

Instituto de Computación - Facultad de Ingeniería

Universidad de la República

Montevideo, Uruguay

Calidad de Datos en Enterprise Service Bus

Informe de Proyecto de Grado

Rodrigo Mendoza Balestena
Mauricio Juan Piñatares Cabral
Nicolas Agustin Granja Gervaso

Mayo, 2016

Docente Supervisor: Ing. Laura González
Docente Supervisor Alterno: Ing. Adriana Marotta
Usuario Responsable: Ing. Silvana Pidre

Resumen

Cada vez son más frecuentes los sistemas de información a gran escala que interconectan sistemas de software de distintas organizaciones autónomas. Estos a menudo presentan patrones de interacción inter-organizacional con el fin de invocar funcionalidades de negocio e intercambiar información entre dichas organizaciones. En este contexto de intercambio de información, donde coexisten formatos, estándares y características diferentes, es de suma importancia garantizar la calidad de los datos que se intercambian.

Este intercambio de datos entre organizaciones generalmente se apoya fuertemente en plataformas de integración. Por tal motivo, resulta conveniente contar con mecanismos para monitorear y asegurar la calidad de los datos que fluyen en dichas plataformas.

Las plataformas de integración comúnmente están basadas en tecnologías de middleware avanzadas como los Enterprise Service Bus (ESB) y brindan una capa de procesamiento intermedio que provee distintas capacidades avanzadas (por ejemplo seguridad, ruteo, entre otras) para facilitar la interacción entre las organizaciones. Sin embargo, actualmente no ofrecen nativamente mecanismos que permitan monitorear y asegurar la calidad de los datos que se intercambian a través de ellas.

El presente proyecto propone soluciones para monitorear y garantizar la calidad de los datos que se intercambian entre organizaciones en la invocación de servicios, a través de una plataforma de integración del tipo ESB, utilizando las capacidades que estas ofrecen.

En primer lugar se realizó un análisis de la problemática con el fin de identificar los requerimientos que una solución de este tipo debería cumplir. Dicho análisis incluyó el estudio de trabajos relacionados con este proyecto.

Luego, se diseñó una solución extensible que permite garantizar la calidad de datos que fluyen por las plataformas ESB, proponiendo el uso de métodos para la evaluación de calidad y acciones a tomar para garantizar el cumplimiento de la misma. Luego se implementó un prototipo utilizando tecnologías de código abierto, seleccionando como herramienta base a Switchyard, para verificar la viabilidad de la solución conceptual brindada.

Además, se definió un caso de estudio en el contexto de una plataforma de salud, y en base a este se especificaron pruebas que permitieron evaluar el comportamiento de la plataforma, para validar que el prototipo implementado cumpliera con los requerimientos identificados para la solución.

Por último, se realizaron pruebas de performance para determinar el *overhead* introducido por la solución implementada en el prototipo. En los resultados de estas pruebas se puede observar que la sobrecarga producida por el manejo de calidad de los datos no es considerable con respecto a las mejoras que se generan.

Palabras Claves: *Calidad de datos, Enterprise Service Bus, web services, modelos de calidad.*

Contenido

Resumen	3
1 Introducción	7
1.1 Contexto y motivación	7
1.2 Objetivos	7
1.3 Aportes del proyecto.....	8
1.4 Organización del documento	8
2 Marco Conceptual.....	11
2.1 Calidad de datos	11
2.2 Enterprise Service Bus	13
2.3 Web Services	17
3 Análisis	23
3.1 Marco de Trabajo	23
3.2 Análisis del Problema	25
3.3 Requerimientos Identificados	27
3.4 Trabajos relacionados	30
3.5 Conclusiones.....	31
4 Solución propuesta	33
4.1 Descripción General.....	33
4.2 Modelo Conceptual.....	36
4.3 Principales características	38
4.4 Arquitectura lógica.....	43
4.5 Flexibilidad y extensibilidad	50
5 Detalles de Implementación del prototipo.....	51
5.1 Tecnologías utilizadas	51
5.2 Plataforma ESB	55
5.3 Backoffice.....	59
5.4 Extensibilidad	66
5.5 Flexibilidad	68
5.6 Problemas Encontrados	69

5.7 Limitaciones	70
6 Caso de Estudio y Pruebas Realizadas	73
6.1 Caso de Estudio: Plataforma de Salud	73
6.2 Servicio Utilizado	74
6.3 Modelo de Calidad utilizado.....	74
6.4 Configuración del caso de prueba	77
6.5 Ejecución del caso de estudio	78
6.6 Pruebas de <i>performance</i>	86
6.7 Conclusiones.....	93
7. Conclusiones y Trabajo a Futuro	95
7.1 Conclusiones.....	95
7.2 Trabajo a Futuro.....	96
8 Referencias.....	99
Apéndice 1. Registro para reportes.....	103
Apéndice 2. Inicialización de la estructura interna.....	105
Apéndice 3. Transformaciones XSL	107
Apéndice 4. Manejo de errores	109
Apéndice 5. Métodos	111
Apéndice 6. Implementación de una Acción.....	113
Apéndice 7. Ejemplo de Método por Reflection.....	115
Apéndice 8. Operaciones utilizadas en el caso de estudio.....	117
Apéndice 9. Propiedades globales.....	121
Apéndice 10. Métricas del caso de estudio	123
Apéndice 11. Métodos implementados con reglas	127
Apéndice 12. Tablas de referencia del servicio Obtener Certificados.....	131

1 Introducción

En este documento se presentan los resultados del proyecto de grado denominado “Calidad de Datos en Enterprise Service Bus”, el cual está enmarcado en la carrera de Ingeniería en Computación de la Facultad de Ingeniería de la Universidad de la República.

En este capítulo se presenta el contexto y motivación del proyecto, se plantean los objetivos y aportes del mismo, y se presenta cómo está organizado el resto del documento.

1.1 Contexto y motivación

Cada vez son más frecuentes los sistemas de información a gran escala que interconectan los sistemas de *software* entre organizaciones autónomas. Estos a menudo presentan patrones de interacción inter-organizacional con el fin de invocar funcionalidades de negocio e intercambiar información entre organizaciones [1].

En la actualidad, el enfoque preferido para realizar integraciones en estos contextos inter-organizacionales es mediante arquitecturas orientadas a servicios (Service Oriented Architecture, SOA). En este tipo de arquitecturas, la cooperación entre las distintas organizaciones se logra compartiendo e integrando servicios de *software* a través de la red [1].

Por otro lado, la integración de servicios en estos contextos se apoya cada vez más en infraestructuras especializadas de *hardware* y *software*, comúnmente denominadas Plataformas de Integración (PI). Estas plataformas brindan una capa de procesamiento intermedio, con lógica de integración y comunicación reutilizable, a través de la cual se comunican los servicios. La puesta en marcha de estas plataformas se basa comúnmente en tecnologías de *middleware* avanzadas, en particular, en productos de tipo Enterprise Service Bus (ESB) [2].

Además, en este contexto de intercambio de información, donde coexisten formatos, estándares y características diferentes, es de suma importancia garantizar la calidad de los datos que intercambian las organizaciones. Por tal motivo, resulta conveniente contar con mecanismos para monitorear y asegurar la calidad de los datos que fluyen en dichas plataformas.

Según lo mencionado anteriormente, el ESB resulta un lugar ideal en donde implantar una infraestructura para monitorear y asegurar la calidad de los datos debido a su rol mediador en una SOA. Sin embargo, las tecnologías y plataformas de integración actuales no proveen soluciones nativas y completas que aborden requerimientos de monitoreo y aseguramiento de calidad de los datos [3].

1.2 Objetivos

El objetivo general de este proyecto es proponer y diseñar mecanismos a incorporar en una plataforma de integración del tipo ESB, que permitan monitorear y garantizar la calidad de los datos que se intercambian entre distintas organizaciones en la invocación de servicios.

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

- Analizar los requerimientos de una plataforma que permita monitorear y garantizar la calidad de los datos que se intercambian entre las organizaciones.
- Proponer una plataforma basada en ESB que cumpla con los requerimientos propuestos.
- Implementar un prototipo de la solución propuesta, utilizando Java para su desarrollo y el producto ESB que se considere conveniente.
- Implementar una herramienta que facilite la administración y configuración de los componentes que permiten monitorear y asegurar la calidad de los datos en la Plataforma ESB.
- Definir al menos un caso de estudio relacionado al área de la Salud, que permita realizar pruebas para verificar el correcto funcionamiento del prototipo y validar la solución.

1.3 Aportes del proyecto

En el presente proyecto se realizan los siguientes aportes en cuanto a la problemática de garantizar la calidad de datos que se intercambian en una plataforma ESB:

- Análisis de requerimientos para proveer a una plataforma de integración de tipo ESB, mecanismos para monitorear y garantizar la calidad de los mensajes que circulan a través de la misma.
- Propuesta de solución que consta de un diseño conceptual, y del diseño de una arquitectura flexible y extensible para soportar dicha propuesta.
- Implementación de un prototipo basado en el diseño conceptual, que utiliza la arquitectura diseñada. El prototipo fue desarrollado en Java utilizando SwitchYard como ESB.
- Implementación de una consola de administración web que facilita administrar, configurar y monitorear los componentes que permiten aplicar controles de calidad sobre los mensajes que pasan a través de la plataforma ESB.
- Desarrollo de un caso de estudio relacionado al área de Salud que comprueba el correcto funcionamiento del prototipo y brinda una primera validación de la solución propuesta.
- Evaluación y análisis sobre las herramientas utilizadas, problemas encontrados durante la implementación y limitaciones del prototipo.
- Realización de pruebas que permitieron evaluar la *performance* de la solución implementada.

1.4 Organización del documento

El resto del documento se organiza de la siguiente manera.

En el capítulo 2, se presenta un resumen del conocimiento existente relevante a la problemática abordada en este proyecto.

En el capítulo 3, se realiza un análisis sobre la problemática planteada y se identifican los requerimientos para monitorear y garantizar la calidad de los datos que se intercambian entre las organizaciones, utilizando las capacidades de mediación de los ESB. Por último, se describen algunos trabajos relacionados a la temática desarrollada en el presente proyecto.

En el capítulo 4, se propone una solución y se presenta la plataforma a implementar, para la cual se especifican sus características generales, sus componentes y sus principales interacciones. Conjuntamente, en este capítulo se presenta cómo se implementan los distintos componentes necesarios en el ESB para garantizar la calidad de los datos dentro del mismo.

En el capítulo 5, se describe detalladamente la implementación de los principales componentes de la Plataforma ESB comentando algunas de las problemáticas presentadas durante la etapa de implementación de este proyecto.

En el capítulo 6, se presenta el caso de estudio de Gobierno Electrónico en el área de Salud, que proporciona un contexto para evaluar el desempeño de la plataforma. Además, se describe cada una de las pruebas a las que fue sometida la plataforma.

Finalmente, en el capítulo 7, se presentan las conclusiones del proyecto, resumiendo el trabajo realizado y comentando posibles trabajos a futuro.

2 Marco Conceptual

En este capítulo se describen los principales conceptos teóricos que aplican a la problemática planteada para este proyecto y que serán de utilidad para comprender de mejor forma la solución planteada. Dichos conceptos son: Calidad de Datos, Enterprise Service Bus y Web Services.

2.1 Calidad de datos

2.1.1 Descripción General

En casi todos los sistemas y tecnologías emergentes, la calidad de los datos tiene un papel importante, por lo cual se han propuesto, en sistemas de información individuales, metodologías para modelar los requisitos específicos de calidad.

La mala calidad de los datos influye de manera significativa y profunda en la efectividad y eficiencia de las organizaciones, así como en todo el negocio, causando grandes pérdidas económicas. En el contexto de una empresa, la falta de datos críticos o la presencia de valores incorrectos en los mismos, podría afectar negativamente los procesos de negocio clave, causando su detención u otros resultados no deseados. Por otra parte, una mala calidad en los datos podría resultar en toma de decisiones inapropiadas para la estrategia de negocio [4].

Cuando múltiples fuentes de datos tienen que estar integradas, la necesidad de una buena calidad de los datos es aún mayor. Por ejemplo, en un escenario de Gobierno Electrónico donde distintos organismos públicos cooperan con el fin de cumplir con las solicitudes de los ciudadanos y las empresas, existen organismos que, debido a que no conocen la calidad de los datos que almacenan los otros organismos, prefieren pedir estos datos a los ciudadanos cada vez que los necesitan.

Es importante comprender el concepto de calidad de datos para entender mejor el problema planteado y la solución implementada. Se puede entender a los datos como colecciones de objetos que representan un estado de la realidad. Como toda representación, la interpretación de su significado puede diferir entre quienes la accedan. Por esto es necesario que quienes interactúan con los datos, los entiendan de forma similar.

La calidad de los datos se puede definir como la distancia que existe entre los datos que son manejados por un sistema de información y los mismos datos en el mundo real [5].

2.1.2 Modelo de Calidad

Un modelo de calidad de datos representa el conjunto de dimensiones y otros aspectos de la calidad (tales como factores, métricas y métodos de medición) y las relaciones que existen

entre ellos. Proporciona la base para especificar y evaluar la calidad de los datos. Para cada caso particular, se podría llegar a definir un modelo de calidad válido [6].

Para poder tratar la calidad de un sistema de información, es necesario definir un modelo de calidad adecuado a las necesidades y prioridades de los consumidores de los datos en ese sistema de información. Por medio de un modelo de calidad se puede determinar y especificar las dimensiones, factores a medir, métricas y datos o grupos de datos donde éstas se aplican, y por último, bases de metadatos de calidad donde se registrarán los resultados de la medición.

En la Figura 2.1 puede observarse la estructura de los principales conceptos de Calidad de Datos.

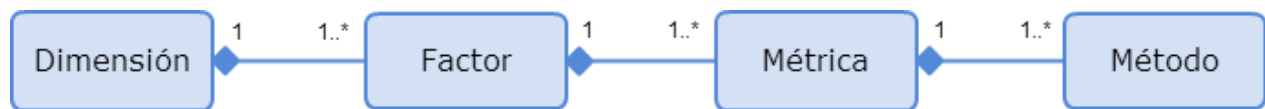


Figura 2.1. Abstracción de los principales conceptos de Calidad de Datos

Dimensión de calidad

Una dimensión representa una faceta o propiedad de la calidad. Algunos ejemplos de dimensión de calidad son:

- **Frescura:** La frescura de los datos (*data freshness*) representa la “edad” de los datos, indicando qué tan “viejos” son y si son “apropiados” para un determinado requerimiento [7]. Se mide como el tiempo transcurrido entre el momento en que los datos fueron producidos y el momento en que son retornados al usuario en respuesta a una consulta.
- **Exactitud:** La exactitud consiste en que los datos registrados están en conformidad con los datos reales [8].
- **Compleitud:** La completitud de los datos representan que son de suficiente alcance y profundidad [8].
- **Consistencia:** La consistencia indica que los datos satisfacen las reglas semánticas definidas sobre ellos mismos o, dicho de otra forma, los valores de los datos son representados igual en todos los casos [8].

Para ayudar a comprender mejor el concepto se plantea un ejemplo que refiere a un registro *Ciudadano*, con los campos *Nombre*, *Sexo* y *Email*, como se muestra en la Tabla 2.1.

Ciudadano		
Nombre	Sexo	Email

Tabla 2.1. Registro de ejemplo.

Si *Nombre* tiene el valor “*Jsé*”, cuando “*José*” es el valor correcto de acuerdo a un diccionario de nombres Españoles, este es un caso de baja exactitud. Por otro lado, si se considera que *Email* tiene el valor *null*, puede tener diferentes significados: que el ciudadano no tenga dirección de email y por lo tanto este campo no aplica para el caso, pero también puede ser que el ciudadano tenga un *Email* que no fue almacenado en el registro. Este último es un ejemplo de baja completitud. Un ejemplo de problema de consistencia se podría dar si los campos *Nombre* y *Sexo* tienen como valores “*José*” y “*Femenino*” respectivamente, este puede ser un caso de baja consistencia [9].

Factor de calidad

Un factor representa un aspecto particular de una dimensión de calidad. En este sentido, una dimensión puede ser vista como un agrupamiento de factores de calidad que tienen el mismo propósito. Por ejemplo, para la dimensión *exactitud*, algunos factores pueden ser los siguientes:

- **Correctitud semántica:** los datos representan estados del mundo real.
- **Correctitud sintáctica:** los datos no contienen errores de sintaxis.
- **Precisión:** los datos tienen el nivel de detalle suficiente.

Métrica de calidad

Una métrica es un instrumento que define la forma de medir un factor de calidad. Un mismo factor de calidad puede medirse con diferentes métricas. Para definir una métrica, se deben especificar tres aspectos: la semántica, las unidades de medición y la granularidad de la medida.

La mayoría de las métricas de calidad de datos son generadas para una situación en concreto, de forma de resolver problemas específicos. La necesidad de definir métricas dependientes de cada contexto particular es inevitable dado que la definición de calidad de datos es sensible al contexto [10]. Las métricas son el instrumento que permiten obtener valores de calidad concretos para las dimensiones de calidad de datos que son de interés [11].

Método de medición

Un método es un proceso que implementa una métrica y que se encarga de tomar las medidas necesarias para la misma. A su vez, una métrica puede ser medida por diferentes métodos.

2.2 Enterprise Service Bus

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 diversas a través de empresas, con integridad transaccional [12].

2.2.1 Introducción

Enterprise Service Bus (ESB) es una solución que surge para alejarse de los problemas que las plataformas EAI (siglas en inglés de Integración de Aplicaciones Empresariales) tenían en un principio. Estas lograban la integración no invasiva entre aplicaciones pero presentaban problemas de escalabilidad y confiabilidad. Además, debido a la falta de normas claras para las arquitecturas EAI, y que la mayoría de las soluciones propietarias eran incompatibles entre ellas, se generaba una gran dependencia con el proveedor.

Una de las características principales y fundamentales de los ESB es que están basados en estándares, combinando distintos componentes y disminuyendo la carga de funcionalidad de cada uno, mediante la distribución de algunas tareas.

2.2.2 Características de un ESB

El uso de un ESB favorece el bajo acoplamiento entre aplicaciones, quitando la lógica de integración y comunicación de estas, y dividiéndola en módulos reutilizables dentro del mismo. En general, las características de los ESBs dan soporte a las necesidades de las arquitecturas SOA, ya que cuentan con capacidades de conectividad, adaptación, ruteo y transformación de mensajes, flujos de mediación, mensajería asíncrona y capacidad de monitoreo para los mensajes que se envían a través de él.

A continuación se listan algunas características de los ESB típicos, que les permiten integrar diversas aplicaciones, a través de diferentes protocolos y con distintos tipos de mensajes.

- **Conectividad:** los ESB típicamente incluyen varios adaptadores que les permiten interactuar mediante múltiples protocolos de comunicación y de transporte.
- **Transformación de mensajes:** las capacidades de transformación de mensajes les permiten comunicar aplicaciones que utilizan distintos modelos o formatos de datos.
- **Ruteo inteligente:** las capacidades de ruteo inteligente les permiten determinar dinámicamente el destino de un mensaje.
- **Flujos de mediación:** en un ESB se pueden definir secuencias de operaciones de mediación para aplicar sobre un mensaje.

En la Figura 2.3 puede observarse un ejemplo de la capacidad de los ESB para integrar distintas aplicaciones y servicios que utilizan tecnologías, formatos de datos y protocolos de comunicación distintos a través de servicios.

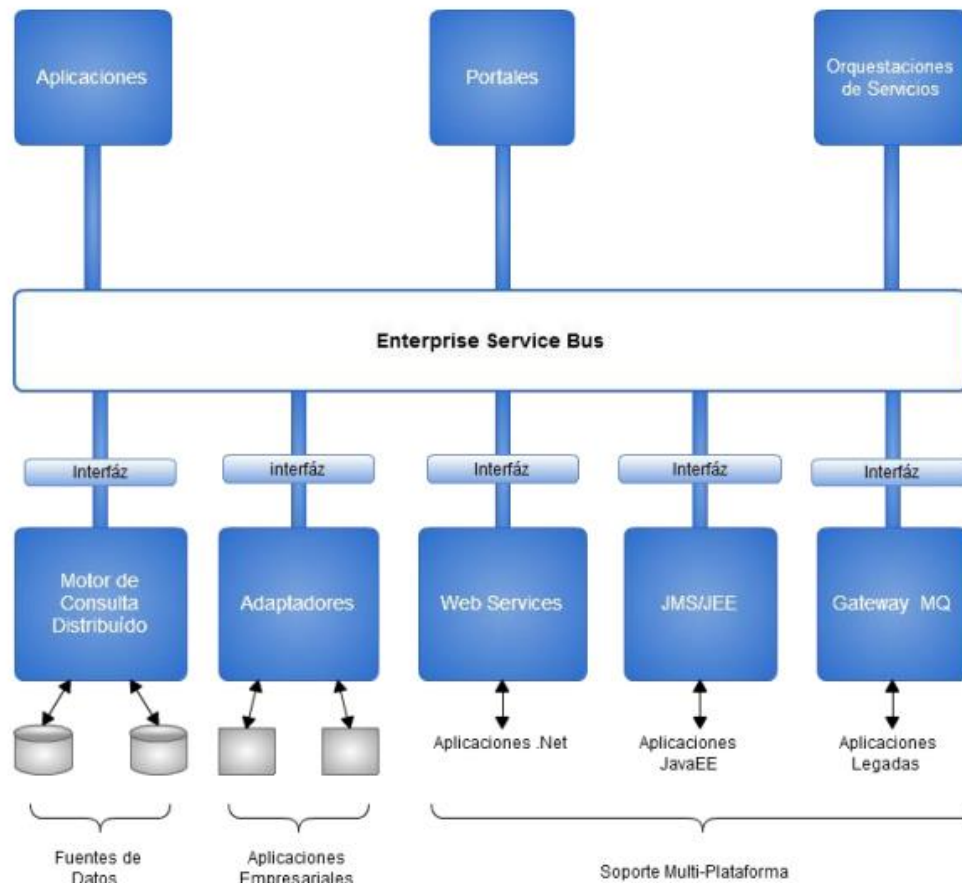


Figura 2.3. Capacidades de los ESB [13].

2.2.3 Patrones ESB

Existen una serie de patrones típicamente ofrecidos por los ESB, y utilizados para resolver problemas que comúnmente surgen en soluciones sobre estas plataformas. A continuación se mencionan los patrones ESB que son utilizados a lo largo de la solución.

Gateway

El Gateway es un patrón utilizado para centralizar el acceso a un sistema, ofreciendo funciones de borde como seguridad, autenticación y auditoría, implementándose por única vez y aplicándose a todos los mensajes que pasan por el ESB. Es utilizado para aplicar un conjunto de operaciones de mediación comunes a todos los mensajes entrantes o salientes. La Figura 2.4 presenta la idea general de este patrón donde un ESB centraliza el acceso a distintos Web Services mediante un Gateway.

