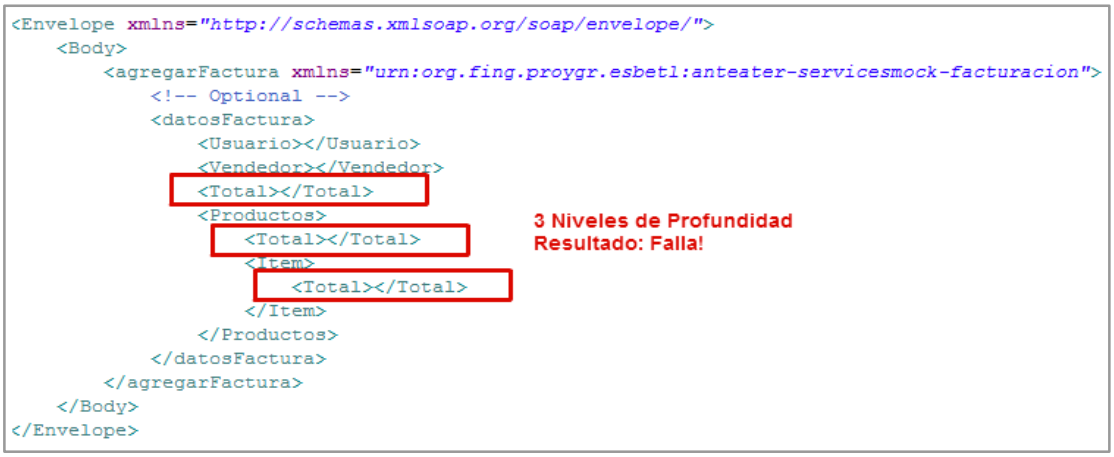


Figura 60: Caso de falla

5.3.3 Condiciones de los eventos complejos admiten un único operador.

Debido al carácter conceptual de la solución presentada, y a los cortos tiempos de implementación, se decidió limitar la posibilidad de complejizar las condiciones de match en la creación de las Vistas. De esta forma no se admite combinar los operadores *AND* y *OR* en la misma expresión. Se puede realizar tantas concatenaciones de los mismos como sea necesario, pero es excluyente que las condiciones sean o bien todas de *OR*, o bien todas de *AND*, o simplemente una igualdad.

5.3.4 Updates

Una clara limitación del prototipo desarrollado, es que no brinda la posibilidad de realizar actualizaciones sobre los elementos creados a través de la consola de administración. Es decir, que cuando se produce un alta de servicio, si el mismo es modificado, será necesario volver a darlo de alta en el sistema, ya que no es posible modificar el que ya se había creado. Por otro lado tampoco se permite eliminar elementos creados.

Sería útil permitir poder realizar actualizaciones y borrados de elementos de la base de datos, pues esto evitaría mantener estructuras que luego no van a ser utilizadas en la misma. Además se brindaría la posibilidad de corregir algún error cometido por el usuario en el momento que se encuentra creando Vistas o DataSets.

5.3.5 Robustez ante fallos de inserción

Una limitación del prototipo implementado es la imposibilidad de recuperarse antes fallos de inserción. Para mejorar esta limitación se propone generar archivos SQL con las sentencias que fallaron de forma de poder ejecutarlas luego.

Se propone concretamente la creación de un indicador de inserción que podría denominarse *OFFLINE_MODE* y el ciclo de vida del mismo es el siguiente:

1. Ante un fallo de inserción en un DataSet se activa el indicador para dicho DataSet y se almacena la información del mensaje en un archivo SQL.

2. Para cada evento enviado al motor de eventos complejos con la información de dicho DataSet se envía adicionalmente el indicador de inserción.
3. Una vez detectado un *match* el manejador de eventos complejos se fija el estado de dicho indicador para cada uno de los eventos participantes del *match*. Si el indicador se encuentra activado para alguno de los eventos, se envía esta información al manejador del Data Warehouse.
4. El manejador del Data Warehouse almacenará la información en la base de datos o en el archivo de inserción según el valor del indicador.

6 Casos de estudio

Con motivo de evaluar el prototipo implementado se definieron tres casos de estudio. Un caso base que se aplica a un único servicio, un segundo en donde se relacionan dos servicios entre sí, y finalmente un tercer caso de estudio que relaciona tres servicios. Para comenzar se detallan los servicios utilizados en los diferentes casos de estudio, luego para cada caso de estudio se presenta la definición del mismo, las pruebas realizadas y los resultados obtenidos. Para finalizar se muestran las conclusiones obtenidas luego de utilizar la plataforma con los tres casos de estudio.

6.1 Servicios utilizados

Se implementaron tres servicios que fueron utilizados en los diferentes casos de estudio.

1. Un servicio de autenticación que tiene una única operación denominada *login.*
2. Un servicio de facturación que cuenta también con una única operación implementada denominada *agregarFactura.*
3. Un servicio auxiliar extraído del catálogo de servicios de la Agesic [53], y constituye una adaptación del servicio básico de información expuesto en dicho catálogo con una única operación denominada *ObtPersonaPorDocumento*. Dicho servicio brinda información sobre una persona a partir de su documento de identidad.

En la Figura 60 se muestra el detalle de la operación *login*.

Operación: Login Parámetro	Descripción	Tipo
Username	Representa el nombre del usuario que quiere ingresar al sistema.	Entrada
Password	Representa la contraseña del usuario que quiere ingresar al sistema.	Entrada
ResultToken	Es el resultado de la invocación realizada.	Salida

Figura 60: Detalle operación *login*.

En la Figura 61 se muestra el detalle de la operación *agregarFactura*.

Operación: AgregarFactura		
Parámetro	**Descripción**	**Tipo**
Código	Representa el código de la factura.	Entrada
Usuario	Representa el nombre del usuario al cual está destinada la factura.	Entrada
DocumentoUsuario	Representa el documento del usuario al cual está destinada la factura	Entrada
Vendedor	Representa el nombre del vendedor que realiza la venta	Entrada
Total	Representa el valor con impuestos incluidos	Entrada
Monto	Representa el valor sin impuestos incluidos.	Entrada
IVA	Representa el valor de dicho impuesto al momento de realizar la factura	Entrada
List<Producto>	Indica para cada producto que integra la factura, una descripción y su precio.	Entrada
Resultado	Indica si la operación fue realizada con éxito.	Salida

Figura 61: Detalle operación *agregarFactura*.

En la Figura 62 se muestra el detalle de la operación *ObtPersonaPorDocumento*.

Operación: obtPersonaPorDocumento		
Parámetro	**Descripción**	**Tipo**
Organizacion	Representa el nombre de la organización.	Entrada
PasswordEntidad	Representa la contraseña de la entidad solicitante.	Entrada
NroDocumento	Representa el representa el número de documento de la persona solicitada.	Entrada
TipoDocumento	Representa el tipo de documento solicitado.	Entrada
NroDocumento	Representa el representa el número de documento de la persona solicitada.	Salida
Nombre1	Representa el primer nombre de la persona solicitada.	Salida
Nombre2	Representa el segundo nombre de la persona solicitada, en caso de que lo tenga.	Salida
Apellido1	Representa el primer apellido de la persona solicitada.	Salida
Apellido2	Representa el segundo nombre de la persona solicitada.	Salida
Sexo	Representa el sexo de la persona solicitada.	Salida
FechaNacimiento	Representa la fecha de nacimiento de la persona solicitada.	Salida
CodNacionalidad	Representa el código que identifica la nacionalidad de la persona solicitada.	Salida
NombreEnCedula	Indica el nombre completo que en el documento de la persona solicitada.	Salida

Figura 62: Detalle de la operación *ObtPersonaPorDocumento*.

6.2 Caso de estudio 1: Monitoreo de un único servicio.

El objetivo del primer caso de estudio es analizar un escenario sencillo en donde se cuenta con un único servicio, se define un DataSet y se desea visualizar los datos obtenidos.

6.2.1 Pruebas realizadas.

Como primer paso se da de alta el servicio de autenticación en la consola de administración. Tomando como referencia la Figura 63, se debe presionar el botón *Choose*, seleccionar el archivo WSDL y finalmente presionar el botón *Upload*.