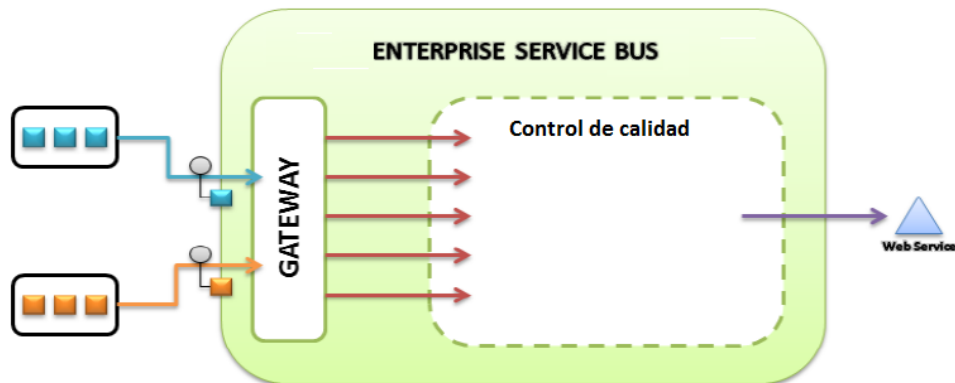


Figura 2.4. Ejemplo de uso del patrón Gateway.

Dynamic Router

El ruteo dinámico permite enviar un mensaje consecutivamente a través de una serie de destinos posibles, que no necesariamente se conocen en tiempo de diseño y pueden cambiar para cada mensaje. En la Figura 2.5 se puede observar el funcionamiento de un componente con ruteo dinámico, en el que el mensaje llega y según el estado de ciertos valores en una base de datos de ruteo, se determina el componente al cual será enviado. Luego se actualizan los valores de ruteo de la base, y el mensaje vuelve a ser procesado con estos.

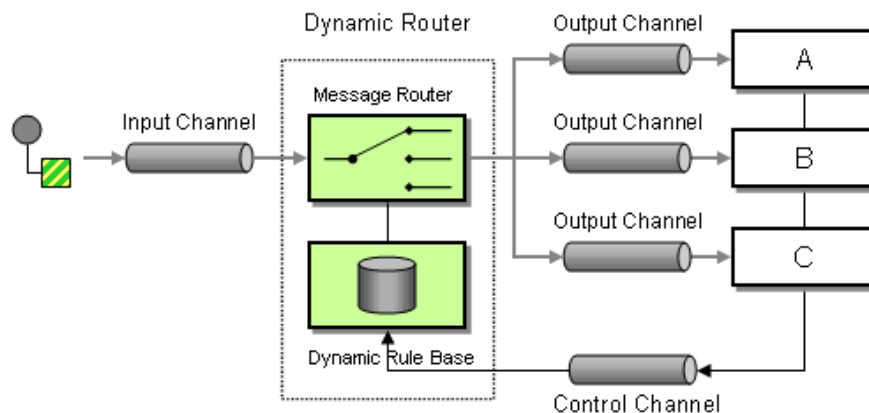


Figura 2.5. Ruteo Dinámico [14].

Content Based Router

El patrón de ruteo basado en contenido es utilizado para redirigir un mensaje a un componente u otro en función de los datos contenidos en el mismo. A su vez el *router* puede calcular el destino basado en varios criterios como: un conjunto de reglas configurables, la

existencia de un determinado campo o un valor específico para un campo dado. La Figura 2.6 presenta la idea general del ruteo basado en contenido, donde un pedido es enviado al ESB, y según su contenido es enviado al destinatario correspondiente.

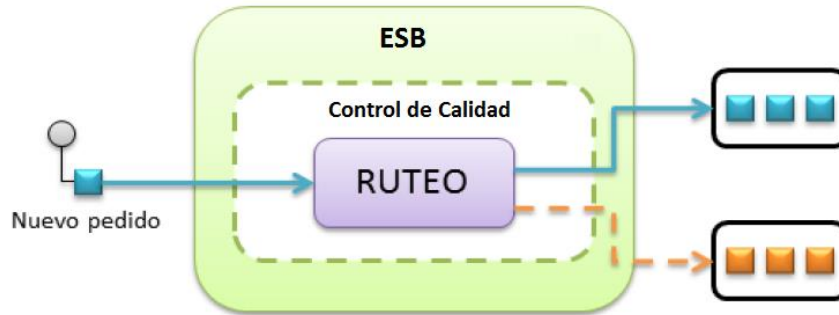


Figura 2.6. Ruteo basado en contenido.

2.3 Web Services

Los Web Services son una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre dos o más sistemas en una red. Se caracterizan por facilitar la interoperabilidad entre aplicaciones (que se consigue mediante la adopción de estándares abiertos). Pueden ser combinados para lograr operaciones complejas, por lo que programas que prestan servicios simples, pueden interactuar entre sí y ofrecer servicios más sofisticados con mayor valor agregado [15].

Un Web Service se identifica por una URI¹ especificada a través de XML Namespace², 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 [16].

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 [16]. En las siguientes sub secciones se brindan más detalles de los mismos.

¹ Un identificador de recursos uniforme o URI (del inglés Uniform Resource Identifier) es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

² http://www.w3schools.com/xml/xml_namespaces.asp

2.3.1. Extensible Markup Language (XML)

XML es un Lenguaje de Etiquetado Extensible muy simple, pero estricto, que juega un papel fundamental en el intercambio de una gran variedad de datos. Es un lenguaje similar a HTML³ pero su función principal es describir datos y no mostrarlos como es el caso de HTML. [17].

XML es un conjunto de reglas para diseñar formatos de texto que permitan estructurar los datos, almacenar e intercambiar información. Facilita a la computadora la tarea de generar datos, leerlos, y asegurar que su estructura no es ambigua. Utilizándolo se evitan las fallas comunes en diseño de lenguajes ya que es extensible, no depende de las plataformas, y soporta internacionalización y localización.

En la Figura 2.7 se muestra un ejemplo de un XML, donde se describe un objeto *ficha*, con cinco atributos: *nombre*, *apellido*, *direccion*, *nacionalidad* y *sexo*.

```
<?xml version=" 1.0 " encoding=" UTF-8 " ?>
<ficha>
  <nombre> Gabriel </nombre>
  <apellido> Molina </apellido>
  <direccion> Soriano 1570 apto 101 </direccion>
  <nacionalidad> Uruguayo </nacionalidad>
  <sexo> M </sexo>
</ficha>
```

Figura 2.7. Ejemplo de XML.

Las tecnologías XML son un conjunto de módulos que ofrecen servicios útiles a las demandas más frecuentes por parte de los usuarios, como por ejemplo XPath⁴, el cual es un lenguaje para acceder a partes de un documento XML.

2.3.2. Web Services Description Language (WSDL)

Es el lenguaje propuesto por el World Wide Web Consortium⁵ (W3C) para la descripción de servicios web y permite describir la interfaz de los mismos en un formato XML. Una de sus ventajas es que permite separar la descripción abstracta de la descripción concreta. Es decir, mediante las etiquetas *types*, *message* y *portType*, se describen los tipos de datos válidos, los mensajes de Entrada y Salida y las operaciones respectivamente. Esta información no depende de la tecnología utilizada. Luego se agrega la información concreta de la implementación, donde,

³ <http://w3.org/MarkUp/>

⁴ <http://www.w3.org/TR/xpath/>

⁵ Comunidad internacional que trabaja para desarrollar estándares web.

mediante las etiquetas *bindings* y *service*, se describe la posible tecnología de transporte (SOAP, HTTP, etc) y el Puerto y Servicio para acceder al servicio, dependiendo de la tecnología utilizada [18].

Para consumir un Web Service se realiza el flujo que se muestra en la Figura 2.8:

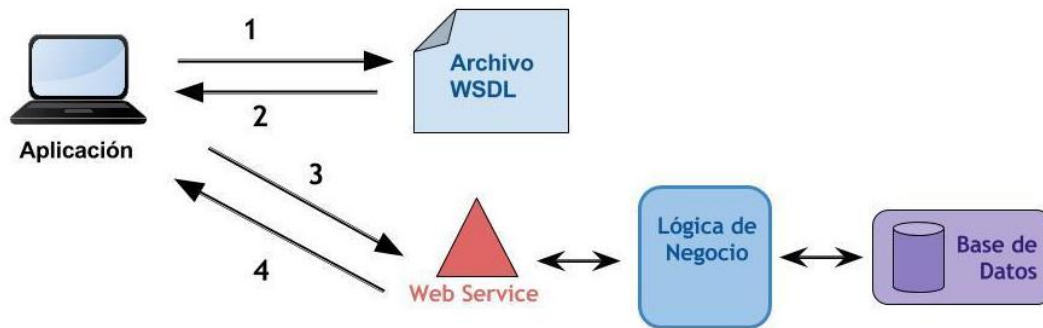


Figura 2.8. Utilización de WSDL para consumir un Web Service.

1. El cliente solicita la definición del servicio especificado en el archivo WSDL.
2. El servidor que contiene el archivo WSDL lo devuelve. Este archivo indica los métodos y propiedades del servicio que están disponibles.
3. El cliente hace la petición en el formato que espera el servicio según las especificaciones del fichero WSDL en el que se dice qué parámetros acepta y de qué tipo.
4. El servicio entrega el resultado de la consulta.

2.3.3 Simple Object Access Protocol (SOAP)

Es un protocolo ligero utilizado para el intercambio de información en un entorno distribuido y descentralizado. Este protocolo está basado en XML y consta de tres partes: un *envelope* que define un marco para describir lo que está en el mensaje y cómo procesarlo, un conjunto de reglas de codificación para expresar instancias de datos definidos por la aplicación, y una convención para representar las llamadas a procedimientos remotos y sus respuestas. SOAP puede ser utilizado en combinación con una variedad de otros protocolos. También define un mecanismo, denominado *SOAP faults*, para indicar y comunicar problemas que ocurran al procesar el mensaje.

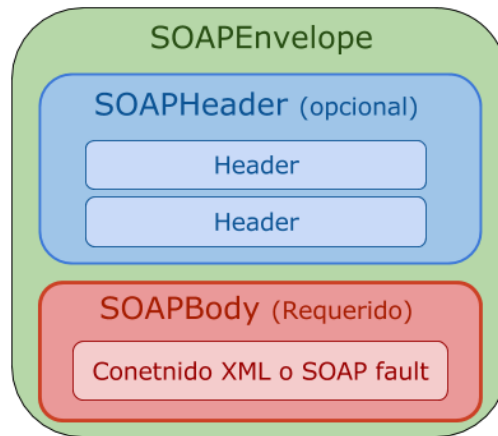


Figura 2.9. Estructura de un mensaje SOAP.

En la Figura 2.9, se representa gráficamente la estructura de un mensaje SOAP. El *Envelope*, está compuesto por dos secciones: *Header*, la cual es opcional y otra denominada *Body*, que es obligatoria. A continuación se detalla cada una de ellas.

SOAP Envelope

Esta estructura sintáctica define un documento XML como un mensaje SOAP, la cual debe estar siempre presente y ser la primera sección del mensaje.

SOAP Header

Es un elemento opcional, para agregar información adicional a la que se encuentra en el elemento *Body* y que en caso de estar presente debe ser el primer hijo del elemento *Envelope*. Es extensible y puede contener varios encabezados que se utilizan para especificar distintos tipos de información, por ejemplo, detalles específicos de la aplicación como seguridad o contexto transaccional.

SOAP Body

Es la sección del cuerpo de un mensaje SOAP que aloja la información enviada. Debe estar obligatoriamente en el mensaje y ubicarse a continuación del *Header* en caso de que éste exista, o en caso contrario, como primer elemento del *Envelope*. Es el encargado de mantener la información relativa a la invocación o respuesta del servicio. Este contenido se encuentra en formato XML y se ajusta a lo especificado en la descripción del Web Service (WSDL) que se invoca.

A continuación, en la Figura 2.10, se puede ver un ejemplo resumido de un mensaje SOAP, donde se diferencia claramente cada una de las secciones explicadas anteriormente.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soapenv:Header>
    ...
  </soapenv:Header>
  <soapenv:Body>
    ...
    <soapenv:Fault>
      ...
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>

```

Figura 2.10. Ejemplo resumido de un mensaje SOAP.

2.3.4 Web Services Addressing (WS-Addressing)

WS-Addressing [19] provee mecanismos neutrales, independientes de la capa de transporte, para direccionar mensajes y servicios web. Esta especificación define elementos de XML para referenciar servicios y facilitar el direccionamiento punto a punto de los *endpoints* en los mensajes. Permite al sistema de mensajes dar soporte a la transmisión de mensajes, a través de redes, que incluyen nodos con procesamiento tales como gestores de *endpoints*, *firewalls* y *gateways* de modo independiente a la capa de transporte.

La especificación WS-Addressing define dos conceptos: *Endpoint References* y *Message Addressing Properties*. A continuación se detallan ambos.

Endpoint Reference (EPR)

El *Endpoint Reference* (Identificador de Punto de Acceso) es una colección de propiedades abstractas que identifica el punto donde deben ser dirigidas las peticiones. Algunas de las propiedades que puede contener un EPR son:

- **Address:** una URI que identifica el punto de acceso. Este elemento es obligatorio.
- **ReferenceProperties:** propiedades individuales necesarias para identificar la entidad o recurso transportado. Las provee el creador del punto de acceso al servicio web.
- **ReferenceParameters:** parámetros que facilitan interacciones fruto del camino que recorre el mensaje. Las provee el punto de acceso al servicio web.

Message Addressing Properties

Provee la información que caracteriza el mensaje y que no se puede modificar a lo largo del transporte del mismo. Define el conjunto completo de información de direcciones que se puede incorporar en la cabecera del mensaje SOAP (*SOAPHeader*). Contiene las siguientes propiedades requeridas [20]:

- **To:** indica a dónde se envía el pedido.
- **Action:** indica la acción a realizar por el receptor del mensaje.

Puede contener también los elementos opcionales:

- **ReplyTo:** especifica la dirección de un EPR al que se debe enviar la respuesta a un requerimiento. Provee de asincronismo a un Web Service.
- **FaultTo:** especifica la dirección de un EPR al que se debe invocar cuando ocurra una situación anómala en el Web Service invocado, y no pueda terminar de procesar el pedido.
- **MessageID:** identifica de forma unívoca el mensaje.
- **RelatesTo:** permite correlacionar mensajes.
- **From:** un EPR del punto de acceso del emisor del mensaje.

3 Análisis

En este capítulo se realiza un análisis en profundidad de la problemática planteada. En primer lugar, en la sección 3.1, se describe el marco de trabajo del cual parte el proyecto. Luego, en la sección 3.2 se realiza el análisis de la problemática planteada en el mismo, para luego identificar un conjunto de requerimientos que se describen en la sección 3.3. En la sección 3.4 se detallan trabajos que se relacionan con el presente proyecto y finalmente en la sección 3.5 se comentan algunas conclusiones del capítulo.

3.1 Marco de Trabajo

En la Figura 3.1, se muestra el escenario de partida, donde existen organismos y aplicaciones que interactúan a través de una Plataforma de Integración, por ejemplo una Plataforma de Integración en el área de la Salud. Los organismos exponen sus funcionalidades por medio de un conjunto de servicios, mientras que las aplicaciones clientes acceden a dichos servicios a través del ESB, dirigiéndose a un único punto de entrada. El ESB se encargará de dirigirlos al destino correspondiente, luego de haber realizado todos los procedimientos pertinentes.

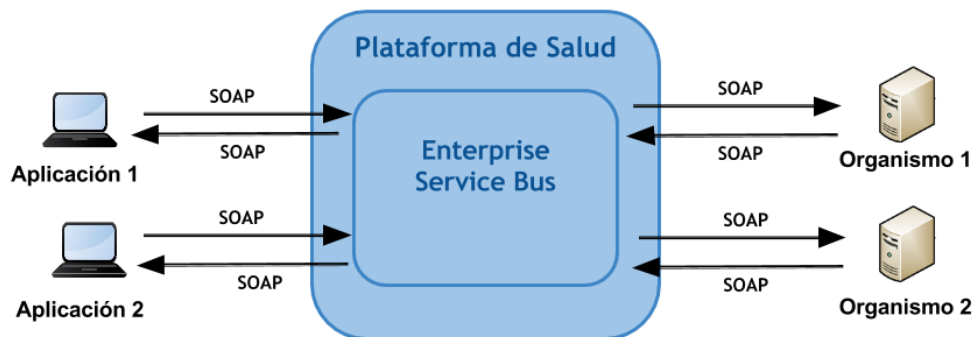


Figura 3.1. Estado de la plataforma sin el componente de calidad de datos.

Los servicios ofrecidos por los proveedores son específicamente Web Services, que se describen por medio de un archivo WSDL y se invocan a través del envío de mensajes SOAP sobre HTTP.

En la Figura 3.2, se presentan algunos de los conceptos identificados en el análisis realizado hasta el momento, relacionados a los organismos y los servicios que ellos ofrecen a través de la plataforma.

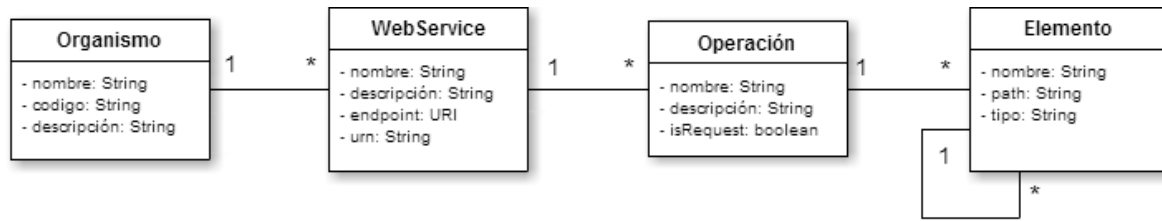


Figura 3.2. Diagrama conceptual parcial (parte 1).

A continuación se describen los conceptos identificados:

- **Organismo:** hace referencia a las distintas instituciones que se encuentran integradas en la plataforma, por ejemplo el Ministerio de Salud Pública. Cada organismo expone un conjunto de Web Services.
- **Web Service:** representa los servicios web publicados en la plataforma. Está vinculado a un Organismo, que es el que publica el servicio. Contiene un conjunto de operaciones y almacena *nombre*, *descripción*, *URN*⁶ (por el cual es conocido para los usuarios de la plataforma) y *endpoint* en el cual está publicado el servicio web.
- **Operación:** hace referencia a la operación de un servicio web publicado en la plataforma, puede estar asociado a varios Elementos. Se diferencia entre operaciones de entrada (*request*) y de salida (*response*).
- **Elemento:** son los parámetros de las Operaciones, tanto de entrada como de salida. Pueden asociarse entre ellos para representar Elementos complejos. Por ejemplo, en la Figura 3.3 se muestran los Elementos: *serie*, *numero*, *organismo* y *contrasenia* de un *request*. En la Figura 3.4 se pueden observar los Elementos de un *response*: *codigoDepartamento*, *codigoEstablecimiento*, *sexo*, *tipoDocumento* y *domicilio*, siendo *domicilio* un Elemento complejo compuesto por los Elementos *calle* y *numero*.