Figura 63: Formulario de alta de servicio

Luego se crea un DataSet con el nombre *Login* incluyendo la operación *login* expuesta por dicho servicio. Un resumen de esta acción se muestra en la Figura 64.

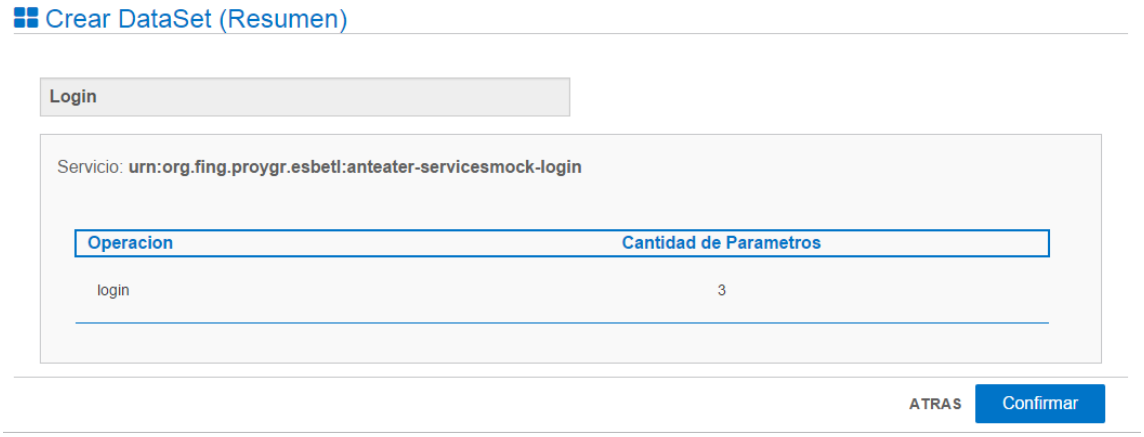


Figura 64: Formulario alta de DataSet Login.

A continuación se envía un mensaje a través de un cliente SOAP invocando la operación incluida en el DataSet generado, como puede verse en la Figura 65.

POST http://localhost:8080/anteater-servicesmock-login/WsAutenticacion

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
        <login xmlns="urn:org.fing.proygr.esbetl:anteater-servicesmock-login">
            <username>marcos</username>
            <password>defensor</password>
        </login>
    </Body>
</Envelope>
```

Figura 65: Mensaje SOAP para el servicio de autenticación.

Adicionalmente se genera un cubo para visualizar los datos en Pentaho como se muestra en la Figura 66.

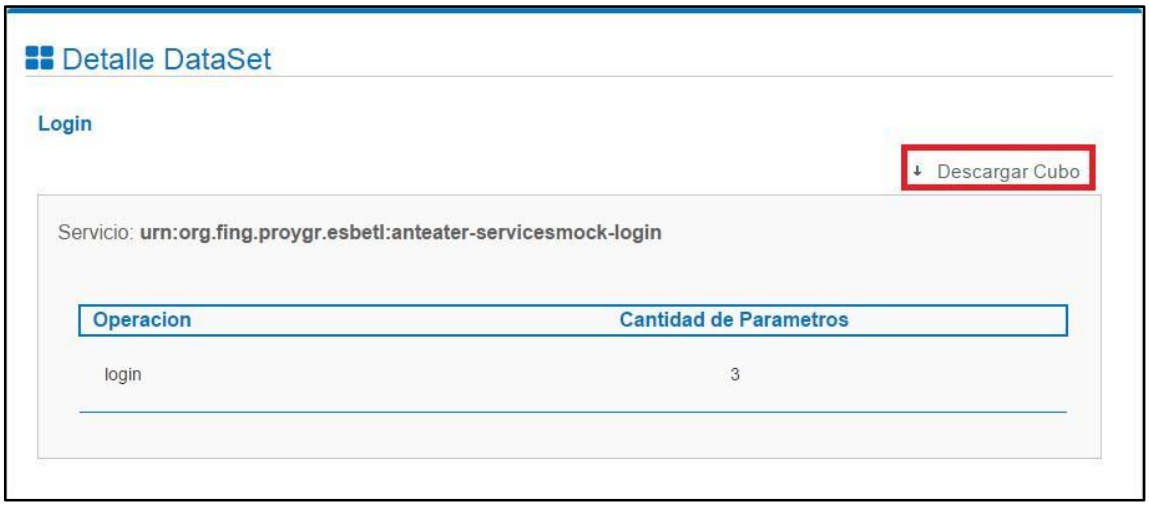


Figura 66: Descarga de cubo para el DataSet *Login*.

6.2.2 Resultados obtenidos.

Se obtienen los resultados esperados, los cuales se listan a continuación:

- Se almacena la información correspondiente al DataSet en el repositorio de soporte.
- Para el Data Warehouse se crea una tabla para la dimensión definida por el DataSet *Login* y una tabla para el resultado del cruce de esta dimensión con la dimensión *Tiempo*.
- Como resultado del mensaje enviado se insertan los datos en las tablas del Data Warehouse. En las Figuras 67 y 68 se muestran las tablas del punto anterior con los datos correspondientes al mensaje enviado.

id	idDataSet	idServicio	nombreServicio	idOperacion	firmaOperacion	username	password	loginResult
1	40	7	urn:org.fing.proygr.esbetl:anteater-s...	11	login	marcos	defensor	tokenLogin
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 67: Tabla dimensión *Login* con los datos del mensaje enviado.

	idtiempo	idLogin	tiempo de respuesta
	201505110315	1	106
▶*	NULL	NULL	NULL

Figura 68: Tabla de hechos *TiempoLogin* con los datos del mensaje enviado.

6.2.3 Visualización de datos.

Es posible visualizar dichos datos en la suite Pentaho, para ello se publica el cubo descargado en el servidor OLAP de Pentaho con el fin de generar dos reportes. Ambos reportes se extraen utilizando la herramienta Saiku del servidor.

El primer reporte tiene como objetivo visualizar el tiempo de respuesta del mensaje enviado. En la Figura 69 se detalla la consulta MDX generada, y en la Figura 70 se detalla el resultado obtenido.

```
WITH SET [~COLUMNS] AS
  Hierarchize (
    {
      { [Login.HLogin].[Operacion].Members },
      { [Login.HLogin].[Mensaje].Members }
    }
  )
SET [~ROWS] AS
  Hierarchize (
    {
      { [Tiempo.HTiempo].[Anio].Members },
      { [Tiempo.HTiempo].[Mes].Members }
    }
  )
SELECT NON EMPTY Crossjoin (
  [~COLUMNS],
  { [Measures].[tiempo de respuesta] }
) ON COLUMNS,
NON EMPTY [~ROWS] ON ROWS
FROM [Tiempo_Login]
```

Figura 69: Consulta MDX reporte 1.

		login
		19
Anio	Mes	tiempo de respuesta
2015	5	297

Figura 70: Resultado reporte 1.

Adicionalmente se genera un segundo reporte para visualizar los datos del mensaje enviado. En la Figura 71 se muestra la consulta MDX utilizada en la herramienta, y en la Figura 72 se detallan los resultados obtenidos.