```

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header/>
  <s:Body>
    <ns2:obtenerComprobante xmlns:ns2="http://ws.cde.msp.sofis.com/">
      <serie>C</serie>
      <numero>21</numero>
      <organismo>Organismo1</organismo>
      <contrasenia>password</contrasenia>
    </ns2:obtenerComprobante>
  </s:body>
</s:Envelope>
  
```

Figura 3.3. Ejemplo de un Request HTTP/SOAP [21].

⁶ Nombre de Recurso Uniforme o URN (acrónimo inglés de Uniform Resource Name). Identifica recursos en la web, pero a diferencia de un "Localizador Uniforme de Recursos" (Uniform Resource Locator, URL), no indica exactamente dónde se encuentra ese objeto.



Figura 3.4. Ejemplo de respuesta de un servicio web [21].

3.2 Análisis del Problema

Para cumplir con el objetivo general del proyecto, se debe extender la Plataforma de Integración del Marco de Trabajo (Figura 3.1) con mecanismos que permitan monitorear y garantizar la calidad de los datos que fluyen por la misma. En la Figura 3.5 se puede observar cómo se extendería la Plataforma de Integración para incorporar el componente de Calidad de Datos.

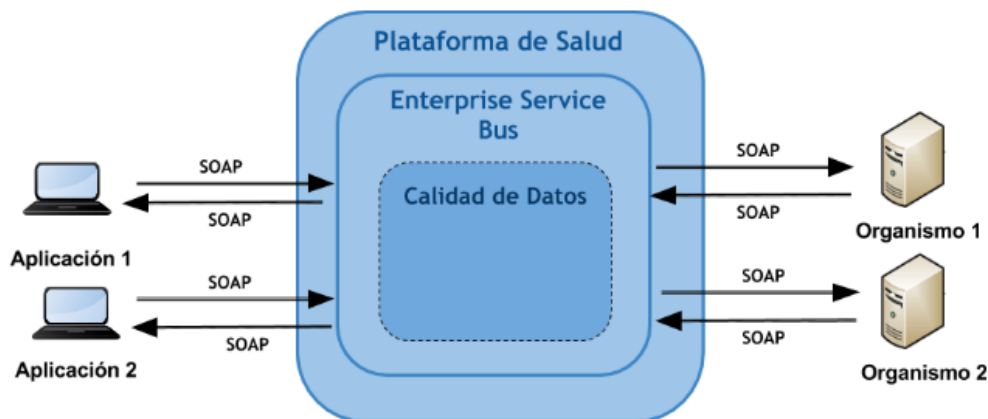


Figura 3.5. Incorporación del componente de Calidad de Datos a la plataforma.

Se considera un contexto donde no se tiene control sobre la infraestructura de los proveedores ni de los consumidores. Por lo tanto, el control de la calidad de los datos solamente se podrá realizar dentro del ESB, basando las acciones y controles del mismo en las capacidades de mediación de un ESB (ruteo, transformaciones, seguridad, etc).

El sistema debe permitir a un usuario administrador gestionar los Modelos de Calidad utilizados, así como las acciones a realizar de acuerdo a los resultados obtenidos por los métodos definidos en dichos Modelos.

Los mensajes de entrada o de salida de los Web Services ofrecidos por las aplicaciones conectadas al ESB serán del tipo SOAP sobre HTTP. Estos servicios deben ser dados de alta por el administrador, y sus operaciones y elementos serán configurados de acuerdo a los Modelos de Calidad que se les desea aplicar.

Se deben implementar mecanismos que permitan crear los Modelos de Calidad en base a la información brindada por el usuario, así como también realizar el control de calidad en tiempo real, utilizando la información de los mensajes intercambiados y capturados.

También debe ser posible mantener un registro de los mensajes procesados, detallando si cumplen con los Modelos de Calidad. Para poder dar de alta los métodos correspondientes a los Modelos de Calidad y las acciones antes mencionadas, se debe proveer una herramienta que facilite la creación y reutilización de estos.

En la Figura 3.6, se profundiza sobre el análisis conceptual relativo al manejo de Modelos de Calidad a aplicar sobre los mensajes interceptados en la plataforma.

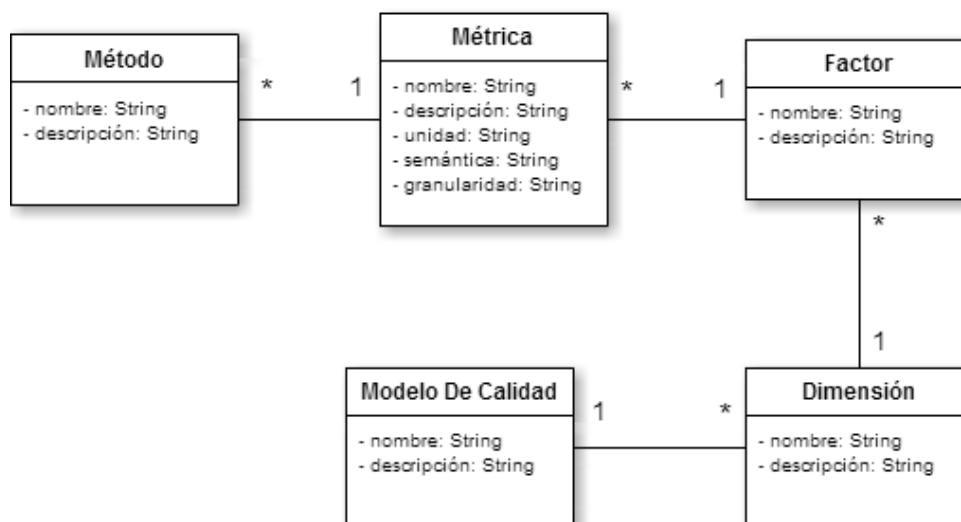


Figura 3.6. Diagrama conceptual parcial (parte 2).

A continuación se detallan los conceptos relativos a Modelos de Calidad, introducidos en la sección 2.1.2 y cómo se interpretan según el presente análisis:

- **Dimensión:** se define con un nombre y descripción. Agrupa un conjunto de factores de calidad con un propósito en común y está asociada a un Modelo de Calidad.
- **Factor:** se define con un nombre y descripción. Un mismo Factor de calidad puede medirse con diferentes métricas.
- **Métrica:** determina la forma en que se mide un Factor de calidad. Se debe definir: la semántica, las unidades de medición y la granularidad de la medida. Una misma Métrica puede ser medida por diferentes métodos.
- **Método:** verifica que un conjunto de elementos cumpla con la Métrica que implementa, retornando *true* en caso que se cumpla, y *false* en caso contrario. Por ejemplo, para una Métrica que verifica si un Elemento tiene valor nulo, el Método para implementarla es un proceso que verifica la existencia del elemento en el mensaje.
- **Modelo de Calidad:** abarca el conjunto de Dimensiones, Factores, Métricas y Métodos que determinan las condiciones que un servicio debe satisfacer. Se puede decir que un mensaje cumple con un modelo de calidad si sus elementos verifican todos los Métodos definidos en dicho Modelo.

3.3 Requerimientos Identificados

El sistema debe contar con dos tipos de usuarios: por un lado, se encuentran los usuarios administradores, quienes son los encargados de gestionar el sistema en su totalidad (administrar modelos de calidad y organismos vinculados al sistema, registrar servicios y configurar Métodos y acciones). Por otro lado, los usuarios pertenecientes a los distintos organismos que integren la plataforma, acceden a visualizar los registros y estadísticas que se obtengan de aplicar los controles de calidad a los servicios web publicados por sus propios organismos, además de solicitar el ingreso o la baja de un servicio web a la plataforma.

3.3.1 Requerimientos Funcionales

A partir del análisis realizado, se identificó un conjunto de requerimientos funcionales, que se detalla a continuación.

En primer lugar, en la Tabla 3.1 se especifican los requerimientos funcionales identificados para el usuario Administrador.

Usuario Administrador	
REF-1	El sistema debe permitir dar de alta, baja y modificar (ABM) Organismos.
REF-2	Se debe permitir ABM de usuarios pertenecientes a organismos.
REF-3	Se debe dar la posibilidad de registrar en el sistema los servicios web, que publican los diferentes organismos, ingresando su correspondiente descripción WSDL.
REF-4	Se debe brindar la posibilidad de dar de alta Modelos de Calidad, ingresando nombre y descripción de las Dimensiones, los Factores que las componen y las Métricas correspondientes.
REF-5	Se debe brindar una herramienta para poder especificar de forma flexible los Métodos que implementan las Métricas.
REF-6	El sistema debe permitir la configuración de acciones a realizar, condicionadas a los resultados de la ejecución de un Método, tales como cancelar el envío del mensaje, notificar por diferentes medios a quien se desee, intentar asegurar la calidad de los datos y registros para su posterior estudio y creación de <i>dashboards</i> .
REF-7	Se debe ofrecer la posibilidad de visualizar los registros generados sobre las validaciones de los pedidos que se efectúan en el ESB.

Tabla 3.1. Requerimientos Funcionales para Usuario Administrador

En la Tabla 3.2 se detallan los requerimientos funcionales para los usuarios de organismos:

Usuarios de Organismos	
REF-8	Visualización de los registros y <i>dashboards</i> , generados sobre las validaciones realizadas por los Modelos de Calidad que abarcan Operaciones de servicios web correspondientes al Organismo al que pertenece el usuario.
REF-9	Recepción y visualización de notificaciones, emitidas por las acciones configuradas por el administrador, donde recibirán información de relevancia con respecto a los servicios publicados por su Organismo.
REF-10	Solicitud de publicación de un servicio, donde el usuario ingresa un servicio web publicado por su Organismo, con el fin de que un usuario administrador lo registre en el sistema.

Tabla 3.2. Requerimientos Funcionales para Usuario de Organismo

Finalmente, en la Tabla 3.3 se especifican los requerimientos relacionados al intercambio de mensajes:

Intercambio de Mensajes	
REF-11	El sistema debe interceptar todos los mensajes que llegan a la plataforma de integración intercambiados entre los Organismos.
REF-12	Evaluar los Métodos correspondientes a los mensajes interceptados, definidos por el usuario administrador.
REF-13	Ejecutar las acciones pre-configuradas por el administrador en los casos que corresponda.
REF-14	Registrar toda la información que se considere relevante, sobre el flujo de mensajes, tales como: mensajes interceptados, validaciones realizadas, Métodos de control de calidad ejecutados, acciones realizadas por la plataforma y notificaciones.
REF-15	Realizar el ruteo de los mensajes interceptados a sus destinos correspondientes.
REF-16	Realizar un manejo correcto de errores, devolviendo mensajes que especifiquen de la mejor forma lo ocurrido, utilizando un formato estándar.

Tabla 3.3. Requerimientos Funcionales sobre el intercambio de mensajes.

3.2.2 Requerimientos No Funcionales

Se definen en la Tabla 3.4 los requerimientos no funcionales que la solución debe contemplar.

Requerimientos No Funcionales	
RENF-1	La solución se debe implementar en lenguaje de programación Java utilizando JBoss EAP como servidor de aplicaciones.
RENF-2	La solución debe ser independiente a nivel de diseño del producto ESB utilizado en la plataforma.
RENF-3	Se debe permitir agregar servicios nuevos en tiempo de ejecución, sin necesidad de reiniciar la aplicación.
RENF-4	Se debe priorizar la disponibilidad de la plataforma, permitiendo realizar la mayor parte de configuraciones en tiempo de ejecución.
RENF-5	La solución debe ser lo más extensible y flexible posible, pudiendo ser adaptada para diferentes ambientes y dominios.

Tabla 3.4. Requerimientos No Funcionales.

3.4 Trabajos relacionados

Se han encontrado diversos trabajos que por alguno de los temas que tratan, o de los conceptos que involucran, se relacionan con el trabajo desarrollado en el presente proyecto. A continuación se describen algunos de ellos y el vínculo encontrado con este.

3.4.1 Protección de Datos Personales en Plataformas de Integración

Uno de los trabajos relacionados que se tuvo en cuenta es el de “Protección de Datos Personales en Plataformas de Integración”, llevado a cabo como proyecto de grado de la Facultad de Ingeniería de la Universidad de la República, Uruguay, entre los años 2014 y 2015.

El mismo tiene similitudes en cuanto a que se aplica sobre una plataforma ESB de intercambio de datos entre organizaciones. Para esos datos, se deben gestionar normativas asociadas a la protección de datos personales, así como monitorear y asegurar el cumplimiento de las mismas en las interacciones que se dan entre los organismos [22]. En este sentido existen semejanzas a nivel procedimental: se interceptan y examinan todos los mensajes que pasan por la plataforma, se validan en base a configuraciones realizadas y según esas validaciones, se pueden transformar y tomar acciones predefinidas en el sistema.

Por estos motivos se tuvo como referencia en decisiones tales como: solicitar que los mensajes enviados al sistema deben contener un cabezal WS-Addressing indicando emisor, destinatario, Web Service y Método solicitado, así como también, el hecho de tener un cabezal propio con datos para uso interno del sistema.

3.4.2 Plataforma ESB Adaptativa para Sistemas Basados en Servicios

Tesis de Maestría en Informática, presentada en el año 2011 en la Facultad de Ingeniería de la Universidad de la República. En dicho trabajo se propone una Plataforma ESB Adaptativa que basándose en las capacidades de mediación de los ESBs, permita responder a las necesidades de adaptación en una SOA de forma dinámica, automática y en tiempo de ejecución [23]. Se tuvo en cuenta a la hora de tomar decisiones a lo largo del proyecto en cuanto a qué características y patrones típicos brindados por ESB podrían utilizarse, por el hecho de que su solución está diseñada en base a las capacidades de mediación típicas de estas plataformas.

3.4.3 Data Quality-Based Resource Management in Enterprise Service Bus

Esta tesis tiene como punto de partida la necesidad de nuevos mecanismos para la selección de recursos en un ESB, utilizando como principal herramienta la calidad de los datos, y proponiendo un marco para su manejo [3].

A pesar de definir un *framework* flexible para el control de calidad de datos, el mismo brinda una lista acotada de dimensiones sin dar la posibilidad de agregarlas a demanda. Por otro lado, la solución propuesta por dicha tesis está fuertemente enfocada en considerar la calidad

de los datos para la selección de las fuentes de datos, sin contemplar acciones en respuesta a los resultados de las mediciones.

Igualmente, fue posible recabar algunas ideas de este proyecto, como por ejemplo el diseño de la forma en que se dan de alta los métodos con sus reglas.

3.4.4 Implementación de una Plataforma ESB Adaptativa

Proyecto de grado de la Facultad de Ingeniería [24], de la Universidad de la República, Uruguay, que lleva a cabo una implementación de la solución propuesta en el proyecto “Plataforma ESB Adaptativa para Sistemas Basados en Servicios” [23]. En este trabajo se realiza una interesante comparación de tecnologías ESB, en base a tres tipos de criterios: estratégicos, funcionales y técnicos. Se utilizó como referencia para comparar con otras tecnologías más recientes y seleccionar el producto para la implementación del prototipo.

3.5 Conclusiones

En este capítulo se describió el marco de trabajo en el cual está contenido el proyecto, identificándose los conceptos existentes en el escenario de partida. Además se realizó un análisis de la problemática planteada, introduciéndose nuevos conceptos necesarios para la resolución del problema, evaluando las características y los requerimientos necesarios que la solución debe contemplar.

En base al intercambio con las tutoras, que también oficiaron de clientes, se fue estudiando el problema y las particularidades del mismo, resultando en la identificación de los requerimientos. De este también surgió el planteo de cómo encarar los conceptos relativos a los Modelos de Calidad y permitir a los usuarios (administradores) de la plataforma gestionar los Métodos y acciones, haciendo hincapié en la flexibilidad y extensibilidad de la misma.

Además, se comentaron los trabajos relacionados con este proyecto que fueron tenidos en cuenta en el desarrollo del mismo. Estos fueron de importancia en cuanto a que en algún momento encaran obstáculos similares a los que surgen del análisis de la problemática, sirviendo como referencia en distintos puntos de la solución a construir.

4 Solución propuesta

En este capítulo se describen las principales características de la plataforma propuesta. En la sección 4.1 se presenta la descripción general de la solución, motivada por los requerimientos definidos en el capítulo anterior. En la sección 4.2 se detalla el modelo conceptual de la solución y en la sección 4.3 se describen sus principales características. En la sección 4.4 se especifica el esquema general de su arquitectura. Por último, en la sección 4.5 se presentan características de flexibilidad y extensibilidad de la solución.

4.1 Descripción General

Se propone extender las funcionalidades que ofrece una plataforma típica de Enterprise Service Bus, de forma de agregar monitoreo, control y aseguramiento de calidad de datos a los mensajes que fluyen por este tipo de plataformas. Dichos mensajes surgen entre las aplicaciones cliente y los organismos servidores, tanto de solicitudes como de respuestas de servicios. En la Figura 4.1 se puede observar cómo se incorpora la solución propuesta a la Plataforma de Integración del Marco de Trabajo (Figura 3.1).

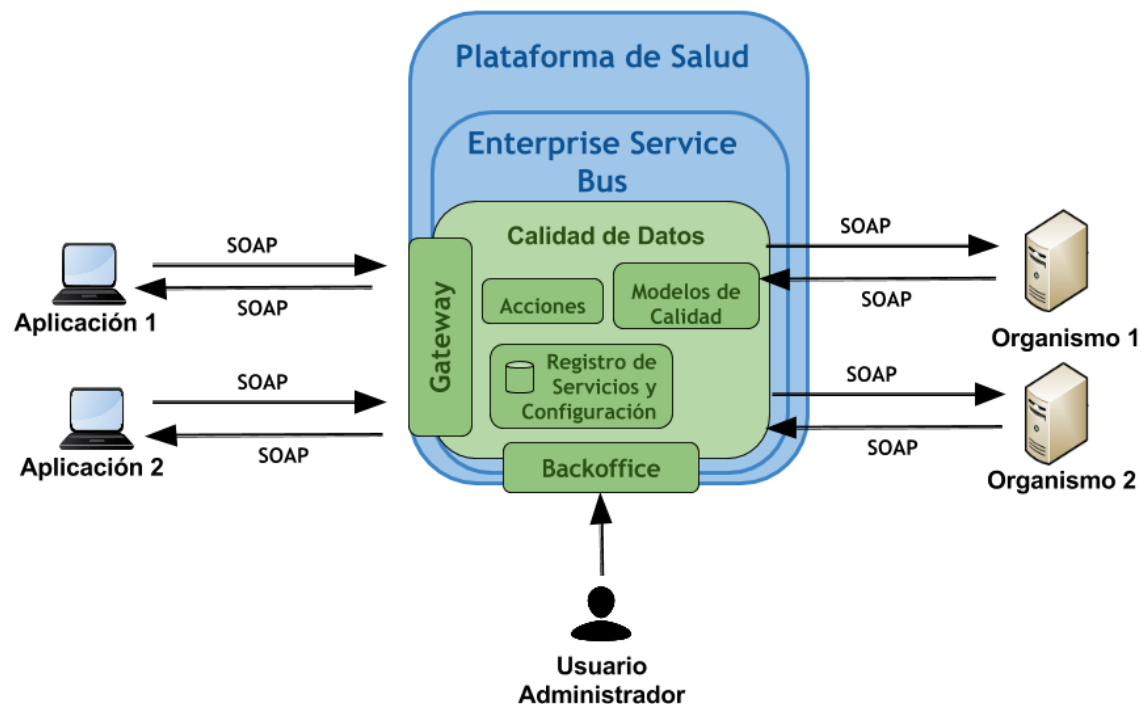


Figura 4.1. Estado de la plataforma con el componente de calidad de datos.

Gateway

Es el componente encargado de centralizar el acceso de los clientes a los servicios brindados por la plataforma, así como también enviarles las respuestas.

Backoffice

Es un componente web para gestionar los Organismos y Web Services que estos publican, así como también los Modelos de Calidad que la plataforma puede abarcar para cada uno de los dominios manejados. Por ejemplo, en la Plataforma de Gobierno Electrónico hay servicios de salud, de identificación civil, de medio ambiente, entre otros. Administrar distintos Modelos de Calidad para cada organización permite a la plataforma adecuarse mejor a las necesidades de cada una. Además, este componente permite configurar las acciones a realizar para cada uno de los Métodos.

Registro de Servicios y Configuración

Los servicios del ESB están almacenados de forma que puedan ser identificados por los usuarios y administradores para ser invocados desde la plataforma. Con esto se logra independizar la localización física de los organismos servidores. Además se almacenan datos de configuración necesarios para el funcionamiento de la plataforma y del *Backoffice*, como pueden ser usuarios, organismos, modelos de calidad, entre otros.

Modelos de Calidad

Los Modelos de Calidad contienen Métodos de evaluación, encargados de resolver si cierta Métrica es cumplida por los elementos que tienen asociados. De esta forma, un Método se encarga de evaluar de forma positiva o negativa cierto conjunto de Elementos de un mensaje según su propia lógica. Estos Elementos pertenecen a las Operaciones de *request* y *response* de los servicios expuestos por la plataforma, y están asociados a uno o más Métodos. De este modo, cuando se intercepta un mensaje por el componente Gateway se chequea a qué Operación corresponde y en base a eso se buscan todos los Elementos definidos para esa Operación.

Haciendo uso de los patrones Ruteo Basado en Contenido y Ruteo Dinámico (detallados en la sección 2.2.3) se aplican los Métodos asociados a cada uno de los Elementos obtenidos para evaluar su calidad.

Acciones

Una vez aplicados los Métodos definidos para esos Elementos, si no se cumple con las Métricas requeridas, se ejecutan las acciones determinadas para esos Métodos. Estas acciones van desde notificar a un administrador (vía SMS, correo electrónico, etc), interrumpir el mensaje respondiendo al cliente con el error correspondiente, o ejecutar mecanismos para tratar de asegurar la calidad, como por ejemplo realizar transformaciones sobre el mensaje.

La Figura 4.2 presenta un ejemplo en el cual se describe el funcionamiento general de la plataforma. En este ejemplo, el organismo Dirección General de Registro de Estado Civil (DGREC) invoca la operación *obtenerCertificado* del Ministerio de Salud Pública (MSP). Existe además un Modelo de Calidad donde se especifica un Método que valida que el formato del elemento *fechaDefuncion*, parámetro de entrada de la operación mencionada, tenga el formato aaaa-mm-dd. Como en este caso el mensaje no cumple con este control, se ejecutan las acciones asociadas al Método: se envía un email notificando el incumplimiento de dicho control de calidad y se realiza una transformación XSL al mensaje, para convertir la fecha al formato esperado.

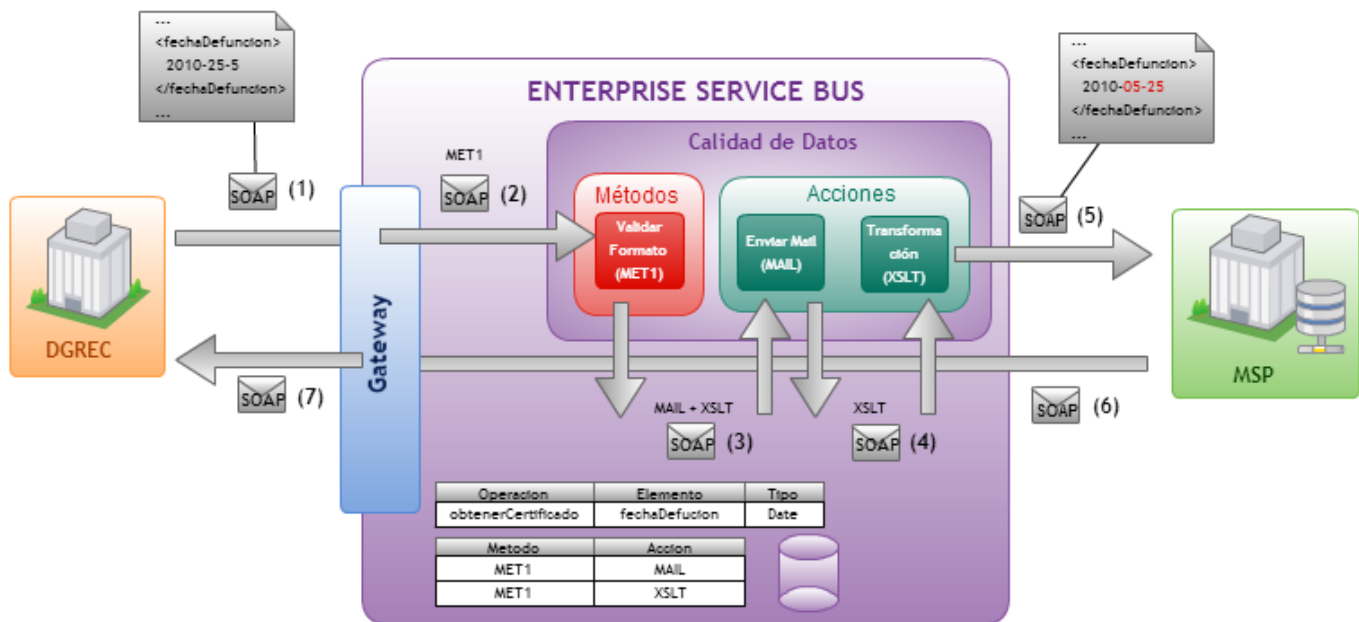


Figura 4.2. Ejemplo control de calidad de datos en ESB

El organismo DGREC envía al Gateway (1) el mensaje que desea enviar al servicio que publica el MSP. En el Gateway se realizan las validaciones correspondientes y el registro del mensaje de entrada. En este caso, según la operación y sus elementos se detecta que debe evaluarse el Método MET1, por lo que redirige el mensaje hacia el Método correspondiente (2). Aquí se ejecuta el Método y, al no cumplirse, el mensaje es redirigido hacia la primera de las acciones asociadas al Método, que en este caso corresponde al envío de una notificación por mail (3). Luego se envía hacia el componente encargado de ejecutar la siguiente acción, donde se ejecuta una transformación XSL (4). El mensaje final es enviado hacia el servicio que el cliente original desea consumir (5). Una vez que el MSP termina de procesar el pedido, retorna la respuesta al ESB (6), y al no existir Métodos configurados para los elementos del mensaje de respuesta, es enviada hacia la DGREC por medio del Gateway (7).

4.2 Modelo Conceptual

En esta sección se describe el modelo conceptual que maneja la plataforma en base a lo descrito en la sección anterior.

La Figura 4.3 presenta el modelo conceptual de la solución propuesta. En él se puede observar que los organismos publican Web Services, los cuales pueden tener varias operaciones. Cada operación contiene un conjunto de elementos de entrada y otro de salida.

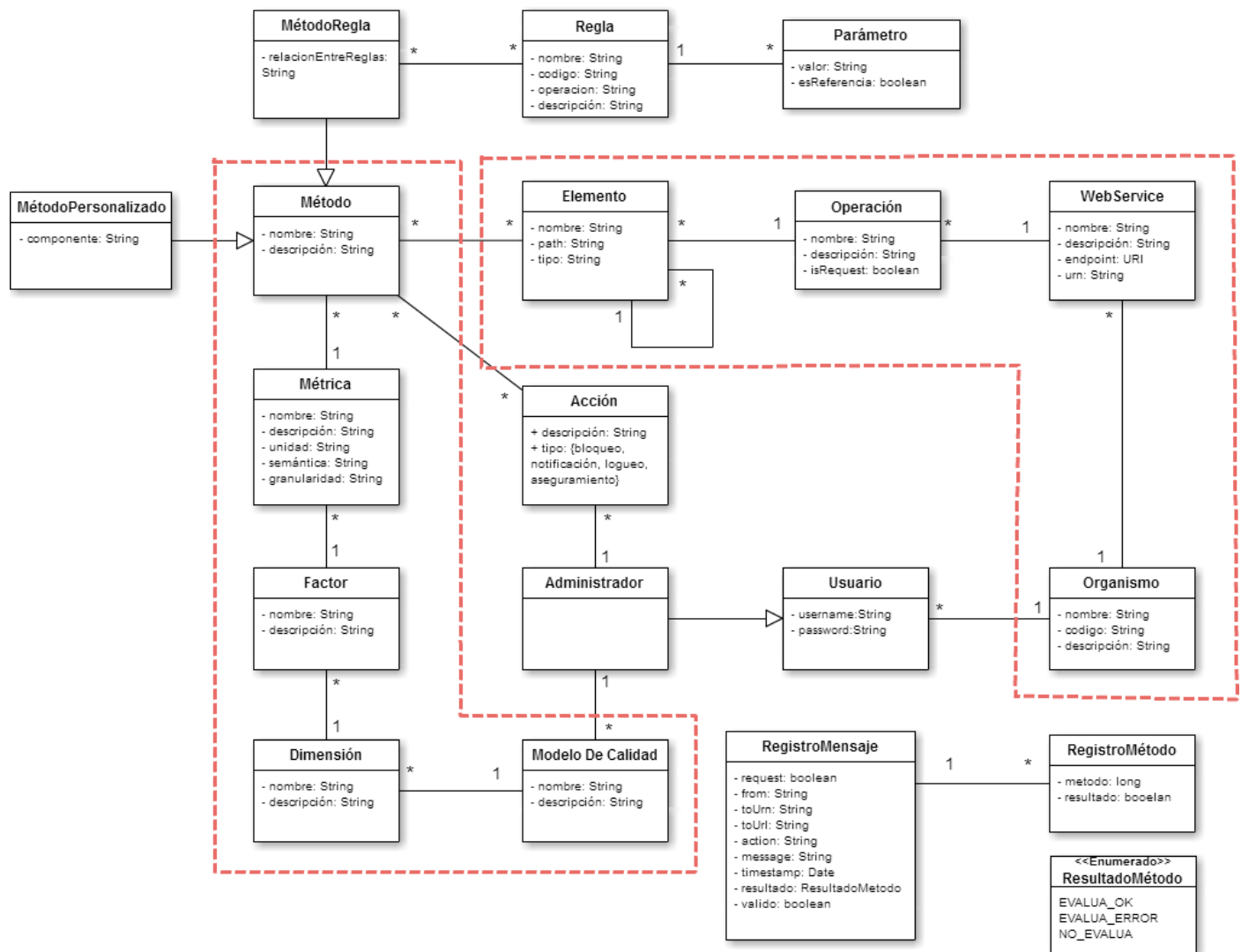


Figura 4.3. Diagrama conceptual de la solución propuesta, se indican los conceptos relativos al análisis de la problemática.

Las Métricas son definidas dentro del contexto de un Modelo de Calidad, los cuales son dados de alta por un usuario administrador, y pueden tener varias implementaciones (Métodos). Los Métodos pueden ser de dos tipos: basados en un sistema de reglas, o personalizados. Para el primer caso, se define un conjunto de Reglas y una relación lógica entre ellas: el resultado de esa relación es el que determina el resultado final de la evaluación del Método. Cada regla puede recibir un conjunto de parámetros, que pueden ser fijos o referencias a elementos del mensaje que se procesa. Para el segundo tipo de Métodos, el usuario (avanzado) implementa un componente del sistema para su resolución y genera un Método Personalizado con la referencia a ese componente.

Por último, para cada una de las relaciones entre conjunto de elementos y Método, se define una serie de Acciones, las cuales serán ejecutadas en caso de que no se cumpla con el Método especificado.

A continuación se describen los conceptos introducidos en la solución propuesta:

- **Acción:** es un conjunto de tareas a ser aplicadas sobre los mensajes interceptados (por ejemplo, transformaciones). Son preconfiguradas por un administrador al momento de dar de alta un Modelo de Calidad.
- **Regla:** cada una de ellas representa una condición que se aplica sobre el mensaje y tiene como resultado *true* o *false* indicando si se validó o no. Se definen utilizando operadores de comparación, funciones de conjuntos, operaciones lógicas, expresiones regulares, entre otros, aplicados entre elementos del mensaje o parámetros definidos por el usuario.
- **Parámetro:** almacena el tipo y valor ingresado por el usuario para ser utilizados en la definición de una Regla.
- **MétodoRegla:** representa un tipo de implementación de Métodos, donde cada uno de estos queda definido por un conjunto de Reglas y una relación lógica que las vincula para determinar el resultado final del Método.
- **MétodoPersonalizado:** representa un tipo de implementación de Métodos, donde un usuario con conocimientos de programación implementa una solución siguiendo una interfaz definida, que da como resultado *true* o *false*, indicando si el Método se cumple o no.
- **RegistroMensaje:** es una entidad utilizada para el registro de la actividad de los mensajes a través de la plataforma. Tiene como objetivo recolectar la información necesaria para generar reportes y Métricas. Se explica con mayor detalle en la sección 4.2.6.
- **RegistroMétodo:** al igual que RegistroMensaje, es una entidad utilizada para la recolección de información para su posterior utilización en reportes y Métricas. En este caso, se enfoca en registrar los Métodos y el resultado de su ejecución.

A continuación se presenta un ejemplo para facilitar la comprensión del modelo conceptual manejado por la solución.

Un administrador del sistema registra un servicio que contiene una operación llamada *obtenerCertificado*, la cual tiene gran cantidad de elementos de entrada y de salida. Entre los de salida se encuentra *fechaDefuncion*, de tipo *Date*. El usuario ingresa un Modelo de Calidad, asociando dicho elemento a un Método “Validar Formato” que implementa la Métrica llamada “Formato Fecha”, encargada de hacer cumplir el Factor “Correctitud sintáctica”. El mismo va a evaluar que el campo *fechaDefuncion* tenga el formato yyyy-MM-dd.

Para el Método ingresado, el administrador determina un conjunto de Acciones, las cuales serán ejecutadas en caso de que no se cumpla con el Método definido. Por ejemplo, para el Método “Validar Formato”, se han definido dos Acciones: una notificación vía correo electrónico a un destinatario encargado de monitorear que las llamadas a los servicios se estén realizando con datos válidos, y una Acción de transformación, que intenta convertir, por ejemplo, fechas del estilo 2013-5-24 a 2013-05-24.

Suponiendo que se hace una invocación a la operación *obtenerCertificado* mencionada anteriormente, y que esta devuelve 2013-5-24 como *fechaDefuncion*, la plataforma detectará una falla en el formato de la misma y ejecutará las Acciones configuradas anteriormente: enviará un correo a la dirección de los destinatarios configurados y luego ejecutará la Acción de transformación, convirtiendo el valor de la fecha a 2013-05-24. En cambio, si la *fechaDefunción* devuelta hubiera sido, por ejemplo: 2013-24-24, la transformación para convertir la fecha al formato adecuado no hubiera solucionado el problema de que la fecha es incorrecta. Para ese caso se podría tener otro Factor “Correctitud semántica”, con una Métrica “Fecha válida” que verifique que se está utilizando una combinación año-mes-día válida para el calendario Gregoriano.

4.3 Principales características

En las siguientes secciones se describen los aspectos más importantes brindados por la plataforma ESB propuesta.

4.3.1 Intercepción de Mensajes

Uno de los requerimientos de mayor importancia de la solución es el análisis de todos los mensajes que se intercambian por la plataforma entre los distintos organismos para realizar los controles de calidad correspondientes. Para poder cumplir con este requerimiento fundamental se expone un *endpoint* en el cual se reciben todos los *requests*. Los mensajes recibidos deben contar con el cabezal WS-Addressing para poder ser procesados. Del mismo se extrae: el organismo origen del campo *From*, el servicio web destino del *To* y la operación específica del

web service del *Action*. En caso de no contener el cabezal, se retorna un mensaje de error, especificando la causa del mismo.

4.3.2 Gestión de Modelos de calidad

La solución se basa en interceptar los mensajes del ESB, y evaluar su calidad según el Modelo de Calidad establecido para el servicio que se está invocando o que está respondiendo. Por tal motivo, se ofrece la posibilidad de cargar Modelos de Calidad al sistema, a través del *Backoffice*, con sus Dimensiones, Factores, Métricas y Métodos.

Para dar de alta un Modelo de Calidad, una Dimensión o un Factor, alcanza simplemente con ingresar su nombre y descripción. Para dar de alta una Métrica, además de estos mismos datos, se debe ingresar la granularidad, semántica y unidad.

4.3.3 Gestión de Métodos y Acciones

Se busca que el sistema sea de lo más flexible posible, principalmente al momento de la creación de los métodos de un Modelo de Calidad, y sus correspondientes acciones en caso de que no se cumplan. Por este motivo, se considera esencial lograr un alto nivel de maleabilidad en la generación de los métodos y acciones, permitiendo la creación de estos de dos maneras: por medio de reglas básicas o utilizando métodos personalizados.

La utilización de reglas básicas permite a un usuario sin conocimientos de programación implementar Métodos o Acciones de mediana complejidad. Esto se logra por medio de la combinación de reglas de comparación, funciones de conjuntos, operaciones lógicas y/o expresiones regulares. Para los Métodos, por ejemplo, un usuario podrá definir un conjunto de reglas sobre los elementos del mensaje y determinar la relación lógica que estas deben cumplir para darlo como válido.

A modo de ejemplo, en la Tabla 4.1 y Tabla 4.2 se representa cómo se encuentra definido un Método, con dos reglas que lo componen, y la relación lógica que determina cómo se realizará la evaluación. En particular, la primer regla especifica que el valor del parámetro Sexo esté contenido en la lista “M, F, D”. Por otro lado, la segunda regla de la Tabla 4.1, especifica que el parámetro Documento tiene un valor de largo menor que 9.

Id Regla	Operation	Parámetro	Valor	Método
R1	Contenido	Sexo	M,F,D	1
R2	LargoMenor	Documento	9	1

Tabla 4.1. Ejemplos de Reglas.

Id Met.	Relación entre Reglas	Métrica	Operación	Acción
1	"R1 and (not R2)"	2	5	3

Tabla 4.2. Ejemplo de Método.

Al momento de realizar la validación del Método, se tiene en cuenta el campo *Relación entre Reglas* definido en la Tabla 4.2, sustituyendo los resultados de las evaluaciones de las reglas R1 y R2, para luego inferir el resultado final. Por ejemplo, si R1 evalúa *true* y R2 *false*, sustituyendo estos valores en la *Relación entre Reglas* se obtiene *true* como resultado del Método.

Por otra parte, para usuarios avanzados, es posible cargar en el sistema implementaciones personalizadas, brindándoles mayor flexibilidad, y permitiendo crear Métodos o acciones de mayor nivel de complejidad. Para esto, debe implementar completamente el código necesario del Método o de la Acción, cumpliendo con la interfaz necesaria para cada caso. La interfaz para los Métodos indica que se debe recibir el cuerpo del mensaje, y retornar el resultado de la validación. Para las Acciones, en cambio, se recibe el cuerpo del mensaje y se retorna un elemento del mismo tipo, por ejemplo, el mensaje transformado según cierta lógica.

Suponiendo que se desea controlar la validez de los elementos "cédula", y que no se tiene un Método que contemple dicha lógica, un usuario con conocimientos de programación podría realizar una implementación para ese Método, agregarlo dinámicamente a la plataforma y asociar esos elementos con el nuevo Método incorporado.

4.3.4 Registro de servicios

Para poder controlar los distintos mensajes intercambiados por medio de la plataforma, es necesario conocer los detalles de los Web Services disponibles. Para esto, cada vez que se expone un nuevo servicio en la plataforma, debe ser registrado en el sistema por un usuario administrador a través del sistema *Backoffice*.

El administrador debe ingresar: el archivo WSDL, el organismo al que pertenece, el *endpoint*⁷ del servicio web y una breve descripción del mismo. El sistema es capaz de detectar las operaciones que se encuentran en el servicio, así como los elementos de entrada y de salida de esas operaciones, a partir del archivo WSDL. Por otro lado, gracias al registro del *endpoint*, la plataforma es capaz de dirigir los mensajes al destino correspondiente de forma dinámica, en tiempo de ejecución.

⁷ URL donde el servicio puede ser accedido por una aplicación cliente.

4.3.5 Módulo de Backoffice

Es un sistema web de administración cuyos objetivos principales son gestionar los servicios de la plataforma y los Modelos de Calidad. A modo de adelanto, como se ve en las secciones 4.3.3 y 4.3.4, por medio de la misma, se podrán dar de alta los Métodos y Acciones, además de registrar los servicios, entre otras funcionalidades que se resumen a continuación:

- Registrar, listar, modificar y eliminar los servicios web publicados por los diferentes organismos pertenecientes a la plataforma ESB
- Alta, baja y modificación de los organismos participantes y sus respectivos usuarios para el acceso al sistema en cuestión.
- Alta, baja y modificación de Modelos de Calidad, especificando sus Dimensiones, Factores, Métricas y Métodos, creando las asociaciones Método-Elemento.
- Seleccionar las Acciones a realizar, en caso de que un Método no se cumpla, para cada par de asociaciones Método-Elemento.
- Acceder al *dashboard* y registro de eventos, que almacena los resultados de los controles de calidad, definidos por los Métodos dados de alta y las Acciones determinadas, así como todo lo relativo a validaciones y controles sobre los mensajes que circulan por la plataforma.
- Visualizar notificaciones de interés.

4.3.6 Registro para reportes

El sistema guarda registros con los datos más relevantes de cada mensaje SOAP, para su posterior procesamiento o creación de nuevos reportes y gráficos. Para cumplir con el objetivo de encontrar una solución extensible, se tomó la decisión de almacenar el registro en dos tablas. Por un lado se crea un registro llamado RegistroMensaje por cada mensaje que ingresa al sistema (ya sea *request* o *response*), y un registro en la tabla RegistroMetodo por cada evaluación de Método que es aplicado al mensaje. En el Apéndice 1, se presentan y describen las tablas, describiendo cuáles son los datos almacenados en ellas.

4.3.7 Estructura interna para el intercambio de mensajes

Los componentes de la plataforma interactúan entre sí, enviándose el mensaje de entrada junto a los datos que sirven para el control interno del flujo y el manejo del mismo. Para esto, se diseñó una estructura XML utilizada internamente en la plataforma. En la Figura 4.4 se muestran los campos de dicha estructura.