```
WITH MEMBER [Measures].[username] AS
  [Login.HLogin].[Mensaje].CurrentMember.Properties ( "username" )
MEMBER [Measures].[password] AS
  [Login.HLogin].[Mensaje].CurrentMember.Properties ( "password" )
SET [~COLUMNS] AS
  { [Login.HLogin].[Mensaje].Members }
SET [~ROWS] AS
  (
  { [Tiempo.HTiempo].[Mes].Members } )
SELECT NON EMPTY
{ [Measures].[tiempo de respuesta], [Measures].[username], [Measures].[password] } ON ROWS,
NON EMPTY NonEmptyCrossjoin ( [~ROWS], [~COLUMNS] ) ON COLUMNS
FROM [Tiempo_Login]
```

Figura 71: Consulta MDX reporte 2.

	5
MeasuresLevel	19
tiempo de respuesta	297
username	marcos
password	defensor

Figura 72: Resultado reporte 2.

6.3 Caso de estudio 2: Monitoreo en dos servicios relacionados entre sí.

El objetivo del segundo caso de estudio es agregar complejidad al primero. Se agrega un segundo servicio, se define un nuevo DataSet, se define una vista que relacione ambos DataSets y finalmente se visualizan los datos obtenidos.

6.3.1 Pruebas realizadas.

Como punto de partida se reutiliza el DataSet *Login* definido en el primer caso. Luego se da de alta el servicio de facturación en la consola de administración, y se crea un DataSet de nombre *Facturación* incluyendo la operación *agregarFactura* expuesta por dicho servicio.

Como segundo paso se crea una Vista con nombre *LoginFacturacion* que contiene los DataSets *Login* y *Facturación*, y se define la condición de join de la misma. Dicha condición es que el parámetro *username* del mensaje de la operación *login* coincida con el parámetro *vendedor* indicado en el mensaje de la operación *agregarFactura*. Se debe especificar además el orden en que suceden estos mensajes (esto se realiza a través de números consecutivos, comenzando con 0 para el primer mensaje), en este caso se ha decidido que suceda primero una invocación a la operación de *login.* Un resumen de los datos ingresados en la consola de administración es detallado en la Figura 73.

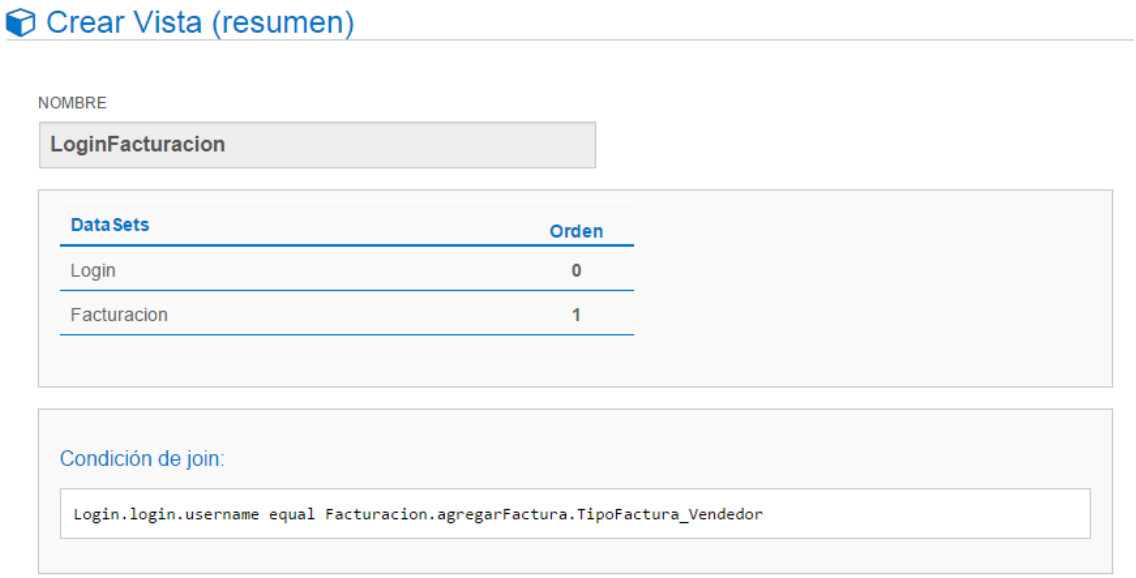


Figura 73: Resumen de la Vista.

En este punto se realizan dos pruebas:

- Se envía primero un mensaje a través de un cliente SOAP invocando la operación incluida en el DataSet *Login*, y otro mensaje invocando a la operación incluida en el DataSet *Facturación*, tal que el valor del parámetro *username* de la operación *login* sea diferente del valor del parámetro *vendedor* de la operación *agregarFactura*.
- Se realiza la misma prueba que la anterior con la diferencia que los parámetros especificados para ambas operaciones sean iguales como se muestra en la Figura 74.

POST http://localhost:8080/anteater-servicesmock-facturacion/WsFacturacion

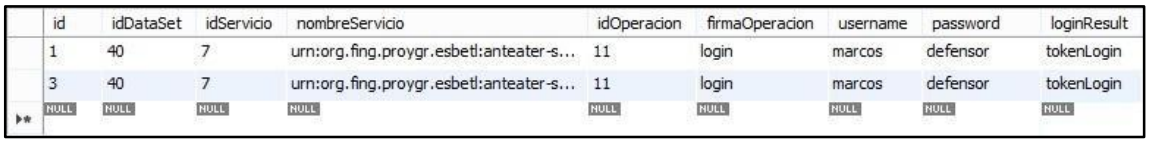
```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
        <agregarFactura xmlns="urn:org.fing.proygr.esbetl:anteater-servicesmock-facturacion">
            <!-- Optional -->
            <datosFactura>
                <Codigo>c1</Codigo>
                <Usuario>u1</Usuario>
                <Vendedor>marcos</Vendedor>
                <DocumentoUsuario>4.247.448-5</DocumentoUsuario>
                <Total>13</Total>
                <Monto>8</Monto>
                <IVA>5</IVA>
                <!-- Optional -->
                <Productos>
                    <!-- Optional -->
                    <Producto>
                        <Descripcion>desc</Descripcion>
                        <Precio>15</Precio>
                    </Producto>
                </Productos>
            </datosFactura>
        </agregarFactura>
    </Body>
</Envelope>
```

Figura 74: Detalle Mensaje SOAP enviado al servicio de facturación.

6.3.2 Resultados obtenidos.

Se obtienen los resultados esperados, los cuales se listan a continuación:

- Se almacena la información correspondiente al DataSet y a la Vista en el repositorio de soporte.
- Para el Data Warehouse se crea una tabla para la dimensión definida por el DataSet *Facturacion* y una tabla para el resultado del cruce de esta dimensión con la dimensión predefinida *Tiempo*. Además se crea una tabla para la Vista *LoginFacturacion*, que se corresponde al cruce de los dos DataSets definidos con la dimensión *Tiempo*.
- Como resultado, solamente para los mensajes enviados en la segunda prueba se insertan los datos en las tablas del Data Warehouse. Esto puede visualizarse en las Figuras 75, 76, 77 y 78.

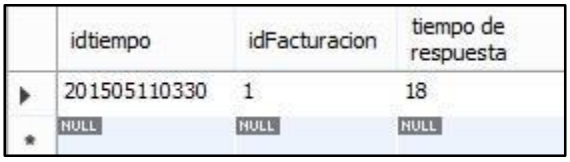


id	idDataSet	idServicio	nombreServicio	idOperacion	firmaOperacion	username	password	loginResult
1	40	7	urn:org.fing.proygr.esbetl:anteater-s...	11	login	marcos	defensor	tokenLogin
3	40	7	urn:org.fing.proygr.esbetl:anteater-s...	11	login	marcos	defensor	tokenLogin
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 75: Detalle tabla dimensión *Login*.

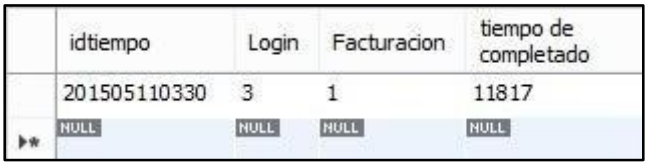
id	idDataSet	idServicio	nombreServicio	idOperacion	firmaOperacion	TipoFa	TipoFa	TipoF	Tipof	Tip	TipoFactura_Vendedor	Tip
1	41	8	urn:org.fing.proygr.esbetl...	12	agregarFactura	u1	c1	15	desc	8	marcos	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 76: Detalle tabla dimensión *Facturacion*.



idtiempo	idFacturacion	tiempo de respuesta
201505110330	1	18
NULL	NULL	NULL

Figura 77: Detalle tabla de hechos dimensión *TiempoFacturacion*.



idtiempo	Login	Facturacion	tiempo de completado
201505110330	3	1	11817
NULL	NULL	NULL	NULL

Figura 78: Detalle tabla dimensión *LoginFacturacion*.

6.3.3 Visualización de datos.

Con motivo de generar dos reportes sobre los datos se publica el cubo correspondiente a la Vista *LoginFacturacion* en el servidor OLAP de Pentaho. Ambos reportes se extraen utilizando la herramienta Saiku del servidor.

En la Figura 79 se detalla la consulta MDX generada para el primer reporte. En la Figura 80 se detalla el resultado obtenido.

```
WITH SET [~COLUMNS] AS
  { [Tiempo.HTiempo].[Fecha].Members }
SET [~ROWS_Facturacion.HFacturacion] AS
  Hierarchize (
    {
      { [Facturacion.HFacturacion].[Operacion].Members },
      { [Facturacion.HFacturacion].[Mensaje].Members }
    }
  )
SET [~ROWS_Login.HLogin] AS
  Hierarchize (
    {
      { [Login.HLogin].[Operacion].Members },
      { [Login.HLogin].[Mensaje].Members }
    }
  )
SELECT NON EMPTY CrossJoin (
  [~COLUMNS],
  { [Measures].[tiempo de completado] }
) ON COLUMNS,
NON EMPTY NonEmptyCrossJoin ( [~ROWS_Facturacion.HFacturacion], [~ROWS_Login.HLogin] ) ON ROWS
FROM [Vista_LoginFacturacion]
```

Figura 79: Consulta MDX reporte 1.

				201505122341
Operacion	Mensaje	Operacion	Mensaje	tiempo de completado
agregarFactura	15	login	21	10.358

Figura 80: Resultado reporte 1.

Finalmente se genera un nuevo reporte para visualizar los datos de los mensajes. En la Figura 81 se detalla la consulta MDX utilizada en la herramienta y en la Figura 82 se detallan los resultados obtenidos.