```

<CalidadDeDatosMessage>
  <idRegistroMensaje></idRegistroMensaje>
  <toUrn></toUrn>
  <toUrl></toUrl>
  <from></from>
  <isRequestMessage></isRequestMessage>
  <method></method>
  <action></action>
  <error></error>
  <finish></finish>
  <methodFail></methodFail>
  <body></body>
</CalidadDeDatosMessage>

```

Figura 4.4. Estructura XML interna, utilizada para el intercambio de mensajes.

Una vez que un mensaje ha sido recibido y validado, se crea una estructura como la mostrada anteriormente y se inicializa según se muestra en el Apéndice 2.

En la Figura 4.5 puede observarse el mensaje original, y cómo los valores de los campos *from* y *to* del cabezal WS-Addressing son los que luego se cargan en la estructura mostrada en la Figura 4.6.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:web="http://webservices.samples.iboss.org/"
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action>obtenerPersona</wsa:Action>
    <wsa:From>
      <wsa:Address>uy:edu:fing:pgrado:msp</wsa:Address>
    </wsa:From>
    <wsa:MessageID>uuid:acbfff7f9-511e-402b-adc4-6177ad01d475</wsa:MessageID>
    <wsa:To>uy:edu:fing:pgrado:dgreg:servicioDGREC</wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <web:obtenerPersona>
      <documento>12345678</documento>
    </web:obtenerPersona>
  </soapenv:Body>
</soapenv:Envelope>

```

Figura 4.5. Campos de un *request* original utilizados en la estructura interna.

En la Figura 4.6 puede observarse cómo se inicializa la estructura interna utilizada para intercambiar mensajes a lo largo de la plataforma, para un mensaje *request*. Los campos *toUrn* y *from* se cargan según el cabezal WS-Addressing del mensaje original, y en el campo *body* se ingresa el mensaje recibido (sin el cabezal mencionado). El valor *isRequestMessage* es inicializado en *true*, indicando que es un mensaje *request*, y las banderas *methodFail* y *finish* son inicializadas en *false*. El valor de *messageResult* se inicializa en *NO_EVALUA* y el campo *idRegistroMensaje* es inicializado en 0 (luego es cargado según el valor obtenido al momento de almacenar el mensaje en la tabla RegistroMensaje). También puede verse cómo el mensaje original es almacenado, sin el cabezal WS-Addressing en el *body* dentro de la estructura.

```
<CalidadDeDatosMensaje>
  <idRegistroMensaje>0</idRegistroMensaje>
  <toUrn>uy:edu:fing:pgrado:dgrec:servicioDGREC</toUrn>
  <toUrl>http://www.dgrec.com.uy:8080/ServerDGREC/ServicioPersonas</toUrl>
  <from>uy:edu:fing:pgrado:msp</from>
  <isRequestMessage>true</isRequestMessage>
  <messageResult>NO_EVALUA</messageResult>
  <methodFail>false</methodFail>
  <finish>false</finish>
  <body>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:web="http://webservices.samples.jboss.org/">
      <soapenv:Body>
        <web:obtenerPersona>
          <documento>12345678</documento>
        </web:obtenerPersona>
      </soapenv:Body>
    </soapenv:Envelope>
  </body>
</CalidadDeDatosMensaje>
```

Figura 4.6. Estructura interna inicializada en un *request*.

4.4 Arquitectura lógica

En esta sección se presenta la arquitectura lógica de la plataforma, se detalla cada uno de los componentes que la conforman y por último se describe la interacción que se da entre estos componentes en tiempo de ejecución.

La arquitectura propuesta está determinada principalmente por los siguientes requerimientos que la plataforma debe cumplir:

- Puntos de extensión que permiten agregar fácilmente nuevas funcionalidades a las ya implementadas. Estas funcionalidades serán principalmente nuevas formas de validar la calidad de un mensaje, así como nuevas Acciones a ejecutar.

Una vez que arriba un mensaje, el Gateway se encarga de realizar chequeos al mismo para verificar la validez de acuerdo a los requerimientos de la plataforma para los mensajes. Estos deben tener un cabezal WS-Addressing, y dentro de este se verifica que:

- El campo *from* haga referencia a un organismo habilitado para acceder a los servicios de la plataforma.
- El campo *to* haga referencia a un organismo que tiene algún servicio publicado en la plataforma.
- El campo *action* corresponda a una operación publicada en algún Web Service del organismo destino.

En caso de que el mensaje no cumpla con las validaciones se toma como un mensaje inválido y es cancelado, devolviendo un error indicativo.

Core

El Core recibe mensajes ya validados y transformados tanto desde el Gateway (invocaciones *request*) como desde el Dispatcher (mensajes *response*). Los transfiere al Router y cuando obtiene su respuesta, evalúa si debe continuar con el mensaje o cancelarlo, según el valor indicado en el campo *finish* de la estructura utilizada para el intercambio de mensajes. En caso de tener que continuar, lo redirige al Dispatcher (si es un *request*) o al Gateway (si es un *response*), para enviar la respuesta al cliente que consume el servicio. En caso de tener que cancelar el mensaje, el Error Handler genera una respuesta informando el motivo por el cual es cancelado, y la envía al cliente a través del Gateway.

Router

Este componente es el encargado de encaminar el mensaje de acuerdo a su contenido. Si existen Métodos pendientes para ejecutar sobre el mensaje, lo redirige hacia el Method Router. En cambio, si se deben ejecutar Acciones, lo envía al Action Router. Para el caso que no queden Métodos ni Acciones a aplicar, se retorna el mensaje hacia el Core.

Method Router

Los mensajes que contengan Métodos para ser ejecutados, son dirigidos al Method Router. Este invoca al componente de validación de la calidad configurado para ese Método (Quality Validator), y actualiza el mensaje para continuar con el resto de las Acciones o Métodos a ejecutar.

Action Router

Cuando un Método se evalúa negativamente, el mensaje es dirigido hacia el Action Router, donde se cargan las Acciones que deben ser ejecutadas. Luego, son ejecutadas secuencialmente sobre el mensaje recibido.

Quality Validator

Ejecuta los Métodos que fueron definidos en el Modelo de Calidad, y se encarga de actualizar la estructura interna según el resultado de esas evaluaciones

Actions

Ejecuta los distintos tipos de Acciones que se agreguen al sistema. Estas pueden ser: cancelar la invocación, enviar notificaciones o alertas, transformar un mensaje, entre otros. La plataforma está diseñada de forma tal que es posible agregar nuevas Acciones, en la medida que se necesiten.

Dispatcher

Obtiene de la estructura interna el destino y el mensaje, para enviarlo al Service Invoker. Luego recibe la respuesta, genera la estructura interna utilizada para procesar el mensaje y la redirige hacia el Core.

Service Invoker

Realiza efectivamente las invocaciones a los servicios externos según el mensaje y destino indicado por el Dispatcher, reenviándole la respuesta una vez obtenida

Error Handler

Genera los mensajes de respuesta cuando se debe retornar algún error, ya sea porque la invocación es incorrecta, o porque el mensaje no cumple con los modelos de calidad definidos.

Logger

Guarda los datos necesarios acerca de las invocaciones (exitosas y fallidas) de mensajes, así como los resultados de evaluación de los Métodos asociados a esos mensajes. Los datos registrados son los especificados en la sección 4.2.6. Los datos obtenidos pueden ser utilizados para la creación de reportes y gráficas sobre las estadísticas almacenadas.

4.4.2 Interacción entre componentes

A continuación se describen los pasos típicos de la plataforma para procesar un pedido de entrada y cómo los componentes interactúan entre sí. En cada paso se colocan referencias a la Figura 4.8 para facilitar la comprensión al lector.