```
WITH MEMBER [Measures].[username] AS
  [Login.HLogin].[Mensaje].CurrentMember.Properties ( "username" )
MEMBER [Measures].[password] AS
  [Login.HLogin].[Mensaje].CurrentMember.Properties ( "password" )
MEMBER [Measures].[TipoFactura_Vendedor] AS
  [Facturacion.HFacturacion].[Mensaje].[Measures].CurrentMember.Properties ( "TipoFactura_Vendedor" )
SELECT NON EMPTY Crossjoin (
  [Tiempo.HTiempo].[Fecha].Members,
  { [Measures].[tiempo de completado], [Measures].[username], [Measures].[password], [Measures].[TipoFactura_Vendedor] }
) ON COLUMNS,
NON EMPTY NonEmptyCrossjoin ( [Login].[Mensaje].Members, [Facturacion].[Mensaje].Members ) ON ROWS
FROM [Vista_LoginFacturacion]
```

Figura 81: Consulta MDX reporte 2.

		201505122341			
Mensaje	Mensaje	tiempo de completado	username	password	TipoFactura_Vendedor
21	15	10.358	marcos	defensor	marcos

Figura 82: Resultado reporte 2.

6.4 Caso de estudio 3: Monitoreo de tres servicios relacionados entre sí.

El objetivo del tercer caso de estudio es analizar la información relacionada a tres servicios. Para ello se define un nuevo DataSet, se define una Vista que relacione los tres DataSets y finalmente se visualizan los datos obtenidos.

6.4.1 Pruebas realizadas.

Como punto de partida se reutilizan los DataSets *Login* y *Facturacion* definidos en los casos anteriores. Para comenzar se da de alta el servicio básico de información en la consola de administración, luego se crea un DataSet con el nombre *Identificación* incluyendo la operación *ObtPersonaPorDocumento* expuesta por dicho servicio.

Luego se crea una Vista con nombre *LoginFacturacionIdentificacion* que contiene los DataSets *Login*, *Facturación* e *Identificación*, adicionalmente se definen las siguientes condiciones de join:

- El parámetro *username* de la operación *login* debe coincidir con el parámetro *vendedor* indicado en el mensaje de la operación *agregarFactura*.
- El parámetro *nroDocumento* de la operación *ObtPersonaPorDocumento* debe coincidir con el parámetro *DocumentoUsuario* indicado en el mensaje de la operación *agregarFactura*.

En este punto se realizan dos pruebas:

- Se envía primero un mensaje a través de un cliente SOAP invocando la operación incluida en el DataSet *Login*, otro mensaje invocando a la operación incluida en el DataSet *Facturacion*, y finalmente otro mensaje invocando a la operación incluida en el DataSet *Identificación*. Se debe cumplir que el valor del parámetro *username* de la operación *login* sea diferente del valor del parámetro *vendedor* de la operación *agregarFactura*, y además el valor del parámetro *DocumentoUsuario* de la operación *agregarFactura* sea diferente del valor del parámetro *Nrodocumento* de la operación *ObtPersonaPorDocumento*.
- Se realiza la misma prueba que la anterior con la diferencia que todos los parámetros especificados para dichas operaciones sean iguales.

6.4.2 Resultados obtenidos.

Se obtienen los resultados esperados, los cuales se listan a continuación:

- Se almacena la información correspondiente al DataSet y la Vista en el repositorio de soporte.
- Para el Data Warehouse se crea una tabla para dimensión definida por el DataSet *Identificación* y una tabla para el resultado del cruce de esta dimensión con la dimensión *Tiempo*. Además se crea una tabla para la Vista

LoginFacturacionIdentificacion, que se corresponde al cruce de los tres DataSets definidos con la dimensión *Tiempo*.

- Como resultado, solamente para los mensajes enviados en la segunda prueba se insertan los datos en las tablas del Data Warehouse.
- Es posible visualizar dichos datos en la suite Pentaho.

7. Conclusiones y trabajo a futuro

7.1 Resumen y conclusiones finales

Como se indica en el Capítulo 1 el objetivo general del proyecto fue proponer mecanismos a incorporar en una plataforma de integración, basada en un ESB, que permitieran analizar y obtener información en tiempo real de los mensajes que circulan a través de la plataforma, almacenando dicha información en un Data Warehouse construido en base a configuraciones del usuario.

Se realizó un análisis detallado de la problemática planteada, que fue refinado a través de sucesivas reuniones con el cliente, obteniéndose una serie de requerimientos. A partir de estos requerimientos se definieron los conceptos básicos de la solución planteada: *Servicio*, *DataSet*, *Vista*. Luego se realizó un relevamiento de soluciones existentes que abordaran problemáticas similares. Sin embargo, no se logró encontrar otras soluciones que resolvieran la problemática planteada. No obstante, se encontraron trabajos cuyos enfoques sirvieron como punto de partida para comenzar el diseño de una solución, en particular enfoques asociados a Data Warehouse en tiempo real.

Luego se diseñó una solución que propone incorporar una serie de componentes a una plataforma de integración. Dicha solución fue refinada en distintas etapas a lo largo de todo el proyecto. Tal es así que se realizó un rediseño de la misma luego de haber completado una versión inicial del prototipo, pues no se estaba considerando de forma adecuada el enfoque de un usuario final del Data Warehouse. Además se construyó el Data Warehouse teniendo en cuenta que la visualización de los datos del mismo pudiera llevarse a cabo utilizando el servidor OLAP de la suite Pentaho.

Se realizó también la construcción de un prototipo para la solución diseñada aprovechando las tecnologías presentes en el ESB que se utilizó como plataforma de middleware. Sin embargo se tuvieron en cuenta otras tecnologías que pudieran incorporarse debido a que se adecuaban mejor a las necesidades del prototipo que las incluidas en el ESB, como fue el caso de Esper. Finalmente se puso a prueba el prototipo construido para un conjunto de casos de estudio.

Como resultado de todo este proceso, se arribó a las siguientes conclusiones finales, tanto del trabajo como de las tecnologías utilizadas.

- Una solución para la problemática planteada puede ser de mucha utilidad debido a que brinda a los usuarios un nuevo mecanismo para obtener información relevante que pueda ser utilizada en la toma de decisiones.
- Se planteó una solución que resuelve satisfactoriamente los objetivos planteados al principio del proyecto. Se logró construir un Data Warehouse cuyo proceso de carga se aproxima a real-time. El diseño de dicho Data Warehouse puede ser mejorado, para generar mejores tiempos de respuesta en las consultas, y proveer una mayor satisfacción al usuario final del mismo.
- Dentro de un ambiente controlado -por ejemplo de una organización- la solución propuesta es aplicable. No obstante, según la experiencia obtenida

a lo largo del proyecto parece complejo llevar este proyecto a un contexto "Internet" donde no existan restricciones de ningún tipo. Sin embargo es un buen punto de partida para lograrlo.

- Dentro de las tecnologías utilizadas se concluye que si bien la plataforma Switchyard es completa y cuenta con muchas funcionalidades interesantes, aún puede ser mejorada de acuerdo a nuestra experiencia en el uso de la misma.
- Las tecnologías CEP tienen mucha aplicabilidad para este tipo de problemáticas, y se debería profundizar su utilización en soluciones orientadas a resolver esta clase de problemas para brindar mejores prestaciones. Por otro lado para explotar al máximo la potencia de CEP, se requiere un conocimiento más profundo sobre estas tecnologías. En particular la plataforma Esper resultó ideal para este contexto.

7.2 Trabajo a futuro

Como trabajo a futuro se proponen una serie de mejoras que se detallan a continuación.

7.2.1 Calidad de datos

Un área que sería interesante indagar en el contexto de este proyecto es la de calidad de la información. La calidad de los datos que se almacenan en un Data Warehouse resulta de vital importancia, pues cuanto más confiable sea la información con la que se trabaja mejores serán los resultados de los análisis que se realicen sobre la misma.

En este punto sería interesante brindar al usuario la posibilidad de generar reglas para definir procesos de calidad. Por ejemplo, supongamos que los mensajes tienen un campo que siempre es un nombre, se podría contar con un tesauro de nombres y definir reglas de distancia para corregir errores sintácticos en la información del mensaje.

7.2.2 Diseño de la estructura del Data Warehouse

El prototipo implementado asigna tamaños y tipos de datos por defecto a las columnas de las tablas del Data Warehouse, según los tipos de datos de los atributos de los mensajes. Esto implica que el diseño de la base de datos del Data Warehouse no optimiza el uso del espacio.

Se podría refinar el proceso de construcción de la base de datos del Data Warehouse brindando al usuario, en la consola de administración, la posibilidad de indicar los tamaños de cada parámetro de las operaciones, al momento de dar de alta un servicio. De esta forma las columnas de las tablas del Data Warehouse tendrán tamaños más ajustados y el espacio del Data Warehouse sería más aproximado a los requerimientos reales de espacio del mismo.

7.2.3 Generación de medidas "on the fly"

En el prototipo implementado se crean algunas medidas por defecto que se calculan en base al arribo de los mensajes. Adicionalmente se brinda un diseño de Data

Warehouse tal que se le permite al usuario generar sus propias medida a la hora de visualizar los datos.

Por otro lado, sería interesante permitirle al usuario definir medidas “on the fly”, es decir al momento de generar una Vista o DataSet desde la consola web. Si se implementa esta nueva funcionalidad, se podrían incluir estas medidas en la estructura del cubo *Mondrian* generado para cada DataSet o Vista. El cubo contaría entonces con las medidas por defecto, junto con las medidas definidas por el usuario.

7.2.4 Mayor capacidad de procesamiento

Como se mostró en la solución planteada, se utiliza un único *Job* de carga que obtiene los mensajes a procesar desde la base intermedia.

Si se sigue en el camino de esta solución, y se utiliza un almacenamiento intermedio, sería interesante agregar paralelismo en el procesamiento de los mensajes para mejorar los tiempos de procesamiento. Para ello es necesario agregar *Jobs* de carga que trabajen en paralelo, utilizando mecanismos de concurrencia para la extracción de los mensajes de la base de datos.

7.2.5 Gestión de cambios.

Como se mencionó en la sección 4.5 una limitación que posee la solución propuesta es que no permite realizar ningún tipo de edición sobre la información dada de alta. Esto presenta un problema para el usuario dado que sería irreal suponer que los servicios ingresados en el sistema no presenten ningún tipo de cambio a través del tiempo.

Una posible mejora a futuro es diseñar un mecanismo que soporte cambios en los *Servicios*, y por consiguiente en las *Vistas* y los *DataSets*. Se propone implementar un módulo que maneje el versionado del Data Warehouse, de forma que ante una posible edición de los datos se cree una nueva versión del mismo. Sería ideal además que dicho módulo pudiera gestionar los cambios, para conservar la información almacenada en los casos en que el cambio no produzca ningún impacto sobre la misma, por ejemplo ante la modificación del nombre de un atributo.

7.2.6 Pruebas de estrés

Sería interesante además someter la solución a diversas pruebas de estrés para ver su respuesta tanto en contextos controlados como en contextos no controlados.

En particular se podrían obtener métricas sobre el rendimiento de carga logrado, diversificando la cantidad de *Jobs* en un bombardeo de mensajes de diferentes aplicaciones, tanto para mensajes de interés como para mensajes que no lo son.

De esto último podrían surgir nuevas ideas de mejoras y refinamiento de la solución, tanto en cuanto a lo inherente a procesamiento y comunicación entre los componentes como en el diseño logrado para el Data Warehouse.

7.2.7 Pre procesamiento de los mensajes y no procesamiento “on the fly”

El enfoque que manejamos tiene la característica de que cuando se consulta un servicio, y se envían mensajes, estos mensajes son almacenados en una base intermedia no relacional, específicamente en una base de datos *Mongo*. Luego se envía una notificación a uno de los componentes que consulta si hay mensajes sin procesar almacenados, los obtiene y los procesa.

Por lo tanto, el procesamiento no es “on the fly” sino que primero se almacenan los mensajes para ser procesados luego. Esto provoca que se generen pequeñas demoras entre que un mensaje llega al ESB y es procesado.

Como mejora debería existir un procesamiento al momento en que el mensaje pasa por el ESB, pero para ello habría que estudiar la velocidad de procesamiento pertinente para evitar la posible pérdida de mensajes. Además se podrían generar varios hilos de procesamiento para dar soporte al procesamiento en paralelo de los mensajes que ingresen a la plataforma.

8 Referencias

[1] Application to Application (A2A) and Business to Business (B2B) integration. [Online]. Available, last accessed on May 2015: http://integrella.com/services/solutions/a2a-and-b2b-integration/

[2] XML Namespaces. [Online]. Available, last accessed on May 2015: http://www.w3schools.com/xml/xml_namespaces.asp

[3] An XML Indexing Structure with Relative Region Coordinate. [Online]. Available, last accessed on May 2015: https://wiki.csc.calpoly.edu/csc560/wiki/Relative%20Region%20Coordinate

[4] XML Path Language (Xpath). [Online]. Available, last accessed on May 2015: http://www.w3.org/TR/xpath/

[5] Transparencias XML, Curso de Middleware 2014. Fing, Udelar.

[6] Desarrollo de aplicaciones con enfoque SOA (Service Oriented Architecture).

[Online]. Available, last accessed on May 2015: http://www.fing.edu.uy/~fpiedrab/downloads/JIISIC06_SOA_AD_v6.6_.pdf

[7] Arquitectura Orientada a Servicios (SOA). [Online]. Available, last accessed on May 2015: http://www.accenture.com/SiteCollectionDocuments/Local_Spain/PDF/SOA.pdf

[8] World Wide Web Consortium (W3C). [Online]. Available, last accessed on May 2015: http://www.w3.org/

[9] Web Services Architecture. [Online]. Available, last accessed on May 2015: http://www.w3.org/TR/ws-arch/

[10] Web Services Description Language (WSDL). [Online]. Available, last accessed on May 2015: http://www.w3.org/TR/wsdl

[11] SOAP Syntax. [Online]. Available, last accessed on May 2015: http://www.w3schools.com/webservices/ws_soap_syntax.asp