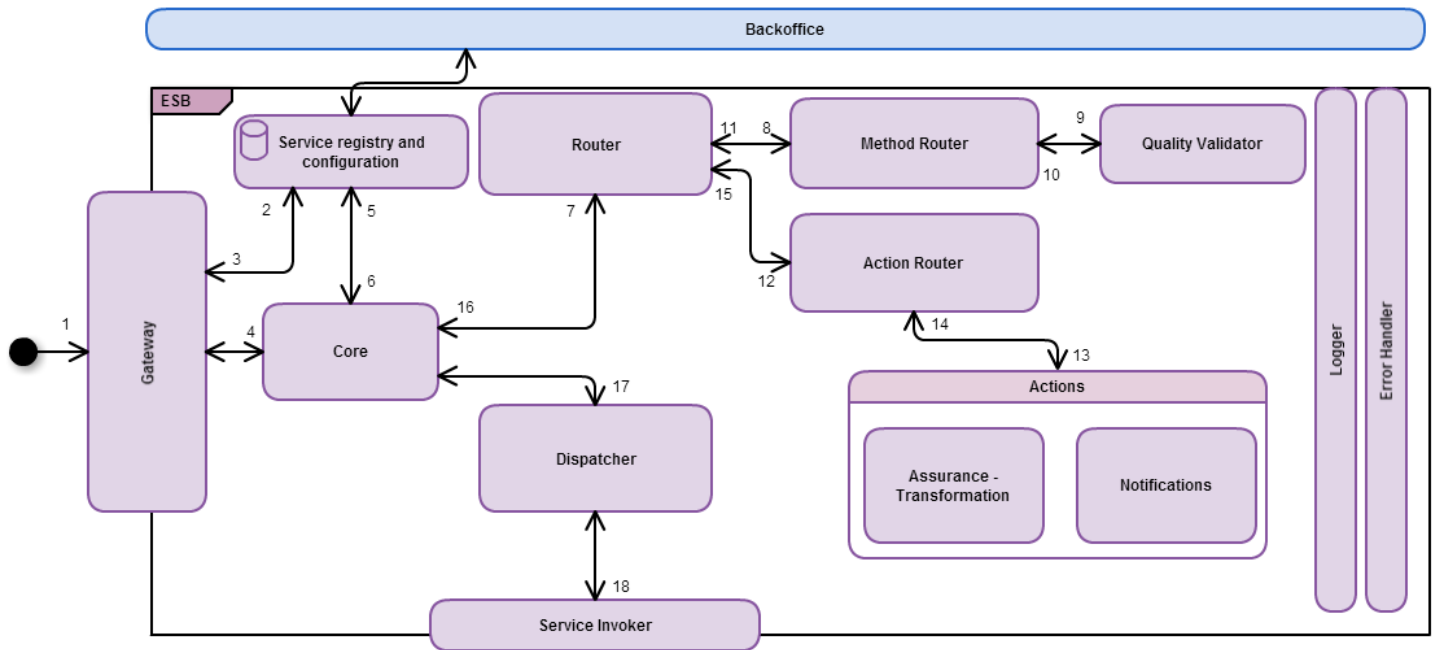


Figura 4.8. Flujo de un mensaje de solicitud procesado por la plataforma.

Cuando un mensaje llega al componente Gateway (1), se valida que contenga un cabezal WS-Addressing válido y que el origen y el destino del mismo estén habilitados en la plataforma (2). Aquí se utiliza el patrón Gateway⁸, para centralizar el acceso de los pedidos y validar el formato del mensaje y su contenido, de acuerdo a las configuraciones de la plataforma.

Luego de realizarse estas validaciones, se inicializa la estructura interna utilizada para el intercambio de mensajes dentro de la plataforma⁹, indicando que es un mensaje *request*. Una vez agregada la estructura al mensaje, se envía al componente Core (4), que se encarga de registrar la entrada (5) y de cargar en la estructura los Métodos correspondientes. Luego invoca al componente Router (7), que define de forma dinámica los siguientes pasos que se ejecutarán para el mensaje.

⁸ Descrito en la sección 2.2.3.

⁹ Se detalla en la sección 4.3.7.

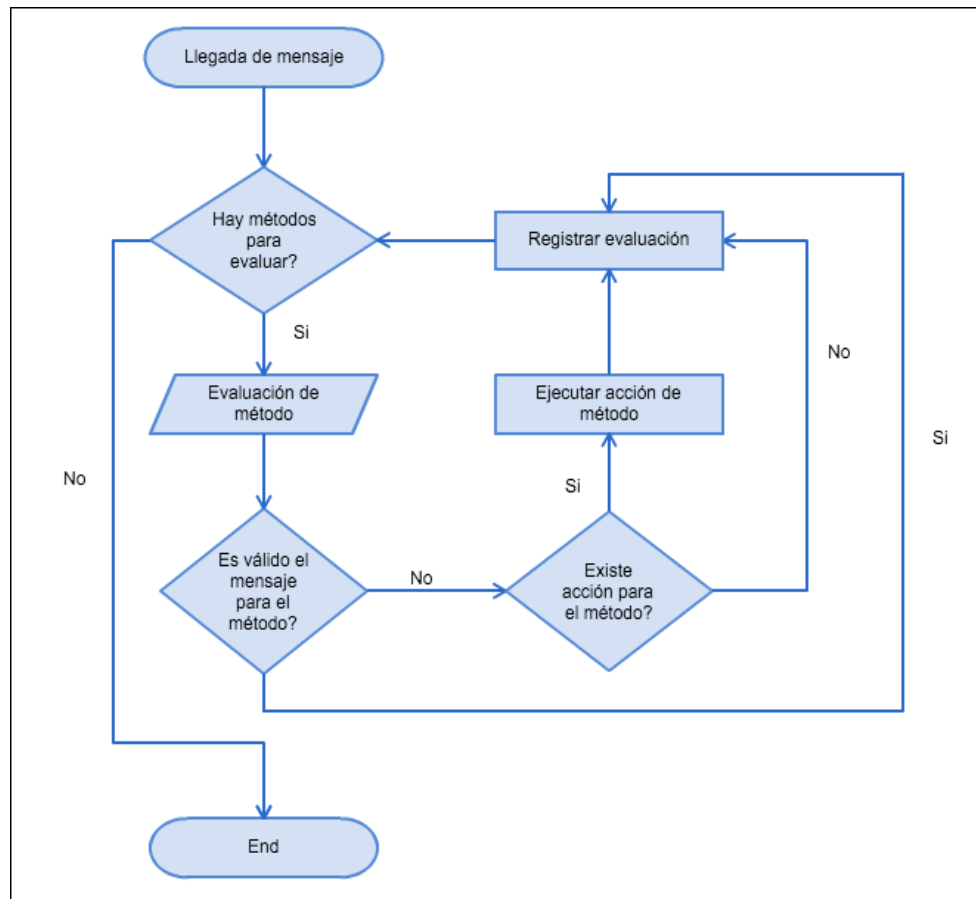


Figura 4.9. Diagrama de flujo de la validación de un mensaje.

En la Figura 4.9 puede verse un diagrama del recorrido que un mensaje toma desde que comienzan a evaluarse sus Métodos, hasta que ya no queda ninguno por evaluar. Primero, el Router detecta si se deben ejecutar Acciones, según la estructura interna que le ha llegado: si hubo un Método anterior que no se cumplió o quedan Acciones por ejecutar, entonces invoca al componente Action Router (12). En caso contrario, y si aún quedan Métodos por procesar, se invoca al Method Router (8). Si nada de lo anterior se cumple, finaliza su ejecución y devuelve el control al Core (16). El Router resuelve esa lógica utilizando el patrón Dynamic Router¹⁰, para determinar el siguiente componente que debe ser invocado, en vías de procesar el mensaje intercaladamente entre los Métodos y sus Acciones, según el estado del mismo. Por ejemplo: si se tienen dos Métodos con una Acción configurada cada uno, y el primero de los Métodos no se cumple, lo siguiente a ejecutar sería la Acción asociada a ese Método, después el segundo Método, y eventualmente, si este evalúa negativamente, su Acción asociada.

¹⁰ Descrito en la sección 2.2.3.

Luego de que el Service Invoker envía la solicitud al servicio externo, recibe la respuesta y la envía de vuelta al Dispatcher (1). Este valida el formato de la misma, inicializa la estructura interna y la envía al Core para ser procesada (2).

Una vez que el Core recibe la respuesta, le aplica un proceso similar que para el caso de un mensaje *request*, desde que invoca al Router (5) hasta que finaliza el procesamiento de Métodos y Acciones (14). En este caso, al ser un mensaje *response*, no se pasa el mensaje al Dispatcher, sino que se envía la respuesta al Gateway (15) para responder finalmente al cliente que realizó la solicitud original (16).

4.5 Flexibilidad y extensibilidad

Debido a su diseño compuesto por varios componentes específicos e independientes, la solución conceptual ofrece la posibilidad de reemplazar algunos de sus componentes. Esto permite, por ejemplo, agregar nuevas implementaciones para la resolución de Métodos y de Acciones, simplemente implementando clases que cumplan con la interfaz adecuada.

5 Detalles de Implementación del prototipo

En este capítulo se describen los detalles de la implementación del prototipo. En la sección 5.1 se presentan las principales herramientas utilizadas. Luego se destacan algunos aspectos propios de la implementación de la plataforma ESB en la sección 5.2 y del *Backoffice* en la sección 5.3. En la sección 5.4 se presentan las características de extensibilidad implementadas, y en la sección 5.5 las de flexibilidad. Además, en la sección 5.6, se plantean algunos problemas encontrados y se describe cómo fueron solucionados. Finalmente, en la sección 5.7, se detallan las limitaciones presentes en el prototipo construido.

5.1 Tecnologías utilizadas

En esta sección se presenta una breve descripción de las tecnologías utilizadas en la implementación del prototipo. Partiendo de los requerimientos detallados en el capítulo 3, se implementó en Java EE 7 y se utilizó el servidor de aplicaciones JBoss EAP 6.4. En la Figura 5.1 se muestra un diagrama de los componentes implementados, indicando para cada uno de ellos las principales tecnologías utilizadas.

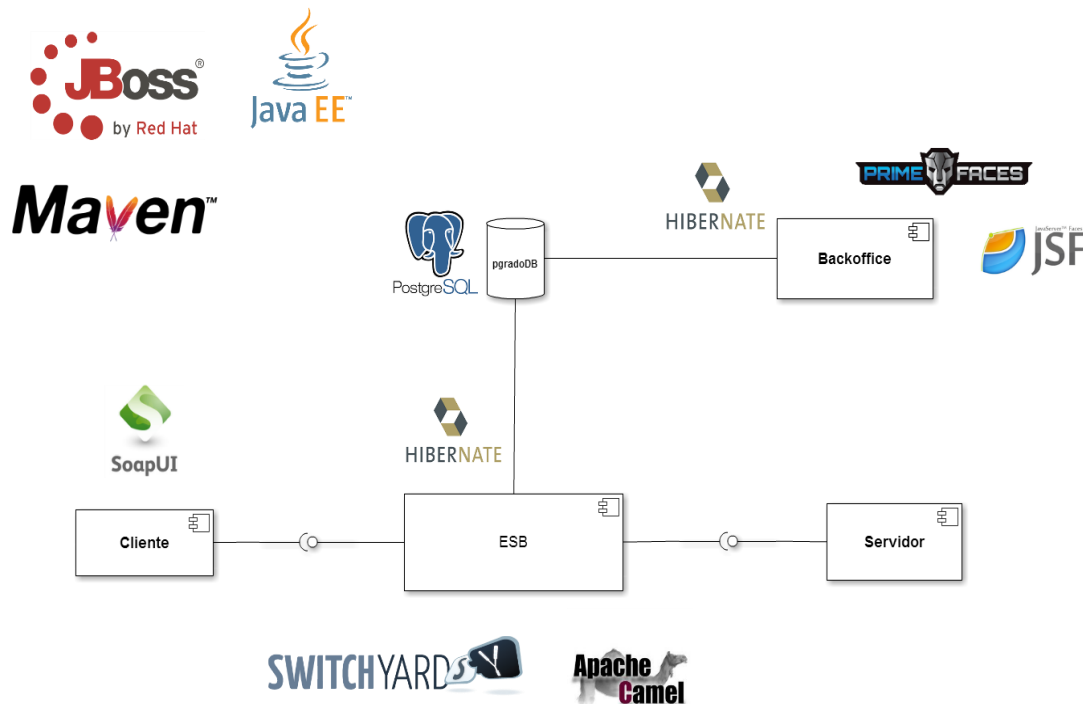


Figura 5.1. Diagrama de componentes simplificados con las principales tecnologías utilizadas.

5.1.1 Apache Maven

Maven [25] es una herramienta de software para la gestión y construcción de proyectos Java. Utiliza archivos de configuración POM (Project Object Model) basado en XML que permite manejar varios aspectos de los mismos, tales como: describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Además permite construir una estructura modular en la cual organizar los proyectos, existiendo normalmente un proyecto padre.

En la solución se utilizó Maven para definir la estructura modular del proyecto, compilación y gestionar las dependencias a librerías utilizadas.

5.1.2 Hibernate

Es una herramienta de Mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos XML o anotaciones en las entidades que permiten establecer estas relaciones [26]. Además facilita el manejo de consultas y la recuperación de datos [27]. 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.

Su configuración es muy sencilla, se debe especificar en un archivo XML los datos de conexión para la base de datos, el driver utilizado y las clases que se mapean con entidades de las base de datos. Además se encuentra integrado en JBoss EAP como un módulo nativo.

Los mapeos de persistencia se realizaron utilizando Java Persistence 2.0 Annotations (JPA) y anotaciones específicas de Hibernate [28].

Se utilizó una combinación de anotaciones de dos diferentes categorías:

- Anotaciones de mapeo lógico, se utilizan para describir, entre otros, el modelo de objetos y la asociación entre entidades.
- Anotaciones de mapeo físico, que describen el esquema físico, tablas, columnas, índices, etc.

5.1.3 PostgreSQL

PostgreSQL [29] es un sistema de gestión de bases de datos objeto-relacional, con su código fuente disponible libremente. Existe gran cantidad de librerías y otras herramientas que permiten su uso a través de una gran cantidad de lenguajes de programación. Es uno de los gestores de bases de datos más utilizados en el mundo del software libre debido a su gran velocidad de procesamiento a la hora de realizar las operaciones, su bajo costo en requerimientos para la elaboración de base de datos y su facilidad para instalarlo y configurarlo.

Se utilizó una base de datos SQL, para almacenar todas las estructuras gestionadas por el *Backoffice*, como Modelos de Calidad, organismos, Web Services y también el registro de logs generados por la plataforma ESB. Se decidió utilizar PostgreSQL porque se contaba con experiencia previa en su utilización, además de las características del motor ya mencionadas.

5.1.4 Switchyard

SwitchYard [30] es un *framework* diseñado para proporcionar soporte durante la implementación y administración de aplicaciones SOA. Contiene todas las características esperables en un típico ESB, además está integrado con Apache Camel añadiendo a JavaEE funcionalidades de ruteo y transformación de mensajes y orquestación de servicios.

Brinda diferentes *bindings* que permiten que un servicio brindando por la plataforma pueda ser accedido de diferentes formas, 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 [31].

Para facilitar la implementación a los desarrolladores, provee una extensión para Eclipse. La misma presenta una interfaz gráfica a través de la cual es sencillo diagramar, crear, modificar y eliminar componentes del ESB.

Se decidió utilizar Switchyard, ya que fue la recomendada por el cliente, y además, se tomó 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” [24]. 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.

5.1.5 Apache Camel

Es un *framework* Java de código abierto, enfocado en facilitar la integración de aplicaciones sobre distintas plataformas [32]. Para ello provee implementaciones concretas de los Patrones de Integración Empresariales (EIP, siglas del inglés Enterprise Integration Pattern) [33] más utilizados. Además brinda facilidades a la hora de utilizar Domain Specific Language (DSL) [34].

En particular para este proyecto se utilizó Camel para resolver el ruteo de los mensajes por la plataforma, y se utilizó en la implementación de forma eficiente de los patrones Content

Based Router y Dynamic Router. Además se utilizó el componente XSLT para realizar las transformaciones de los mensajes a partir de un archivo XSL.

5.1.6 Java Server Faces

Es un *framework* web basado en Java, orientado a componentes, para implementar interfaces de usuario. Proporciona un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entradas, definir un esquema de navegación de las páginas, manejo de errores, entre otras cosas.

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.

Su principal beneficio es que permite separar la lógica de negocio, de las vistas, y proporciona un Método fácil para enlazar ambas partes. Además, JSF está incluido en la plataforma Java EE, por lo que permite la creación de aplicaciones web sin agregar librerías extras a los proyectos.

5.1.7 Primefaces

Es una librería de código abierto, que cuenta con un conjunto de componentes enriquecidos para Java Server Faces, que facilita el desarrollo de interfaces de usuario. Es una librería liviana y muy sencilla de configurar utilizando Maven, ya que solamente se requiere incluir esta librería como dependencia dentro del proyecto web.

Se utilizó para el desarrollo de la interfaz de usuario del *Backoffice* y fue de mucha ayuda, en particular, para implementar el *file uploader* de archivos WSDL y transformaciones XSL, para las validaciones de los formularios y para los gráficos del *dashboard*.

5.1.8 Soap UI

Es una extensa herramienta de código libre muy versátil que permite probar, simular y generar código de servicios web de forma ágil, partiendo del contrato de los mismos en formato WSDL y con vínculo SOAP sobre HTTP, ideal para el testeado de aplicaciones SOA [35].

Se utilizó Soap UI, ya que es una herramienta altamente configurable, que permite de forma muy sencilla insertar el cabezal *WS-Addressing* a los mensajes SOAP. Por otro lado, también permite automatizar las pruebas a un *Web Service* utilizando lo que él llama Test Cases, con lo cual se realizaron las pruebas de performance que se detallan en la sección 6.2.

5.1.9 XPath

XPath es un lenguaje de consulta, definido por el W3C, para seleccionar nodos de un documento XML. En adición, XPath puede ser utilizado para computar valores (como *String*, *Number*, *Date*, etc) desde el contenido del documento XML.

XPath fue utilizado en gran medida al momento de procesar los mensajes que pasan a través de la plataforma, tanto los de entrada como los de salida. De esta manera es posible también transformar los elementos de los mensajes a clases Java, lo cual permite realizar las evaluaciones pertinentes definidas para cada modelo de calidad.

5.2 Plataforma ESB

5.2.1 Descripción de los principales componentes

La solución implementada en Switchyard ESB contiene los componentes presentes en el diagrama brindado por el entorno de desarrollo JBoss Developer Studio 8.1, que puede observarse en la Figura 5.2.

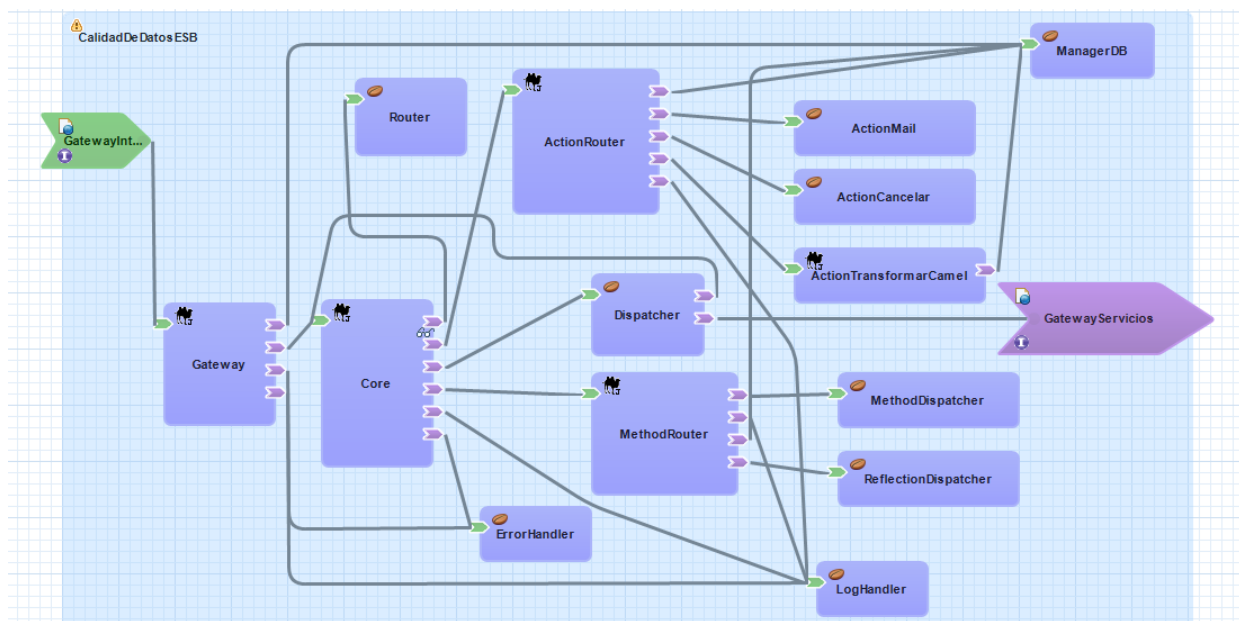


Figura 5.2. Diagrama switchyard.xml obtenido de la herramienta de desarrollo.

Gateway

Cumple la función de punto de entrada de los mensajes a la plataforma ESB. Como se observa en la Figura 5.3, este componente tiene una interface Java, con un *binding* que recibe las solicitudes HTTP que llegan a la ubicación de la plataforma, bajo el *path* CalidadDeDatosESB. En este sentido, se debe exigir a los clientes que desean consumir alguno de los servicios manejados por el ESB, que envíen sus solicitudes a la dirección {urlPlataforma}/CalidadDeDatosESB, con un cabezal WS-Addressing como se especifica a continuación:

- **To:** una referencia URN al servicio que se desea consumir. Esta referencia será usada internamente para identificar el servicio a invocar.
- **Action:** indica la operación específica del servicio que se desea consumir.
- **MessageID:** identifica de forma unívoca el mensaje.
- **From:** referencia URN, proporcionada por la plataforma, que identifica el organismo al cual pertenece el cliente que envía el pedido.

Todos estos campos están incluidos en las validaciones realizadas al mensaje.

```
<sca:service name="GatewayInterface" promote="Gateway/GatewayInterface">
  <sca:interface.java interface="uy.edu.fing.pgrado.CalidadDeDatosESB.GatewayInterface"/>
  <http:binding.http name="HttpGateway">
    <http:contextMapper class="uy.edu.fing.pgrado.CalidadDeDatosESB.GatewayContextMapper"/>
    <http:contextPath>CalidadDeDatosESB</http:contextPath>
  </http:binding.http>
</sca:service>
```

Figura 5.3. Especificación del servicio Gateway.