[12] SOAP Header Element. [Online]. Available, last accessed on May 2015: http://www.w3schools.com/webservices/ws_soap_header.asp

[13] SOAP Body Element. [Online]. Available, last accessed on May 2015: http://www.w3schools.com/webservices/ws_soap_body.asp

[14] Transparencias Enterprise Service Bus, Curso de Middleware. Fing, Udelar. [Online]. Available, last accessed on May 2015: https://eva.fing.edu.uy/pluginfile.php/69594/mod_resource/content/2/Enterprise%20Service%20Bus.pdf

[15] ESB Tutorial. [Online]. Available, last accessed on May 2015: http://searchsoa.techtarget.com/tutorial/ESB-Tutorial

[16] Introduction to the Microsoft ESB Guidance. [Online]. Available, last accessed on May 2015: https://msdn.microsoft.com/en-us/library/ff648282.aspx

[17] Chappell, David. 2004. Enterprise Service Bus: Theory in Practice. O'Reilly Media. [Online]. Available, last accessed on May 2015: https://www.safaribooksonline.com/library/view/enterprise-service-bus/0596006756/ch01s07.html

[18] Enterprise Integration Patterns. [Online]. Available, last accessed on May 2015: http://www.enterpriseintegrationpatterns.com/toc.html

[19] Service Virtualization. [Online]. Available, last accessed on May 2015: http://www.ibm.com/developerworks/library/ws-enterpriseconnectivitypatterns

[20] Plataforma ESB Adaptativa para Sistemas Basados en Servicios. [Online]. Available, last accessed on May 2015: https://www.fing.edu.uy/~lauragon/tesis/tesis-laura-gonzalez-final.pdf

[21] Complex Event Processing (CEP) Definition. [Online]. Available, last accessed on May 2015: http://searchsoa.techtarget.com/definition/complex-event-processing

[22] Event Processing for Business Organizing the Real- Time Enterprise. Autor: DAVID LUCKHAM Copyright © 2012. [Online]. Available, last accessed on May 2015: https://www.safaribooksonline.com/library/view/event-processing-for/9781118171851/xhtml/copyright.html

[23] Real Time Intelligence & Complex Event Processing. [Online]. Available, last accessed on May 2015: http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2

[24] A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. [Online]. Available, last accessed on May 2015: http://www.en.pms.ifi.lmu.de/publications/PMS-FB/PMS-FB-2010-8/PMS-FB-2010-8-paper.pdf

[25]Data Warehouse. [Online]. Available, last accessed on May 2015: http://www.sinnexus.com/business_intelligence/datawarehouse.aspx

[26] Data Warehouse solution. [Online]. Available, last accessed on May 2015: http://savis.vn/solution/data-warehouse-solution/

[27] William H.Inmon. Building the data warehouse. Wiley, fourth edition edition, 2005. [Online]. Available, last accessed on May 2015: http://users.itk.ppke.hu/~szoer/DW/Inmon%20-%20Building%20The%20Data%20Warehouse%203rd%20Ed%20%5BWiley%202003%5D.pdf

[28] Pentaho official site. [Online]. Available, last accessed on May 2015: http://www.pentaho.com/

[29] Sistemas de Data Warehousing: Diseño Conceptual. [Online]. Available, last accessed on May 2015: http://www.fing.edu.uy/inco/grupos/csi/esp/Cursos/cursos_act/2003/DAP_Sist DW/Material/2-SDW-Conceptual-2003.pdf

[30] Implantación de Data Warehouse Open Free. [Online]. Available, last accessed on May 2015: http://www.fing.edu.uy/~asabigue/prgrado/2010dw.pdf

[31] Sistema de Gestión II Data Warehouse. [Online]. Available, last accessed on May 2015: http://www.edutecne.utn.edu.ar/sist-gestion-II/Apunte%20BI.pdf

[32] Margy Ross Ralph Kimball. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. Wiley Computer Publishing, second edition, 2002. [Online]. Available, last accessed on May 2015: http://home.elka.pw.edu.pl/~rbzoma/The%20Data%20Warehouse%20Toolkit%20-%20The%20Complete%20Guide%20to%20Dimensional%20Modeling%20(2nd%20Ed%202002%20Wiley)%20-%20Kimball%20&%20Ross.pdf

[33] Considerations for Building a Real-time Data Warehouse. [Online]. Available, last accessed on May 2015: http://www.grcdi.nl/considerations.pdf

[34] On-The-Fly Generation of Multidimensional Data Cubes for Web of Things. [Online]. Available, last accessed on May 2015: http://www.edwardcurry.org/publications/MMehdi_ideas13.pdf

[35] Zhou, Honbo. The Internet of Things in the Cloud: A Middleware Perspective. [Online]. Available, last accessed on May 2015: https://books.google.com.uy/books?id=nQw6B9ziOhQC&pg=PT150&dq=%22web+of+things%22&hl=es&sa=X&ei=9h0rVcXrBMGaNoeZgEA&ved=0CEwQ6AEwBg#v=onepage&q=%22web%20of%20things%22&f=false

[36] Near Real-Time Data Warehousing Using State-of-the-Art ETL Tools. [Online]. Available, last accessed on May 2015: http://wwwlgis.informatik.uni-kl.de/cms/fileadmin/users/joerg/documents/joergBIRTE09.pdf

[37] OLAP Cube Member Properties. [Online]. Available, last accessed on May 2015: http://diethardsteiner.github.io/mondrian/2014/12/29/Mondrian-Member-Properties.html

[38] Introduction to Addressing. [Online]. Available, last accessed on May 2015: http://www.oracle.com/au/products/database/ws-addressing-intro-087619.html

[39] Apache Maven official site. [Online]. Available, last accessed on May 2015: https://maven.apache.org/

[40] MySQL 5.7 Reference Manual. [Online]. Available, last accessed on May 2015: http://dev.mySQL.com/doc/refman/5.7/en/index.html

[41] HIBERNATE - Persistencia relacional para Java idiomático. [Online]. Available, last accessed on May 2015: https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/index.html

[42] Eclipselink documentation. [Online]. Available, last accessed on May 2015: http://eclipse.org/eclipselink/documentation/

[43] Red Hat Jboss Enterprise Application Platform. [Online]. Available, last accessed on May 2015: http://www.jboss.org/products/eap/overview/

[44] JDBC: acceso a bases de datos. [Online]. Available, last accessed on May 2015: http://www.vc.ehu.es/jiwotvim/ISOFT2009-2010/Teoria/BloqueIV/JDBC.pdf

[45] MongoDB official site. [Online]. Available, last accessed on May 2015: http://www.mongodb.com/es

[46] Switchyard 1.1 documentation. [Online]. Available, last accessed on May 2015: https://docs.jboss.org/author/display/SWITCHYARD11/Home

[47] ESB-adaptativo. [Online]. Available, last accessed on May 2015: https://code.google.com/p/esb-adaptativo/

[48] Enterprise Integration Patterns. [Online]. Available, last accessed on May 2015: http://www.eaipatterns.com/toc.html

[49] Apache Camel Enterprise Integration Patterns. [Online]. Available, last accessed on May 2015: http://camel.apache.org/enterprise-integration-patterns.html

[50] Esper documentation. [Online]. Available, last accessed on May 2015: http://www.espertech.com/esper/release-5.2.0/esper-reference/html/technology_overview.html

[51] Quartz official site. [Online]. Available, last accessed on May 2015: http://quartz-scheduler.org/overview/whos-using-quartz

[52] Pentaho Mondrian documentation. [Online]. Available, last accessed on May 2015: http://mondrian.pentaho.com/documentation/schema.php

[53] Servicio Básico de Información. [Online]. Available, last accessed on May 2015: http://www.agesic.gub.uy/innovaportal/v/1799/1/agesic/servicio_basico_de_info

Apéndice 1. Primer diseño del Data Warehouse

Se realizó un primer diseño del Data Warehouse el cual se construía siguiendo el mismo proceso que se detalló en la sección 4.3, pero con algunas diferencias en las decisiones tomadas para su diseño.