El componente Gateway cuenta con:

- Una implementación *Camel*, encargada de contener la lógica del ruteo del componente.
- Una referencia al componente ManagerDB, usado en este caso para validar el mensaje recibido, y luego para inicializar la estructura interna que será utilizada para el intercambio de mensajes dentro de la plataforma.
- Una referencia al componente Core, al cual se dirige el mensaje para ser procesado luego de ser validado correctamente.
- Una referencia al componente ErrorHandler, utilizado a lo largo de la plataforma para el manejo de errores, y en el Gateway particularmente, para enviar una respuesta adecuada al cliente, en caso de que su mensaje no pase las validaciones satisfactoriamente.
- Una referencia al componente LogHandler, utilizado para el registro de mensajes a lo largo de la plataforma. En este caso se lo utiliza para registrar los mensajes no validados.

Core

Es el encargado de interactuar con las principales partes de la solución. Recibe tanto los mensajes *request* por parte del Gateway, como los *response* por parte del componente Dispatcher. Al recibir un mensaje, genera un registro en el Registro de Mensajes, por medio del LogHandler. Luego, por medio del ManagerDB, carga en la estructura interna los Métodos correspondientes al mensaje recibido e inicia un ruteo dinámico, para lo cual utiliza el componente Router, encargado de determinar el siguiente destinatario: MethodRouter, ActionRouter o finalizar el ruteo, según el contenido del cabezal del mensaje recibido en el Core.

Una vez finalizado el ruteo, se actualiza el Registro de Métodos, guardando el resultado obtenido para el mensaje procesado. Luego, el Core se encarga de chequear si debe seguir evaluando Métodos o Acciones, o, si el resultado de alguna Acción indica que se debe cancelar el mensaje. En este último caso se envía el mensaje hacia el ErrorHandler para generar un mensaje de error adecuado; con el código 400 como estado del mensaje a devolver. Si no se debe cancelar y es un request, se envía hacia el Dispatcher, y si es un response se limpia el mensaje (se quita de la estructura interna) y se retorna al Gateway.

ActionRouter

Se encarga de cargar en la estructura interna las Acciones asociadas a un Método cuando no se cumple, y dirigir el mensaje al *endpoint* asociado al tipo de Acción a ejecutar. Este *endpoint* lo obtiene por medio del ManagerDB, ya que se guarda en la base de datos una relación entre el tipo de Acción a ejecutar y el *endpoint* del componente asociado.

Para el prototipo se definieron tres tipos de Acciones:

- **Acción-Mail:** envía un correo a la lista de direcciones, con un asunto y cuerpo, todos estos datos configurables a nivel de Acción. Además, se permite indicar si se desea incluir el mensaje interceptado y el Método que se evaluó negativamente asociado a la Acción. El envío del correo se hace de forma asincrónica, a través de una cuenta configurable por medio del archivo de configuración “calidaddedatos.properties”, que se describe en la sección 5.5.1.
- **Acción-Cancelar:** establece en el cabezal del mensaje interno utilizado, que se debe terminar el procesamiento del mensaje. Agrega además un código de error adecuado, que luego será utilizado por el ErrorHandler para generar el mensaje de respuesta al cliente. Existe además una propiedad definida en el archivo “calidaddedatos.properties”, que indica si ante una Acción-Cancelar, el procesamiento se debe finalizar inmediatamente, o si se debe continuar la ejecución del resto de los Métodos y Acciones antes de cancelar el mensaje. Esto último puede ser útil si se desea ejecutar la totalidad del procesamiento del mensaje, independientemente que se sepa que se debe cancelar.
- **Acción-Transformar:** ejecuta una transformación XSL, guardada en un archivo de transformación especificado por el usuario que configuró la Acción. Estos archivos son almacenados en la carpeta indicada con la propiedad “pathTransformaciones” del archivo “calidaddedatos.properties”.

Este componente actualiza la estructura de la siguiente manera: si hubo una Acción que determinó que se debe cancelar el mensaje y además está configurado el sistema para cancelar inmediatamente, se quitan los Métodos y las Acciones de la estructura para que el Router no siga ejecutándolos. Si no se cumplen las condiciones anteriores, se quita la última Acción procesada, y si no quedan más Acciones también se quita el Método, para seguir procesando el siguiente.

MethodRouter

Se encarga de invocar los componentes encargados de ejecutar los Métodos en el sistema, según el *endpoint* asociado a cada Método a procesar. A los efectos del prototipo se brindaron dos componentes para la ejecución de Métodos: *MethodDispatcher* y *ReflectionDispatcher*.

Luego de la ejecución del Método, se registra el resultado, y por último, si el Método no falló, se quita de la estructura para que se pase a ejecutar el siguiente. En caso de que el Método no se haya cumplido, se actualiza el valor *ResultadoMensaje* de la estructura utilizada, para reflejar que se evaluó con error.

Dispatcher

Recibe del Core los mensajes que deben ser enviados al servicio que el cliente desea consumir. Luego, obtiene los datos de la estructura interna utilizada para identificar la dirección destino y el cuerpo del mensaje a enviar. Invoca el servicio y obtiene su respuesta. Finalmente inicializa la estructura interna para el manejo de esa respuesta y envía dicha estructura al componente Core, para procesar.

5.2.2 Acción de transformación

Para implementar una Acción de transformación se optó por la utilización de Transformaciones XSL (XSLT) [36]. Esta se basa en expresiones XPath [37] para seleccionar y examinar la estructura del documento de entrada de la transformación.

Este mecanismo de transformación permite modificar los mensajes que se envían a través del ESB. SwitchYard permite nativamente ejecutar transformaciones XSL, lo cual resultó muy útil para la creación y ejecución de Acciones de este tipo.

En el Apéndice 3 se encuentran dos transformaciones implementadas a modo de ejemplo.

5.2.3 Validación de Mensajes

Cuando un mensaje *request* llega a la plataforma, se valida que el mismo cumpla con las siguientes condiciones:

- Debe contener el cabezal *WS-Addressing* con los campos *From*, *To*, *Action* y *MessageId*.
- El valor indicado en el campo *From* del cabezal *WS-Addressing* debe corresponderse con un organismo existente en la plataforma.
- El valor indicado en el campo *To* del cabezal *WS-Addressing* debe corresponderse a una URN de un servicio registrado en la plataforma.
- El cuerpo no puede ser vacío.

5.2.4 Manejo de Errores

Para el manejo de errores a lo largo del sistema, se incluyó en el cabezal implementado una lista que es actualizada a lo largo del procesamiento del mensaje. Cuando el mensaje es enviado al componente ErrorHandler, este prepara la respuesta que se enviará al cliente según los errores descritos en la estructura interna. La respuesta se construye como un mensaje *Soap Fault* [38]. Algunos ejemplos de respuestas de error pueden verse en el Apéndice 4.

5.2.5 Registro de Mensajes

Todos los mensajes que ingresan al sistema son registrados en la base de datos, junto con el resultado de la ejecución de cada uno de los Métodos que tienen asociados. Esta información es útil para generar estadísticas y reportes.

Con el fin de centralizar el registro de dicha información, se utiliza un componente exclusivo para su almacenamiento, llamado LogHandler. Este componente es invocado a lo largo de la plataforma, en las siguientes situaciones:

- El Gateway recibe un mensaje que no es válido (según las validaciones indicadas en la sección 5.2.4), y lo registra como tal en la tabla RegistroMensaje.
- Llega un mensaje al Core y se registra como válido en la tabla RegistroMensaje.
- En el MethodRouter, luego de ejecutar cada Método, se registra una entrada en la tabla RegistroMetodo con el resultado del Método ejecutado, y se asocia con la entrada del mensaje en la tabla RegistroMensaje.
- En el Core, luego de haberse procesado los Métodos y Acciones, se actualiza la entrada en RegistroMensaje con el resultado de la evaluación total del mensaje.¹²

5.3 Backoffice

La consola de administración está orientada a facilitar la administración, configuración y monitoreo de la solución propuesta. A continuación se presentan sus principales funcionalidades implementadas.

5.3.1 Gestión de Modelos de Calidad

A través del *Backoffice* se ofrecen funcionalidades que permiten realizar la gestión (crear, eliminar, modificar y visualizar) de los Modelos de Calidad en la plataforma, junto con sus respectivas Dimensiones, Factores, Métricas y Métodos.

5.3.2 Alta de Métodos

Una de las etapas más relevantes de la creación de un Modelo de Calidad, es la definición de sus Métodos. El *Backoffice* cuenta con una herramienta que permite de manera sencilla e

¹² El resultado de un mensaje es *true* si todos sus Métodos se cumplen o *false* en caso contrario.

intuitiva crear Métodos básicos para validar los mensajes (ver detalle en sección 5.5.2), así como también permite subir implementaciones de Métodos más complejos.

En un Método por Reglas se debe especificar un nombre, una descripción, una Métrica para el Método, el Web Service al cual aplicará el Método, y una operación del mismo. Una vez definidos estos datos, se cargarán dinámicamente todos los parámetros que pueden llegar en los mensajes (dependiendo si es de entrada o de salida), los cuales se recuperan de los datos obtenidos al momento de dar de alta el servicio con el WSDL. Teniendo en cuenta que también se recupera los tipos de cada dato, la página guía al usuario a definir Reglas dependiendo de estos tipos (el listado completo de Reglas puede verse en el Apéndice 5).

Por ejemplo, en caso de que exista un campo *nombre* con tipo *String*, se listarán las operaciones pertinentes, como pueden ser la de igualdad con otro *String*, que el largo del mismo sea mayor o menor a un número determinado, o su comparación con respecto a otro campo que provenga desde el mensaje.

5.3.3 Gestión de Organismos

De igual forma que con los Modelos de Calidad, a través de la consola de administración, se ofrecen las funcionalidades que permiten la gestión de los organismos junto con sus usuarios y el registro de los servicios web ofrecidos.

Para esto, el usuario Administrador cuenta con funcionalidades para dar de alta Organismos ingresando el nombre, código (el cual el sistema registra como `uy:edu:fing:pgrado:codigo`) que identifica al organismo en la plataforma, y descripción.

Posteriormente, es posible dar de alta usuarios para los organismos existentes, y los servicios web que ofrecen a través de la plataforma. Sobre el registro de servicios web se entrará en detalle en la sección 5.3.4.

La Figura 5.4 presenta el listado de los organismos ingresados con las funcionalidades *Ver*, *Editar*, *Eliminar* y *Agregar*.

proyGrado | Modelos de Calidad ▾ Web Services ▾ Organismos ▾ Metodo ▾ Reportes ▾

Organismos

1 de 1					
Nombre ↕	Código ↕	Descripción ↕	Ver	Editar	Eliminar
Ministerio de Salud Publica	uy.edu.fing.pgrado.msp	Descripcion del MSP			
Direccion General del Registro de Estado Civil	uy.edu.fing.pgrado.dgrec	Descripcion del DGREC			

1 de 1

+ Agregar

2015 Fing - PGrado - Versión 0.1 - Fecha 23/11/2015

Figura 5.4. Listado de organismos.

La consola de administración permite crear, eliminar, modificar y visualizar los Organismos, Usuarios y Web Services.

5.3.4 Gestión de Web Services

A la hora de dar de alta un Web Service en la plataforma, se cuenta con una interfaz que permite al administrador, luego de seleccionar el organismo al cual pertenece el servicio, subir el archivo WSDL del mismo. Se procesa el contenido del mismo, utilizando la librería `javax.wsdl`, para obtener de forma automática algunos de los datos del Web Service como el nombre, el *endpoint* y la lista de operaciones (distinguiendo si son *request* o *response*). Adicionalmente, junto a las operaciones se obtienen sus respectivos elementos.

La extracción de los datos directamente del archivo WSDL facilita la tarea al administrador para la creación de nuevos Web Services, por consiguiente se reduce la posibilidad de errores producidos por el ingreso manual. Igualmente, se permite editar de forma manual los campos nombre y *endpoint*, así como también, seleccionar qué operaciones del servicio interesa registrar en la plataforma, e ingresar una descripción del servicio, dando más libertad y flexibilidad al administrador a la hora de registrar los servicios.

En la Figura 5.5 se observa la interfaz para el registro de un nuevo Web Service en el sistema.

proyGrado | Modelos de Calidad ▾ Web Services ▾ Organismos ▾ Metodo ▾ Reportes ▾

Web Services

Registro

Organismo: Dirección General del Registro de Estado Civil ▾

WSDL: Seleccionar archivo Ningún archivo seleccionado Cargar

WebService: * ConsultaPersonasDGREC

Endpoint: * http://localhost:8080/servicios-DGREC/ConsultaPersonasDGREC

Descripción:

Operaciones	
☐	Nombre
☐	obtenerPersonaPorDocumento
☐	obtenerPersonaPorDocumentoResponse
☐	existePersona
☐	existePersonaResponse
☐	obtenerPersonasMayorEdad
☐	obtenerPersonasMayorEdadResponse

✓ Crear ✕ Cancelar

2015 Fing - PGrado - Versión 0.1 - Fecha 23/11/2015

Figura 5.5. Ejemplo de registro de un servicio web.

5.3.5 Dashboard

El *Backoffice* pone a disposición de los usuarios un *dashboard* donde es posible visualizar varias gráficas y registros sobre los controles de calidad aplicados a los mensajes que fluyen por la plataforma.

Para la construcción del *dashboard* se realizan consultas sobre las tablas RegistroMensaje y RegistroMetodo (ver sección 4.2.6), las cuales se presentan en el Backoffice

como gráficas o tablas de registros. Tanto para la implementación de las gráficas como de las tablas se utilizó el framework PrimeFaces 5.3.6.

A continuación se analizan las distintas gráficas y reportes implementados en el prototipo.

Resultado del total de los mensajes recibidos

La siguiente gráfica (Figura 5.6) representa el porcentaje de mensajes recibidos según el resultado de los Métodos aplicados sobre los mismos. Con este reporte se puede sacar como conclusión qué tan buena es la calidad de los mensajes que circulan por la plataforma, y cuántos de ellos cuenta con Métodos que evalúan su calidad.

En la Figura 5.6 se puede observar que el 54% de los mensajes no fueron evaluados por ningún Método, el 28% fue evaluado y todos sus Métodos devolvieron *true*, y el 18% restante, también fue evaluado por algún Método y al menos uno retornó *false*.

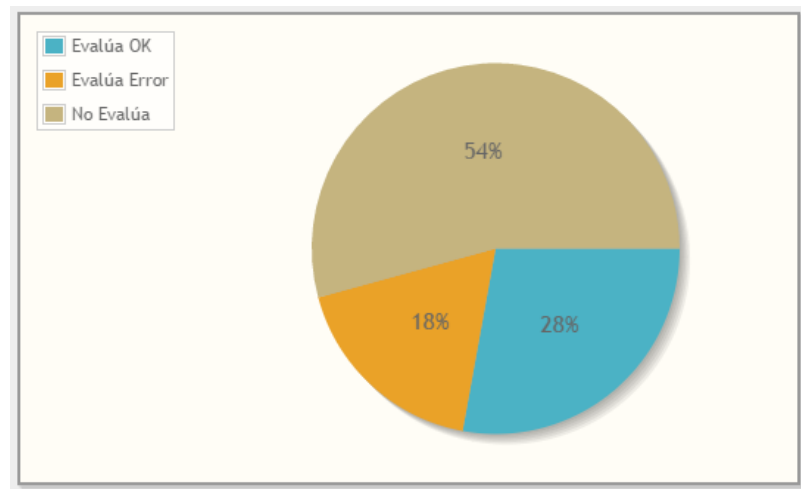


Figura 5.6. Resultado del total de los mensajes recibidos.

Resultado según Modelo de Calidad

Para la creación de la siguiente gráfica de barras se tienen en cuenta los mensajes evaluados de dos modelos de calidad, y se grafica la cantidad que tuvieron resultado “Evalúa OK” y “Evalúa Error”. En la Figura 5.7 se puede observar un ejemplo en el cual aproximadamente 110 mensajes evaluados por Métodos del Modelo de Calidad “Salud.uy” evaluaron OK de un total de 130. También, puede observarse que aproximadamente 20 mensajes de un total de 80, evaluados por el Modelo de Calidad “Personas”, evaluaron OK.

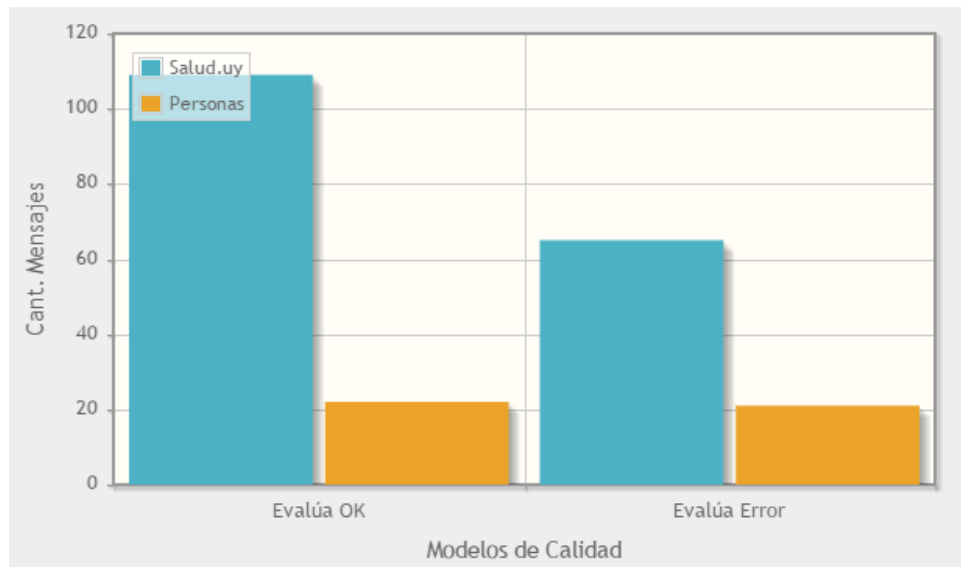


Figura 5.7. Resultado según Modelo de Calidad

Resultado de los mensajes de la última semana

Esta gráfica muestra la cantidad de mensajes recibidos en la última semana, permitiendo monitorear la fluctuación de mensajes y detectar si hay días en los que aumenta el tráfico entre organismos. Al igual que en la Figura 5.7, los mensajes están fraccionados según el resultado de los Métodos aplicados sobre los mismos.

En la Figura 5.8, se puede observar que el tráfico no varía mucho en general, notándose una leve disminución los últimos días. A simple vista se puede obtener el total de mensajes intercambiados entre organismos y diferenciar sobre cuántos de ellos se realizaron controles de calidad, distinguiendo entre estos últimos, cuántos pasaron los controles. Por ejemplo el 25/02 se intercambiaron 38 mensajes en total, de los cuales 12 evaluaron OK y 12 evaluaron ERROR.

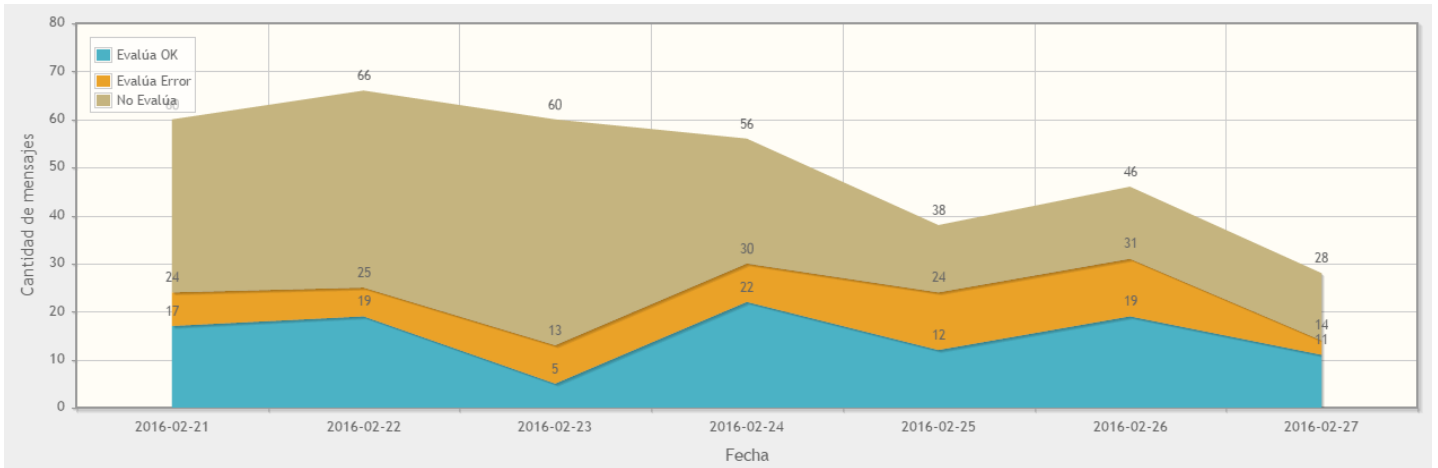


Figura 5.8. Resultado de los mensajes de la última semana

Detalle de los 10 mensajes más recientes

Por último, se presenta una tabla con el desglose de los últimos 10 mensajes recibidos, mostrando el tipo (*request* o *response*), *timestamp*, *from*, *to*, operación, resultado y la opción de visualizar el mensaje SOAP completo en formato XML.

En la Figura 5.9 que se presenta a continuación se muestra un ejemplo.

Tipo	TS	From	To	Operacion	Resultado	Mensaje
Response	27-02-2016 08:14:14	uy:edu:fin:pggrado:dgrec:cons	uy:edu:fin:pggrado:msp	obtenerPersonasMayorEdadF	NO_EVALUA	Ver
Request	27-02-2016 08:14:14	uy:edu:fin:pggrado:msp	uy:edu:fin:pggrado:dgrec:cons	obtenerPersonasMayorEdad	NO_EVALUA	Ver
Response	27-02-2016 08:14:08	uy:edu:fin:pggrado:dgrec:cons	uy:edu:fin:pggrado:msp	obtenerPersonaPorDocument	EVALUA_OK	Ver
Request	27-02-2016 08:14:07	uy:edu:fin:pggrado:msp	uy:edu:fin:pggrado:dgrec:cons	obtenerPersonaPorDocument	EVALUA_ERROR	Ver
Response	27-02-2016 08:14:03	uy:edu:fin:pggrado:dgrec:cons	uy:edu:fin:pggrado:msp	obtenerPersonaPorDocument	EVALUA_ERROR	Ver
Request	27-02-2016 08:14:03	uy:edu:fin:pggrado:msp	uy:edu:fin:pggrado:dgrec:cons	obtenerPersonaPorDocument	EVALUA_OK	Ver
Response	27-02-2016 08:14:00	uy:edu:fin:pggrado:dgrec:cons	uy:edu:fin:pggrado:msp	existePersonaResponse	NO_EVALUA	Ver
Request	27-02-2016 08:13:59	uy:edu:fin:pggrado:msp	uy:edu:fin:pggrado:dgrec:cons	existePersona	EVALUA_ERROR	Ver
Response	26-02-2016 07:13:24	uy:edu:fin:pggrado:dgrec:cons	uy:edu:fin:pggrado:msp	obtenerPersonaPorDocument	EVALUA_ERROR	Ver
Request	26-02-2016 07:13:24	uy:edu:fin:pggrado:msp	uy:edu:fin:pggrado:dgrec:cons	obtenerPersonaPorDocument	EVALUA_OK	Ver

Figura 5.9. Detalle de los últimos 10 mensajes recibidos.

5.4 Extensibilidad

5.4.1 Desarrollo de nuevos componentes

En vías de poder extender la solución brindada en el prototipo, se presenta cómo pueden agregarse nuevos componentes a la plataforma, particularmente de ejecución de Acciones y Métodos. Estos se encuentran destacados en el Figura 5.10, en la que se presenta una modificación al diagrama de arquitectura original.

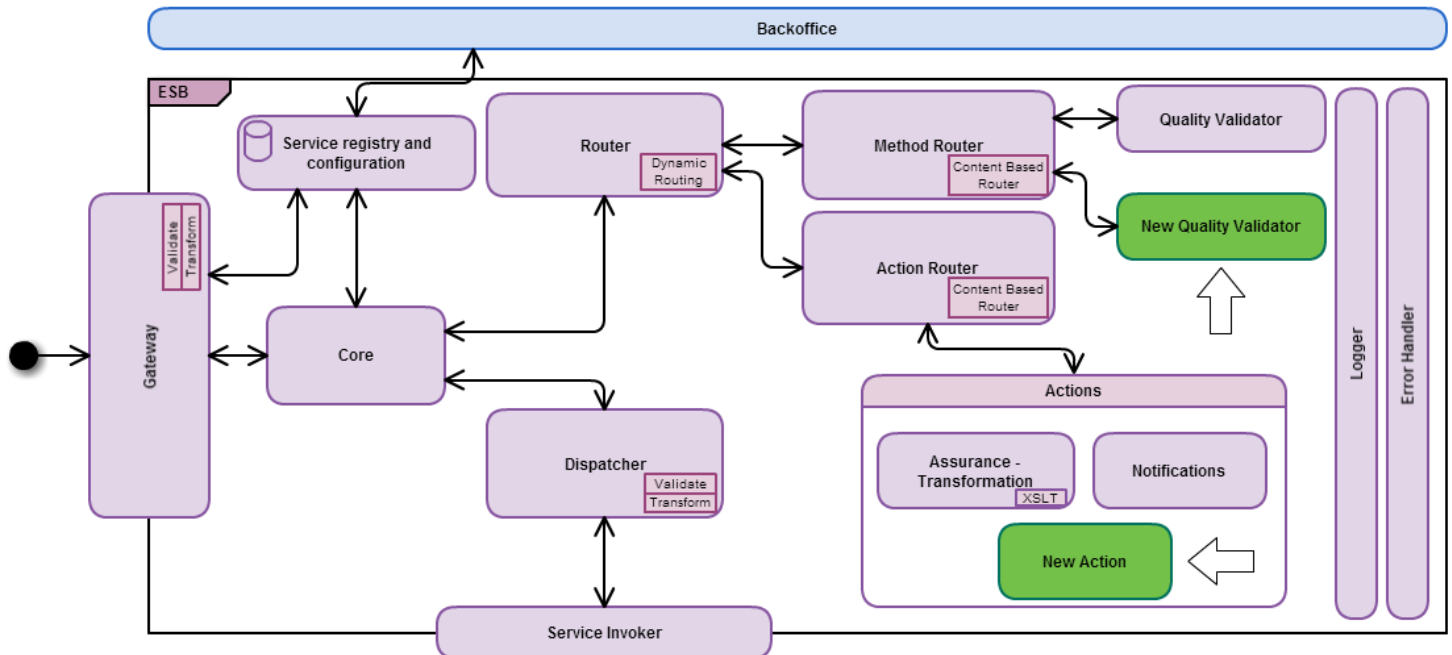


Figura 5.10. Diagrama de arquitectura luego de creados los nuevos componentes.

Componentes de ejecución de Acciones

Para poder agregar un nuevo componente de ejecución de Acciones a la plataforma, lo primero que debe hacerse es generar una clase que extienda de *Accion*. En ella se colocarán los datos particulares que esa Acción podrá utilizar. Por ejemplo, en la clase *AccionMail* se guarda el destinatario, asunto del correo, texto, un campo para indicar si en el mail se debe incluir el mensaje y otro similar para que el componente sepa si debe incluir en el cuerpo del correo el Método por el cual se dispara la Acción.

Luego deberá generarse un componente Switchyard, que tenga una función pública que reciba como único parámetro un objeto de clase *Mensaje*, definida en el *Backoffice* de la plataforma, y devuelva un elemento de la misma clase, que será el que se siga procesando. Por lo tanto, ese *Mensaje* debe respetar el formato de los valores recibidos por el componente, pudiendo ser modificados solamente el *body*, la bandera *finish* y la lista *errores*.

Para acceder al identificador del Método que no se cumplió y el de la Acción que se está ejecutando, el componente puede utilizar las operaciones *getNextMethod* y *getNextAction*, de la clase *Mensaje*. En base a estos identificadores es posible hallar las clases de los métodos que evaluaron negativamente el mensaje, y las acciones que fueron ejecutadas en respuesta a estos. Se puede ver un ejemplo de cómo hacer esto en el código del *ActionCancelar*, en el Apéndice 6.

Por último, deberá agregarse una entrada en la tabla *EndpointAccion*, que tenga como *accion* el nombre de la nueva *Accion* creada, y como *endpoint* la ubicación del nuevo componente, por ejemplo: "switchyard://NewAction".

Componentes de evaluación de Métodos

Para agregar un nuevo componente de evaluación de Métodos, se realiza un proceso similar al de agregar componentes para las Acciones, con algunas salvedades. La más importante es que los Métodos no extienden de ninguna clase en particular, por lo que no serán categorizados a la hora de rutear el mensaje hacia un *endpoint* u otro, sino que ese valor se definirá para cada instancia. Por lo tanto, no debe generarse una nueva clase para el Método, sino que directamente se puede pasar a generar el componente Switchyard.

De forma similar a las Acciones, el nuevo componente Switchyard agregado para definir una nueva forma de resolver Métodos, recibirá y devolverá un elemento de la clase *Mensaje*. El único valor que puede ser modificado entre el elemento recibido y el retornado es el campo '*methodFailed*', que toma valores *true* o *false* y sirve para indicar al sistema si el Método fue evaluado positivamente o no.

Se podrá acceder a los datos del *Mensaje* y obtener la instancia del Método que se está evaluando, de forma similar a como se resuelve en los componentes para la ejecución de Acciones.

El nuevo componente deberá ser agregado como referencia del componente *MethodRouter*, para que Switchyard pueda resolver su ubicación en tiempo de ejecución.

Para cada Método se debe indicar en su campo *endpoint*, la ubicación del componente, por ejemplo: si el nuevo componente se llama *NewQualityValidator*, el campo *endpoint* debe tomar el valor: "switchyard://NewQualityValidator".

Por defecto se cuenta con dos componentes de evaluación de Métodos: *MethodDispatcher*, que evalúa los Métodos mediante el sistema de reglas descrito en la sección 4.3.3, y *ReflectionDispatcher*, que evalúa los Métodos Personalizados. La lógica, implementada mediante un Método Personalizado, es agregada a la solución e invocada mediante *Reflection*¹³.

¹³ Es una API de Java utilizada para inspeccionar e invocar clases, Métodos y atributos dinámicamente en tiempo de ejecución, sin conocer, a priori, los tipos y nombres de las clases específicas con las que se trabaja.

Para lo cual, en el Método se debe indicar el *endpoint* del componente ReflectionDispatcher y el nombre de la nueva clase que lo resuelve.

Esta solución para agregar nuevos Métodos mediante Reflection, tiene como restricción, que la nueva clase debe implementar una función llamada “resolveMethod” (nombre configurable en el archivo “calidaddedatos.properties”, descrito en la sección 5.5.1), que recibe un *String* y devuelve *true* o *false*, indicando si el mensaje recibido es evaluado por el Método de forma positiva o negativa. En el Apéndice 7 puede observarse el ejemplo de la implementación de un Método para poder ser utilizado de esta forma.

5.4.2 Dashboard

El *dashboard* es una representación gráfica de los principales indicadores que intervienen en la plataforma, y tiene como objetivo la toma de decisiones para optimizar las estrategias de los organismos. En pocas palabras, el *dashboard* debe transformar los datos en información útil para un usuario de la plataforma.

En el prototipo, se implementaron apenas unos pocos ejemplos de lo que es posible hacer con los datos recabados por la plataforma y almacenados en las tablas *RegistroMensaje* y *RegistroMetodo*. Estas fueron pensadas para generar un registro lo más completo y útil posible, almacenando los principales indicadores del procesamiento de mensajes, con mucho potencial para la generación de gran variedad de registros, y así brindar nueva información a los usuarios.

Con toda esta información disponible en la base de datos, se logró una solución fácilmente extensible, ya que se cuenta con toda la traza de actividad que realiza cada uno de los mensajes que circulan por la plataforma.

5.5 Flexibilidad

El sistema permite al usuario parametrizar algunas características del mismo, con el fin de lograr un sistema configurable en varios de sus componentes.

5.5.1 Propiedades globales

El sistema cuenta con un archivo externo “calidaddedatos.properties”, modificable en tiempo de ejecución, desde el cual se recuperan propiedades pertinentes para la funcionalidad del mismo. En el Apéndice 9 se presentan algunas de las propiedades utilizadas en el prototipo.

5.5.2 Métodos

Al momento de dar de alta los Métodos de evaluación, el sistema ofrece una cantidad básica de operaciones para la creación de Reglas de validación, que permiten al usuario definir cómo desea que se realicen las evaluaciones. Combinando estas reglas es posible realizar Métodos personalizados, siendo posible extenderlos agregando validaciones propias como ya se explicó anteriormente.

Algunas de las operaciones básicas definidas en el sistema incluyen:

- Comparaciones entre números, fechas, y cualquier clase que pueda extender la interface Comparable de Java (<, ≤, >, ≥, ==, !=).
- Validaciones para *String*'s, igualdad, desigualdad, contenido en, largo, expresiones regulares.
- Validaciones genéricas para tipos complejos, principalmente verificando la existencia del campo en el mensaje.
- Validaciones para listas, todas las validaciones anteriores son aplicables a listas de elementos del mensaje, verificando que todos los elementos cumplan, por ejemplo, que son mayores a cierto valor especificado, o en caso de ser una lista de *String*, que no son iguales a cierto texto ingresado.
- Validaciones de conjuntos, permiten verificar que un cierto elemento se encuentra (o no) en un conjunto de elementos ingresado al momento de definir la regla, cualesquiera sean los tipos de ellos. (Ej: Montevideo ∈ [Durazno, Montevideo, Flores]).
- Validaciones entre elementos del mensaje, permite realizar comparaciones de cualquier tipo entre todos los campos de un mensaje. Es posible realizar una validación que, por ejemplo, determine la coherencia entre los campos "fecha de nacimiento" de tipo *Date* y "edad" de tipo *Integer*.

En adición, es posible definir la relación lógica que vincula a todas las reglas definidas para un Método. Utilizando los operadores lógicos convencionales and, or, y not, y el uso de paréntesis, se le permite al administrador declarar la forma de evaluación, como por ejemplo, en caso de que un Método esté formado por 4 reglas, de R1 a R4, podría ingresarse una relación para que sean evaluadas de la manera "R1 and (R2 or !R3) and R4".

Combinando todas las herramientas que ofrece el sistema para la generación de Métodos, es posible generar modelos de calidad bastante abarcativos, permitiéndoles a los administradores generar validaciones para los mensajes con un alto grado de flexibilidad y lograr así un mayor control sobre la calidad de los datos.

Además, mediante los Métodos Personalizados, se permite que un usuario con conocimientos de programación implemente su propia lógica de resolución de Métodos y la agregue de forma dinámica a la plataforma.

5.6 Problemas Encontrados

5.6.1 Invocación dinámica a servicios externos

Uno de los principales problemas encontrados durante la implementación del prototipo fue la imposibilidad, durante mucho tiempo, de realizar invocaciones de forma dinámica a distintos Web Services. El problema fue que con la versión de Switchyard utilizada en un

principio, al configurar una referencia HTTP o SOAP se debía indicar el *endpoint* del Web Service específico a consumir, no pudiendo ser modificado posteriormente en tiempo de ejecución.

Luego de muchas pruebas e investigación para intentar solucionarlo, se encontró que esta era una limitante de Switchyard y que esta limitación era discutida en foros donde se planteaba el problema [39], sin solución en aquel entonces, por lo cual se analizaron otras alternativas. Una de ellas fue dar de alta una referencia para cada uno de los Web Services registrados en la plataforma, teniendo como contrapartida que al querer agregar un servicio, se debía implementar esa nueva referencia y volver a publicar la plataforma. Esta alternativa no conformó al equipo, considerando la invocación dinámica de servicios como un requerimiento básico de la solución, ya que de lo contrario habría que reiniciar la plataforma con cada nuevo servicio que se agregara.

Siguiendo con la búsqueda de una mejor solución, se encontró en uno de los foros en los cuales se había visto que el problema se presentaba como una limitante de Switchyard [39], que la comunidad de JBoss planteaba una solución mediante la cual podría hacerse este ruteo dinámico, pero que implicaba una actualización de uno de los componentes de Switchyard. Este cambio fue realizado en una versión beta de la plataforma, la cual por este motivo se terminó utilizando en este proyecto.

5.7 Limitaciones

Si bien el proyecto se enfoca en una solución conceptual a la problemática planteada, es importante recalcar cuáles fueron las limitaciones que surgieron al momento de la implementación del prototipo, ya que a partir de estas se puede determinar cuáles son los principales aspectos a tener en cuenta en caso de querer extenderlo.

5.7.1 Procesamiento WSDL

Para inspeccionar los archivos WSDL se utilizó la API brindada por la biblioteca `javax.wsdl`, la cual simplifica varios de los problemas para interpretar los archivos, permitiendo abstraerse en gran parte del tipo de documento y de la versión de WSDL (1.2 o 2.0).

La implementación del procesador de documentos WSDL, utilizado en el *Backoffice* para registrar los Web Services, debió afrontar la problemática de que la declaración de los elementos en el WSDL pueden ser definidos en un archivo XSD externo, al cual se puede hacer referencia por medio de la función *import*. Debido a los tiempos del proyecto, se decidió que el *procesador* soporte únicamente documentos WSDL que contengan los elementos en el mismo documento, sin permitir *imports*.

Otro problema encontrado y que quedó fuera del alcance de este proyecto, es la posibilidad de detectar cuándo un parámetro es opcional. Para esto se puede inspeccionar el

documento WSDL y obtener para cada parámetro su valor mínimo de ocurrencia (min-occur), o utilizar algún tipo de documentación adicional que indique su opcionalidad.

Por último, también se decidió acotar la solución aceptando como válidos los elementos definidos respetando cierto formato. Por un lado, los elementos simples, incluyendo de forma obligatoria el atributo *name* y como valores posibles del atributo *type*: *string*, *int*, *dateTime*, *float*, *double* y *boolean*. Por otro lado, también se aceptan elementos complejos (*complexType*) definidos con el atributo obligatorio *name* y respetando la estructura que se muestra en la Figura 5.11.

```
<xs:complexType name="obtenerPersonaPorDocumento">
  <xs:sequence>
    <xs:element minOccurs="0" name="documento" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Figura 5.11. Ejemplo de elemento complejo “obtenerPersonaPorDocumento” válido.

5.7.2 Acciones

Con respecto a las Acciones, se brindan tres tipos: uno para el envío de mails, otro para cancelar el mensaje y otro para la transformación de mensajes mediante XSL. El prototipo se encuentra restringido a estas tres Acciones, aunque la estructura del mismo fue pensada con la finalidad de poder extenderlo de manera relativamente sencilla. El enfoque no se centró en buscar Acciones más complejas, ya que estas pueden depender mucho de las particularidades del ambiente al cual pertenece el mensaje.

La otra limitante con respecto a las Acciones es que la forma en la que se realiza el manejo de estos componentes en la plataforma Switchyard, implica que ante nuevas implementaciones de Acciones, deban generarse estos nuevos componentes, y configurarse en el *ActionRouter*, lo que implica no poder aplicar estos cambios en tiempo de ejecución, sino que se debe reiniciar el sistema para que los cambios se vean reflejados.

5.7.3 Manejo de listas

Al momento de dar de alta un Método, es posible definir una regla que permita realizar una comparación de un elemento del mensaje contra una lista ingresada por el usuario. Como ejemplo podría indicarse si un número cualquiera se encuentra en un arreglo de números definido por el usuario.

Actualmente las listas que se pueden ingresar para esta validación, se encuentran acotadas al ingreso a través de un campo en la página de alta de Métodos. En una segunda

etapa se podría considerar como una mejora poder cargar listas a partir de un archivo, o incluso poder validar a través de otros Web Services a dónde realizar las consultas.

5.7.4 Manejo de errores

Existe una clara estructura para el manejo de los errores, contando con un componente específico para esto, pero para el prototipo no se profundizó en la implementación de estos mensajes, resultando ser un conjunto acotado de errores, que puede tornarse poco específico, además de ser no configurables.

5.7.5 Manejo de tipos

Si bien los mensajes que se generan basados en un WSDL pueden tener una gran cantidad de tipos básicos (*String*, *Date*, *Double*, *BigInt*, etc) por razones de no sobre extender el prototipo, se acotó la cantidad de tipos a manejar, estos son: *String*, *Boolean*, *Integer*, *Double*, *Float* y *Date*.

La razón por la cual la solución no maneja de forma transparente el tipo de los elementos, se debe a la forma en que se envían cuando se consume un Web Service. El tipo de los mismos se encuentra definido por el WSDL, del cual es recuperado para poder realizar un casteo de los elementos que viajan en el mensaje, y así poder utilizarlos como elementos del lenguaje. Si bien es sencillo extender la solución para soportar más tipos, se priorizaron otros aspectos de mayor relevancia, considerando las clases manejadas hasta el momento suficientes para abarcar una amplia gama de validaciones sobre los mensajes de una realidad.

6 Caso de Estudio y Pruebas Realizadas

En esta sección se presenta el caso de estudio definido para validar la solución planteada y mostrar el funcionamiento general del sistema.

6.1 Caso de Estudio: Plataforma de Salud

El caso de estudio analizado en este proyecto está centrado en la Plataforma de Salud (PS). Esta plataforma está implementada como una extensión de la Plataforma de Gobierno Electrónico (PGE), reutilizando y extendiendo los componentes de seguridad e interoperabilidad de la misma.

Las PGEs consisten en una infraestructura de *hardware*, *software* y comunicaciones que permiten integrar sistemas en un marco de Gobierno Electrónico, aportando también funcionalidades y servicios adicionales (por ejemplo seguridad, monitoreo, etc.). Resultan altamente apropiadas para implementar funcionalidades inter-organismo.

La plataforma se basa en una arquitectura SOA, la cual cuenta con un sistema de control de acceso, un sistema de gestión de metadatos y una plataforma de *middleware*, integrada con tecnología ESB. En este marco, se facilita que los organismos interactúen en base a una SOA, caracterizada por modelar las aplicaciones como servicios, donde los organismos pueden utilizar esta plataforma para publicarlos y descubrirlos. Esto resulta en un marco adecuado para implementar controles de calidad sobre los datos generados por dichos servicios.

En la Figura 6.1 puede verse la ubicación de la plataforma ESB, dentro del contexto de la Plataforma de Gobierno Electrónico.

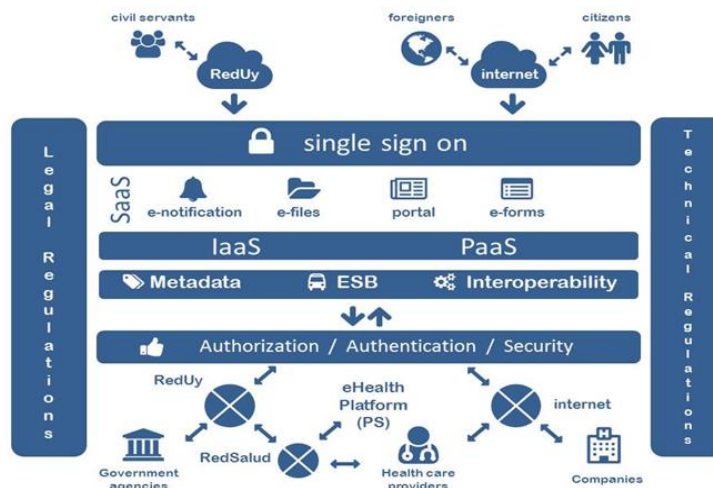


Figura 6.1. Esquema de la PGE [40].

6.2 Servicio Utilizado

Con motivo de evaluar el prototipo implementado, se toma como caso de estudio el conjunto de servicios expuestos por el Sistema del Certificado de Defunción, brindado por el Ministerio de Salud Pública, en el contexto del proyecto llamado Sistemas Estadísticas Vitales, Vacunas, Embarazo y Niñez (SEVEN).

En este caso de estudio se maneja una versión simplificada de dicho servicio, donde se utilizaron los parámetros de mayor relevancia, que permiten mostrar todo el potencial y funcionamiento de la solución. El servicio mencionado cuenta con las operaciones: *obtenerCertificado* y *obtenerListaCertificados*, las cuales se describen en detalle en el Apéndice 8.

6.3 Modelo de Calidad utilizado

Para la definición del Modelo de Calidad utilizado, se determinó en primer lugar las Dimensiones y los Factores a medir. En este caso, se define un Modelo de Calidad, para el servicio descrito anteriormente.

6.3.1 Dimensiones y Factores a medir

Las Dimensiones y Factores que se identificaron para medir la calidad de los datos que se intercambian en los servicios son los indicados en la Tabla 6.1

Característica	Dimensión	Factor
Los documentos intercambiados deben contener los datos mínimos establecidos por Salud.uy	Complejidad	Densidad
Los datos deben ser precisos con el fin de ser útil y de generar información correcta. La documentación debe reflejar el evento como realmente sucedió	Exactitud	Correctitud Semántica
El formato de los datos mínimos tiene que ser el establecido	Exactitud	Correctitud Sintáctica
El dato debe pertenecer a un dominio definido	Consistencia	Integridad del Dominio
Los datos deben ser oportunos, llegar a tiempo para su uso específico	Frescura	Oportunidad
Los datos relacionados deben satisfacer reglas semánticas definidas	Consistencia	Integridad Intra-relación

Tabla 6.1. Definición de Dimensiones y Factores utilizados.

6.3.2 Métricas

Las Dimensiones y Factores de calidad son definidos de forma general para cualquier conjunto de datos que se defina.

Al momento de definir una Métrica se especifican, para cada una de ellas, los siguientes datos:

- **Definición:** qué mide la Métrica.
- **Tipo del resultado** (de la medida que se obtiene): por ejemplo: (0,1).
- **Granularidad:** es la unidad de datos sobre la que se aplica la Métrica. Puede ser un dato o un conjunto de datos

Las Métricas definidas para los datos que se intercambian en el contexto Salud, según las características de calidad que se quieren evaluar, se listan en el Apéndice 10.

Para el servicio web *obtenerCertificado* se aplicarán las Métricas a sus elementos según el tipo de dato de cada uno de ellos, como se describen a continuación en la Tabla 6.2 para la operación *obtenerCertificado (request)* y en la Tabla 6.3 para *obtenerCertificadoResponse (response)*:

obtenerCertificado	
Dato	Tipo de dato
numeroCertificado	Campo numérico
tipoDocumento	Tipo Documento
paisDocumento	Pais
numeroDocumento	Documento de Identidad

Tabla 6.2. Tipo de datos de los elementos de la operación *obtenerCertificado*.

obtenerCertificadoResponse	
Dato	Tipo de dato
numeroCertificado	Campo numérico
tipoDocumento	Tipo Documento
paisDocumento	País
numeroDocumento	Documento de Identidad
primerNombre	Campo texto
primerApellido	Campo texto
lugarDefuncion	Lugar Defunción
codigoInstitucion	Institución de Salud
codigoEstablecimiento	Establecimiento de Salud
codigoDepartamento	Departamento de Uruguay
codigoLocalidad	Localidad Departamento de Uruguay
direccionDefuncion	Texto libre
estadoCivil	Estado Civil
Nacionalidad	País
fechaDefuncion	Fecha Evento
fechaNacimiento	Fecha pasada
Sexo	Sexo
fechaFirmaCertificado	Fecha pasada
cedulaMedicoFirmante	Documento de Identidad
cjppuMedicoFirmante	Personal de Salud

Tabla 6.3. Tipo de dato de los elementos de la operación *obtenerCertificadoResponse*.

Para el servicio web *obtenerListaCertificados* sólo se aplicarán las Métricas a los elementos del *response obtenerListaCertificadosResponse*, según el tipo de dato de cada uno de ellos, como se describen a continuación en la Tabla 6.4:

obtenerListaCertificadosResponse	
numeroCertificado	Campo numérico
numeroDocumento	Documento de Identidad
paisDocumento	País
tipoDocumento	Tipo Documento

Tabla 6.4. Tipo de datos de los elementos de la operación *obtenerListasCertificadoResponse*.

6.4 Configuración del caso de prueba

Para este caso de estudio, se asume que ya fueron dados de alta en el sistema los usuarios administradores y los organismos listados a continuación:

- Ministerio de Salud Pública
- Dirección General del Registro de Estado Civil

Se comenzó el caso de estudio dando de alta el servicio expuesto por el Sistema del Certificado de Defunción especificado en la sección 6.2, para lo cual el administrador registra el Web Service subiendo el archivo WSDL correspondiente al servicio web, con lo cual se obtienen automáticamente los datos de las operaciones.

6.4.1 Alta de Métodos

La mayoría de las Métricas planteadas en el Modelo de Calidad (ver sección 6.3.2) fueron implementadas utilizando la herramienta provista por el *Backoffice*, a través de conjuntos de Reglas, empleando algunas de las operaciones ofrecidas por la misma (el listado de estas operaciones puede verse en el Apéndice 5).

En el Apéndice 11 se detallan los Métodos utilizados en el caso de estudio, definidos a partir de Reglas.

Por otro lado, aprovechando la flexibilidad de la solución, se implementaron Métodos personalizados, para poder realizar los controles que definen las Métricas más complejas y que escapan del alcance de la herramienta mencionada.

Métodos personalizados

Se utilizaron Métodos personalizados para implementar las validaciones que, dada su complejidad, quedaban por fuera del alcance de la modalidad de Métodos por configuración de Reglas, utilizada para la resolución de las validaciones anteriores.

Los Métodos que se resolvieron por medio de implementaciones personalizadas fueron *VerificaCIUruguay*, *VerificaLocalidadDepartamento* y *VerificaEstablecimientoInstitucion*.

En el primero se valida que el mensaje recibido contenga una cédula válida según el algoritmo de verificación de cédula uruguaya, en el segundo se verifica que la localidad pertenezca al departamento indicado y en el tercero que el establecimiento se corresponda a la institución enviada en el mensaje.

Las implementaciones de estos Métodos son cargadas por medio de la herramienta de *Backoffice*, y luego son integradas a la plataforma por medio de un usuario administrador. Un ejemplo de implementación personalizada de Métodos puede verse en el Apéndice 7.

6.4.2 Alta de Acciones

Al momento de dar de alta cada uno de los Métodos es necesario indicar las Acciones que se realizarán en caso que los mismos arrojen un resultado negativo, es decir, los datos no cumplen con las validaciones de calidad.

Para simplificar, se les asoció la Acción de cancelar mensaje a todos los Métodos dados de alta. Para el Método Nulo, asociado a la operación *obtenerCertificado* (solicitud), se asignó además la Acción Enviar Mail, para poder comprobar su funcionamiento. Acciones de transformación, son asociadas posteriormente, para poder mostrar de forma más clara su funcionamiento.

6.5 Ejecución del caso de estudio

Una vez cargado el Modelo de Calidad y las Acciones al sistema, se ejecutaron algunas pruebas para validar el correcto funcionamiento del prototipo implementado, así como la utilidad de la solución brindada, para un caso de estudio semejante a un caso de la vida real. Para las siguientes pruebas, hay que destacar que la plataforma se configuró de forma tal que si se ejecuta una Acción Cancelar, el mensaje es cancelado luego de que se ejecuten todos los Métodos pendientes (ver sección 5.5.1).

A continuación, configurando el cabezal WS-Addressing correspondiente y dirigiendo el mensaje al *endpoint* que publica la plataforma, se realiza la invocación al servicio, donde ninguno de los elementos del *request* cumple con las Métricas del Modelo de Calidad configurado.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:con="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action>obtenerCertificado</wsa:Action>
    <wsa:From>
      <wsa:Address>uy:edu:fing:pgrado:dgreg</wsa:Address>
    </wsa:From>
    <wsa:MessageID>uuid:5b5f0d56-1d4f-4a28-8d15-a7e38b352cf8</wsa:MessageID>
    <wsa:To>uy:edu:fing:pgrado:msp:consultacertificadodefucion</wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <con:obtenerCertificado>
      <numeroCertificado>-</numeroCertificado>
      <tipoDocumento>cedula</tipoDocumento>
      <paisDocumento>URUGUAY</paisDocumento>
      <numeroDocumento/>
    </con:obtenerCertificado>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 6.2. Ejemplo de solicitud con errores.