Al igual que en el diseño definitivo, un DataSet se traduce en una dimensión del Data Warehouse (como puede verse en la Figura 83), pero en este diseño cuenta con la jerarquía formada por los niveles: parámetros, operaciones, servicio.

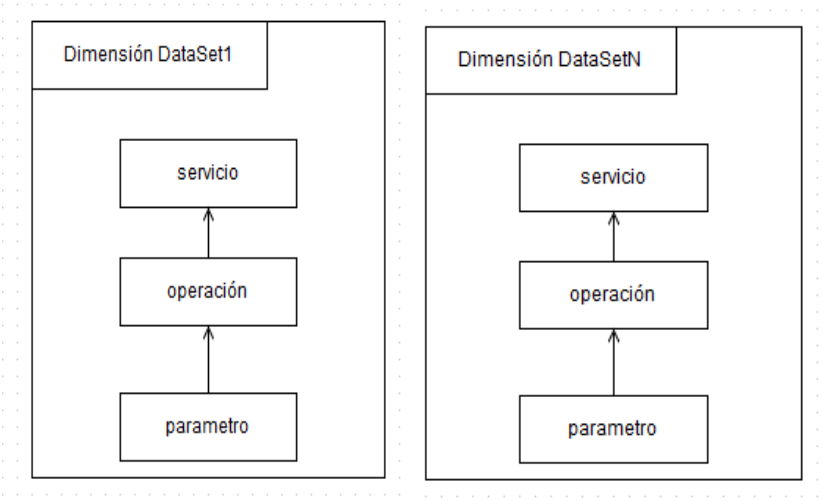


Figura 83: Jerarquías de las dimensiones generadas.

Además por defecto se mantenía la dimensión Tiempo (como puede verse en la Figura 84) con nivel jerárquico fecha, día, mes, año, debido a que el principal interés de los análisis de información serán con respecto al tiempo.

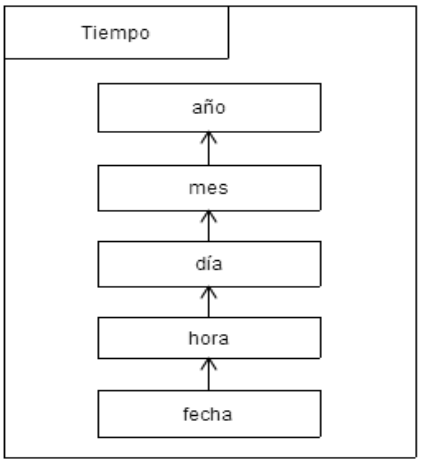


Figura 84: Jerarquía de las dimensión Tiempo.

Cuando el usuario guarda el DataSet en la consola de administración web, se crea una tabla para representar la dimensión formada por dicho DataSet en el Data Warehouse. También se crea una tabla que representa a la dimensión Tiempo. Por último, se crea la tabla de hechos que asocia estas dos dimensiones. El diagrama

conceptual de este cruce se presenta en la Figura 85, mientras que el diseño lógico se muestra en la Figura 86.

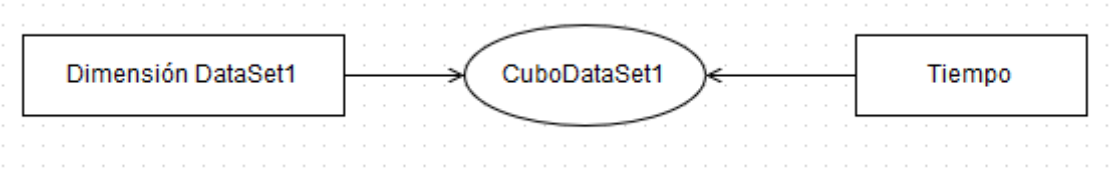


Figura 85: Diseño conceptual DataSet-Tiempo

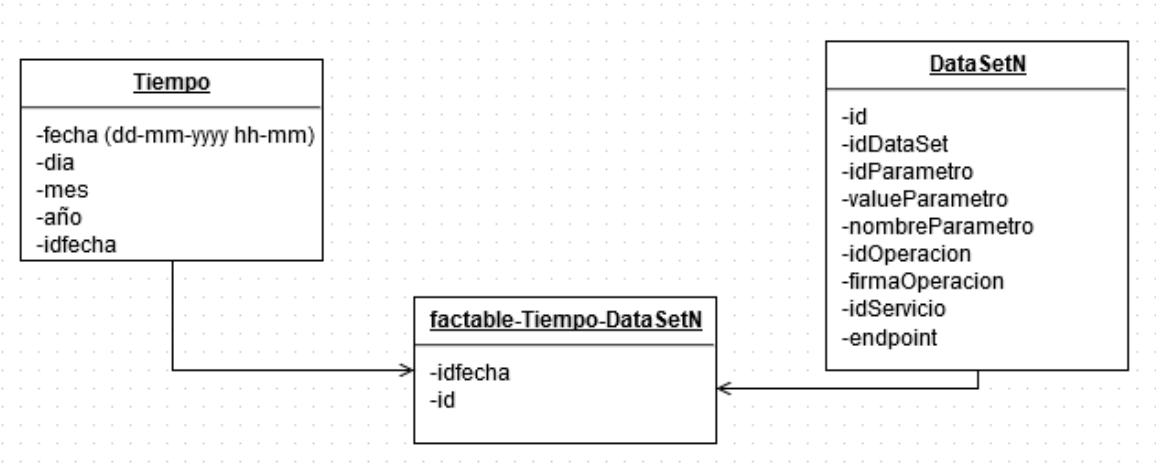


Figura 86: Diseño lógico DataSet-Tiempo

Para la creación de las Vistas, al igual que en el diseño presentado en la solución, se crea una tabla de hechos que asocia cada una de las dimensiones correspondientes a los DataSets seleccionados y la dimensión *Tiempo*. En la figuras 87 y 88 se detallan los diseños conceptual y lógico correspondientes.

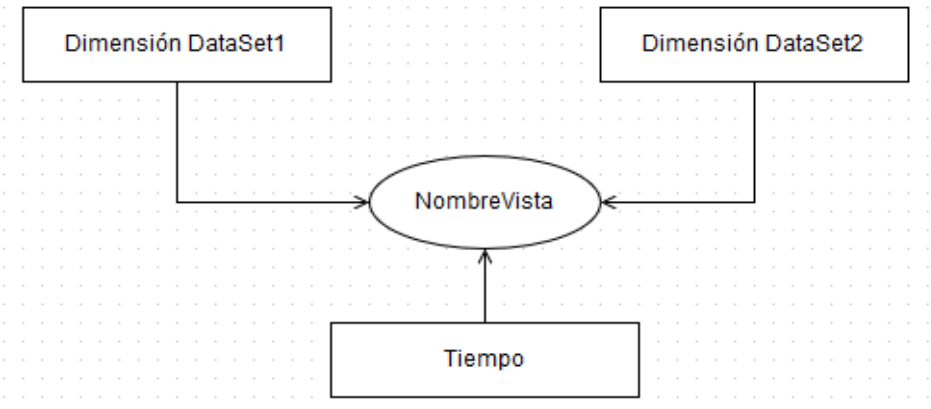


Figura 87: Diseño conceptual DataSet-Tiempo

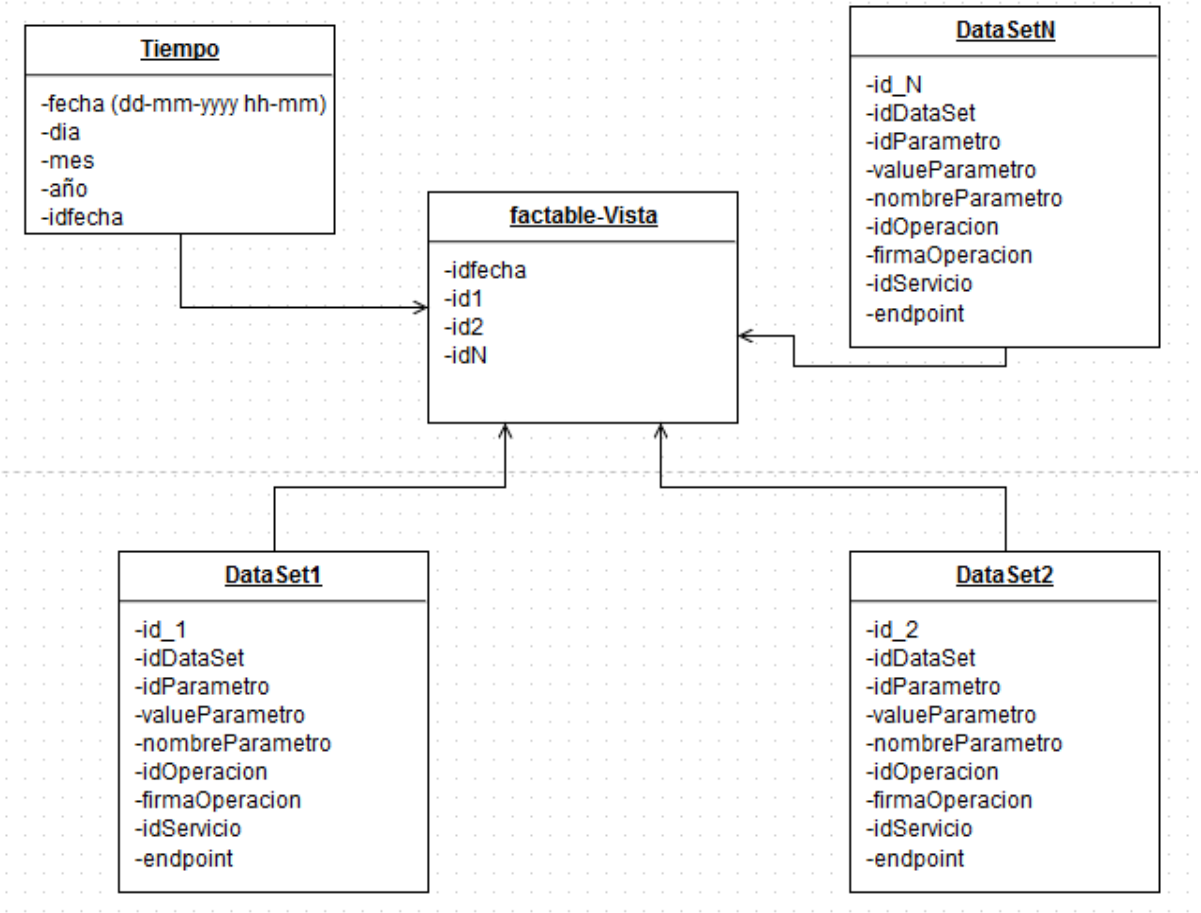


Figura 88: Diseño lógico DataSet-Tiempo

La diferencia con respecto al diseño final de la solución radica en la jerarquía utilizada en la dimensión DataSet. Esta jerarquía presenta un nivel *parámetros* que en el diseño final no está presente, y en su lugar se tiene el nivel *mensaje.*
Esta diferencia se traduce en el diseño lógico cuando se mapea el diseño conceptual a tablas de la base de datos, ya que se obtiene una estructura diferente.
En el caso de este diseño primario, se tienen los campos *valueParametro* y *nombreParametro* para almacenar el valor y nombre de cada parámetro de la operación. Esto hace que para un mensaje, se tengan varias entradas en la tabla, una para cada parámetro de la operación invocada.
En cambio, en el diseño final se tiene una sola entrada por mensaje, ya que cada parámetro representa un campo de la tabla.

El diseño primario fue descartado debido a que esta estructura dificulta la posibilidad de poder utilizar los valores del mensaje como medidas a la hora de definir un cubo para consultar estos datos.

Se presenta a continuación, a través de un ejemplo por qué este diseño tiene dicha dificultad.
Se analiza el caso del servicio de básico de información publicado por la DNIC. Dicho servicio tiene una operación *ObtenerPersonaPorDocumento* que brinda información sobre las personas.

Se acotan en este ejemplo la cantidad de parámetros de dicha operación a los efectos de explicar la problemática que se desea plantear.
Se supone que se obtienen para cada persona consultada:

- Nombre
- Apellido
- Número de documento

Un mensaje para dicho servicio se mapea en el Data Warehouse de la forma indicada en las Figura 89 y 90.

id	Id Mensaje	Id Servicio	Servicio	Id Operacion	Operación	idParametro	Parámetro	ValorParametro
1	1	1	Identificación	1	ObtPersona	1	Nombre	Juan
2	1	1	Identificación	1	ObtPersona	2	Apellido	Pérez
3	1	1	Identificación	1	ObtPersona	3	NroDoc	45716923

Figura 89: Valores almacenados para el mensaje correspondiente al DataSet *ObtenerPersonaPorDocumento*

idTiempo	IdObtenerPeronsaPorDoc
201403032234	1
201403032234	2
201403032234	3

Figura 90: Valores almacenados para el mensaje correspondiente a la Vista *ObtenerPersonaPorDocumento*

Ahora suponiendo que al analista de reportes del Data Warehouse le interesa saber cuántas veces se consultó el usuario de cuyo *NroDoc* es *45716923*.
Para definir métricas sobre este cubo formado por la dimensión Tiempo y la dimensión *DataSet_ObtenerPersonaPorDocumento* habría que poder generar una consulta que a cada fila cuyo valor en la columna Parámetro sea diferente de *NroDoc* asigne un valor 0 a esa medida y para los que el Parámetro sea *NroDoc* y el *ValorParametro* sea *45716923* asigne 1 a esa medida.

Esta consulta es compleja de hacer, y no parece que sea eficiente por la cantidad de comparaciones que debe hacer.

A su vez la complejidad de la consulta aumenta proporcionalmente con la complejidad de la métrica que se desea obtener.
Por este motivo se planteó que si el enfoque se debe centrar en obtener métricas sobre la información de los parámetros esta construcción no es la más adecuada.
Por otro lado las Vistas presentan un problema mayor. Se debería obtener un match para una Vista tomando en cuenta el mensaje en su totalidad y no cada uno de los parámetros, por ende en la tabla de una Vista se debería considerar el id de un mensaje que no es la clave primaria de ninguna fila de las tablas de los DataSets.
Finalmente el diseño no contemplaba las medidas por defecto. Esto no hubiera permitido publicarlo en el servidor OLAP de la suite Pentaho.
Debido a estas razones se tomó la decisión de realizar el diseño detallado en la sección 4.3.

Apéndice 2 Glosario

API Application Programming Interface.

BPMN Business Process Model and Notation.

CEP Complex Event Processing. Procesamiento de eventos complejos.

CRM Customer Relationship Management. Es una sigla que se usa para definir un sistema de gestión de negocios enfocada en la relación de una organización con sus clientes.

CSS Cascading Style Sheets. Estilos en cascada, es un lenguaje utilizado para cambiar la presentación de un elemento en un sitio web, o aplicación.

DB Database, base de datos en español.

DW Data Warehouse.

DWs Data Warehousing.

EAP Enterprise Application Platform. Es una plataforma de Jboss.

EIP Enterprise Integration Patterns. Refiere a un conjunto de patrones de integración entre plataformas empresariales presentado en un libro escrito por Martin Fowler, que justamente presenta el mismo nombre.

EPL Event Processing Language. Lenguaje de alto nivel utilizado para definir eventos complejos.

ESB Enterprise Service Bus.

ETL Extracción Transformación y Carga.

HTML HyperText Markup Language. Lenguaje utilizado en la construcción de páginas web.

HTTP HyperText Transfer Protocol.

JDBC Java Database Connectivity.

JMS Java Message Service.

JSF Java Server Faces.

MDX Multidimensional Expresión.

MTSS Ministerio de Trabajo y Seguridad Social.

OLAP Online Analytical Processing.

POM Project Object Model.

REKB Registered Events Knowledge Base.

REST	Representational State Transfer.
RQ	Requerimiento.
SCA	Service Component Architecture.
SOA	Service Oriented Architecture.
SOAP	Simple Object Access Protocol.
SQL	Structured Query Language.
UDDI	Universal Description, Discovery and Integration.
UI	User Interface.
WSDL	Web Services Description Language.
WOT	Web of Things.
XML	Extensible Markup Language.
XPath	XML Path Language.