```

<soap:Envelope xmlns:soap="https://www.w3.org/2001/06/soap-envelope">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>400</soap:faultcode>
      <soap:faultstring>El mensaje no cumple las restricciones de calidad</soap:faultstring>
      <soap:detail>
        <error>452 - Nulo: Controla valores nulos en operación obtenerCertificado</error>
        <error>1011 - FormatoPais: Controla el formato pais en la operacion obtenerCertificado</error>
        <error>1013 - ListaPais: Verifica que sea un pais valido</error>
        <error>1022 - FormatoNumero: El atributo debe contener sólo dígitos</error>
        <error>1023 - FormatoTipoDocumento: El tipo documento debe ser numérico</error>
        <error>1025 - ListaTipoDocumento : El tipo documento debe ser válido</error>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>

```

Figura 6.3. Ejemplo de respuesta de un mensaje cancelado.

En la Figura 6.2, se puede observar el mensaje de invocación y en la Figura 6.3 el mensaje de error obtenido como respuesta, generado por la plataforma al cancelar el mensaje. La respuesta es el resultado de la ejecución de las Acciones Cancelar de los Métodos asociados a la operación: *Nulo*, *FormatoPais*, *ListaPais*, *FormatoNumero*, *FormatoTipoDocumento* y *ListaTipoDocumento*.

Además, como resultado de la Acción de notificación por mail, asociada al Método *Nulo*, se recibió el mail presentado en la Figura 6.4.

```

El mensaje ha sido cancelado porque tiene campos en null
Fallo el request:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:con="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion"><soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing"/>
  <soapenv:Body>
    <con:obtenerCertificado>
      <numeroCertificado></numeroCertificado>
      <tipoDocumento>cedula</tipoDocumento>
      <paisDocumento>URUGUAY</paisDocumento>
      <numeroDocumento/>
    </con:obtenerCertificado>
  </soapenv:Body>
</soapenv:Envelope>
El siguiente metodo ha fallado:
Nulo

```

Figura 6.4. Mail recibido con detalles del error.

Luego se procede a editar los Métodos *ListaPais* y *FormatoPais*, quitando la Acción Cancelar y agregando la Acción Transformación XSL con el archivo transformacionPais.xsl que se observa en la Figura 6.5.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

  <xsl:template match="paisDocumento/text()[.='URUGUAY' or .='Uruguay' or .='uruguay' or .='UY']">URY</xsl:template>
  <xsl:template match="paisDocumento/text()[.='ARGENTINA' or .='Argentina' or .='argentina' or .='AR']">ARG</xsl:template>
  <xsl:template match="paisDocumento/text()[.='BRASIL' or .='Brasil' or .='brasil' or .='BR']">BRA</xsl:template>

</xsl:stylesheet>
```

Figura 6.5. Implementación de TransformacionPais.xsl.

La transformación toma el valor del elemento *paisDocumento* y en el caso que este contenga valores imprecisos, como por ejemplo “URUGUAY”, “Uruguay”, “uruguay” o “UY”, lo convierte al valor “URY”. De forma análoga, contempla casos similares para Argentina y Brasil, sustituyéndolo por el valor adecuado para el servicio, en este caso “ARG” o “BRA” respectivamente.

Para continuar, considerando los mensajes de error retornados por la plataforma, se cambian algunos valores y se realiza una nueva solicitud. La misma cumple satisfactoriamente con todos los Métodos del Modelo, excepto con *FormatoPais* (el largo de “URUGUAY” es mayor a 3). En la Figura 6.6 se puede observar la nueva solicitud realizada.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:con="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action>obtenerCertificado</wsa:Action>
    <wsa:From>
      <wsa:Address>uy:edu:fing:pgrado:dgrec</wsa:Address>
    </wsa:From>
    <wsa:MessageID>uuid:5b5f0d56-1d4f-4a28-8d15-a7e38b352cf8</wsa:MessageID>
    <wsa:To>uy:edu:fing:pgrado:mmp:consultacertificadodefucion</wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <con:obtenerCertificado>
      <numeroCertificado>1000</numeroCertificado>
      <tipoDocumento>68909</tipoDocumento>
      <paisDocumento>URUGUAY</paisDocumento>
      <numeroDocumento>44072453</numeroDocumento>
    </con:obtenerCertificado>
  </soapenv:Body>
</soapenv:Envelope>

```

Figura 6.6. Ejemplo de solicitud al servicio *obtenerCertificado*.

Al no cumplir con el Método FormatoPais, la Acción Transformación XSL es ejecutada sobre el mensaje original, cambiando el valor de DocumentoPais por "URY". De esta forma el mensaje modificado cumple con todos los Métodos pendientes, incluyendo ListaPais, y así se logra invocar el servicio satisfactoriamente, obteniéndose la respuesta que se muestra en la Figura 6.7.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:obtenerCertificadoResponse
      xmlns:ns2="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
      <return>
        <cedulaMedicoFirmante>42706507</cedulaMedicoFirmante>
        <cjppuMedicoFirmante>123</cjppuMedicoFirmante>
        <codigoDepartamento>MO</codigoDepartamento>
        <codigoEstablecimiento>201</codigoEstablecimiento>
        <codigoInstitucion>200</codigoInstitucion>
        <codigoLocalidad>MONTEVIDEO</codigoLocalidad>
        <direccionDefuncion>Soriano 1570</direccionDefuncion>
        <estadoCivil>C</estadoCivil>
        <fechaDefuncion>2016-04-18 08:50:10</fechaDefuncion>
        <fechaFirmaCertificado>2016-04-18 20:50:10</fechaFirmaCertificado>
        <fechaNacimiento>1974-07-08 00:00:00</fechaNacimiento>
        <lugarDefuncion>2</lugarDefuncion>
        <nacionalidad>URY</nacionalidad>
        <nombreEstablecimiento>ASSE-1</nombreEstablecimiento>
        <nombreInstitucion>ASSE</nombreInstitucion>
        <numeroCertificado>1000</numeroCertificado>
        <numeroDocumento>44072453</numeroDocumento>
        <paisDocumento>URY</paisDocumento>
        <primerApellido>Perez</primerApellido>
        <primerNombre>Jose</primerNombre>
        <sexo>1</sexo>
        <tipoDocumento>68909</tipoDocumento>
      </return>
    </ns2:obtenerCertificadoResponse>
  </soap:Body>
</soap:Envelope>

```

Figura 6.7. Respuesta a solicitud del servicio *obtenerCertificado*.

Para continuar con las pruebas, se realizó una nueva consulta, donde el servicio *obtenerCertificado* devuelve un Certificado vacío, para que se ejecuten los Métodos asociados a la respuesta de la operación, y a su vez se ejecuten las Acciones correspondientes a esos Métodos. En la Figura 6.8 se puede observar la invocación y en la Figura 6.9 la respuesta obtenida.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:con="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
  <soapenv:Body>
    <con:obtenerCertificado>
      <numeroCertificado>2003</numeroCertificado>
      <tipoDocumento>68909</tipoDocumento>
      <paisDocumento>URY</paisDocumento>
      <numeroDocumento>47072252</numeroDocumento>
    </con:obtenerCertificado>
  </soapenv:Body>
</soapenv:Envelope>

```

Figura 6.8. Ejemplo de invocación a la operación *obtenerCertificado*.

```

<soap:Envelope xmlns:soap="https://www.w3.org/2001/06/soap-envelope">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>400</soap:faultcode>
      <soap:faultstring>El mensaje no cumple las restricciones de calidad</soap:faultstring>
      <soap:detail>
        <error>1001 - Nulo: Controla valores nulos en operación obtenerCertificadoResponse</error>
        <error>1002 - FormatoFechaHora: Controla que las fechas tengan formato yyyy-MM-dd'T'HH:mm:ssZ</error>
        <error>1003 - FechaHoraMenorHoy: Verifica que la Fecha/Hora sea menor o igual a la Fecha/Hora de hoy.</error>
        <error>1004 - ListaInstitucion: Verifica si la institución se encuentra dentro de la lista de
          instituciones autorizadas.</error>
        <error>1005 - ListaPersonalSalud: Verifica que el personal de salud está habilitado</error>
        <error>1008 - ListaSexo: Verifica si el sexo se encuentra dentro de la lista de sexos definida en el Modelo.</error>
        <error>1009 - FormatoEstadoCivil : Verifica que el estado civil sea texto y de largo 1</error>
        <error>1010 - ListaEstadoCivil: El estado civil debe pertenecer al dominio de estados civiles válidos</error>
        <error>1012 - FormatoPais: Controla el formato pais en la operacion obtenerCertificadoResponse</error>
        <error>1014 - ListaPais: Verifica validez del pais, para obtenerCertificadoResponse</error>
        <error>1015 - FormatoDepartamento : Verifica si la variable tiene formato char(2) en opera</error>
        <error>1016 - ListaDepartamento: Verifica que el departamento sea de Uruguay según estandar ISO 3166-2</error>
        <error>1017 - VerificaDepartamentoPais: Verifica que si existe departamento es porque,
          para el mismo conjunto de datos, el país de referencia es Uruguay.</error>
        <error>1018 - FormatoLocalidad : La localidad debe tener formato texto</error>
        <error>1019 - FechaMenorHoy: La fecha debe ser menor o igual a la actual</error>
        <error>1020 - FormatoTexto: El nombre y el apellido deben tener formato texto</error>
        <error>1021 - FormatoNumero: Variables que sólo pueden tener dígitos</error>
        <error>1026 - ListaTipoDocumento : El tipo documento debe ser válido en el obtenerCertificadoResponse</error>
        <error>1027 - ListaEstablecimiento: El establecimiento debe estar habilitado</error>
        <error>1028 - ListaLugarDefuncion: El lugar de defunción debe estar dentro de la lista habilitada</error>
        <error>1029 - ListaInstitucion: La institución debe estar habilitada</error>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>

```

Figura 6.9. Mensaje de respuesta.

Como se puede observar en la Figura 6.9 el mensaje de respuesta es cancelado, generando un mensaje de error conteniendo el detalle de los errores generados por las Acciones Cancelar de los Métodos que no cumplieron las validaciones.

Para finalizar, se realiza una invocación al servicio obtenerListaCertificados, poniendo a prueba la capacidad de la solución de evaluar la calidad a una lista de elementos. En este caso, se devuelve una lista de *Certificados*, donde el campo *paisDocumento* de la mayoría de los *Certificados* tienen valores incorrectos, incumpliendo con el Método *FormatoPais* y/o *ListaPais*. Se editaron las Acciones de ambos Métodos, quitando la Acción Cancelar y agregando la Acción Transformación (que ejecuta la transformación *transformacionPais.xsl*), con la cual los valores del campo *paisDocumento* son transformados a sus equivalentes valores válidos.

En la Figura 6.10, se puede observar el mensaje de solicitud de la operación obtenerListaCertificados utilizada.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:con="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
  <soapenv:Header/>
  <soapenv:Body>
    <con:obtenerListaCertificados>
      <fechaInicio>2016-01-01T00:00:00</fechaInicio>
      <fechaFin>2016-12-31T23:59:59</fechaFin>
      <inicialInicio>A</inicialInicio>
      <inicialFin>Z</inicialFin>
    </con:obtenerListaCertificados>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 6.10. Ejemplo de solicitud a la operación *obtenerListaCertificados*.

Invocando el servicio web publicado por el MSP directamente, (sin utilizar la plataforma de calidad) se obtiene la respuesta que se puede ver en la Figura 6.11. Observándola en detalle, se puede apreciar que los valores del campo *paisDocumento*, en los Certificados devueltos, no pertenecen al estándar ISO 3166-1 Alpha3.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:obtenerListaCertificadosResponse
      xmlns:ns2="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
      <return>
        <certificados>
          <numeroCertificado>2000</numeroCertificado>
          <numeroDocumento>4270650</numeroDocumento>
          <paisDocumento>URUGUAY</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>4854</numeroCertificado>
          <numeroDocumento/>
          <paisDocumento>Argentina</paisDocumento>
          <tipoDocumento>68945</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>321456</numeroCertificado>
          <numeroDocumento>42741349</numeroDocumento>
          <paisDocumento>brasil</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>147744</numeroCertificado>
          <numeroDocumento>51361576</numeroDocumento>
          <paisDocumento>uruguay</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>98547</numeroCertificado>
          <numeroDocumento>12972803</numeroDocumento>
          <paisDocumento>UY</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
      </return>
    </ns2:obtenerListaCertificadosResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

Figura 6.11. Respuesta de la operación *obtenerListaCertificados* invocado directamente.

Al realizar la invocación al servicio nuevamente, en este caso a través de la plataforma, se obtiene la respuesta mostrada en la Figura 6.12.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:obtenerListaCertificadosResponse
      xmlns:ns2="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">
      <return>
        <certificados>
          <numeroCertificado>2000</numeroCertificado>
          <numeroDocumento>4270650</numeroDocumento>
          <paisDocumento>URY</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>4854</numeroCertificado>
          <numeroDocumento/>
          <paisDocumento>ARG</paisDocumento>
          <tipoDocumento>68945</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>321456</numeroCertificado>
          <numeroDocumento>42741349</numeroDocumento>
          <paisDocumento>BRA</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>147744</numeroCertificado>
          <numeroDocumento>51361576</numeroDocumento>
          <paisDocumento>URY</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
        <certificados>
          <numeroCertificado>98547</numeroCertificado>
          <numeroDocumento>12972803</numeroDocumento>
          <paisDocumento>URY</paisDocumento>
          <tipoDocumento>68909</tipoDocumento>
        </certificados>
      </return>
    </ns2:obtenerListaCertificadosResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

Figura 6.12. Respuesta del servicio obtenerListaCertificados invocado por la plataforma

6.6 Pruebas de *performance*

Los objetivos de estas pruebas fueron medir los tiempos de respuesta de la solución implementada en relación al tiempo de respuesta si se consumiera directamente al servicio, sin pasar por los componentes de Calidad de Datos, y evaluar el comportamiento de la plataforma al estar sobrecargada.

Para la adaptación del Modelo de Calidad descrito anteriormente, se han cargado 7 Métodos para el control de los elementos de entrada de la operación *obtenerCertificado*, y 27 para los elementos de respuesta de la misma. A su vez, estos se corresponden a 9 reglas de entrada y 55 de salida.

6.6.1 Overhead generado por la plataforma

La primera de las pruebas consistió en ejecutar cinco tipos de invocaciones al servicio *obtenerCertificado*, una directamente al servicio publicado y cuatro de ellas por medio de la plataforma de calidad: una sin Métodos configurados, otra sin errores en la evaluación de los Métodos, otra con valores incorrectos en los datos de entrada para provocar errores en los *requests* y otra con valores correctos a la entrada pero incorrectos en la respuesta del servicio, para provocar errores en los *responses*.

Para esto se simularon cinco clientes ejecutando consultas al servicio web de forma concurrente, en intervalos de 1000 ms durante 60 segundos. Se registraron para cada una de las estrategias empleadas los tiempos de procesamiento, los cuales fueron utilizados para realizar una comparación con la invocación directa al servicio.

Los tiempos promedio y el *overhead* generado por la solución en cada caso, se pueden consultar en la Tabla 6.5.

Estrategia	Promedio (ms)	Overhead (ms)
Directo	7	-
Sin Métodos	125	118
Con errores en <i>request</i>	132	125
Sin errores	256	249
Con errores en <i>response</i>	268	261

Tabla 6.5. Resumen del *overhead* generado según dos estrategias.

De la anterior tabla se puede interpretar que el sólo hecho de pasar por la plataforma, como es el caso de la estrategia sin Métodos configurados, agrega un *overhead* aproximado de 118ms. Esto puede explicarse debido a que una vez que un mensaje ingresa a la plataforma, se agrega un tiempo fijo de procesamiento considerable debido a que se realizan operaciones de validación del formato y de la validez del mensaje de acuerdo a las configuraciones de la plataforma.

Luego puede observarse una diferencia entre las estrategias sin Métodos y con errores en el *request*, con respecto a las que no tienen errores o los errores son detectados en el *response*. Esto puede explicarse debido a que para la primera estrategia no se evalúan Métodos, y en la segunda se evalúan sólo 7, que son los aplicados a un mensaje *request* que consume ese servicio. Mientras que para las dos siguientes estrategias se evalúan los 34 Métodos que aplican a los elementos del *request* o a los de *response*, además de agregarse la invocación al servicio final.

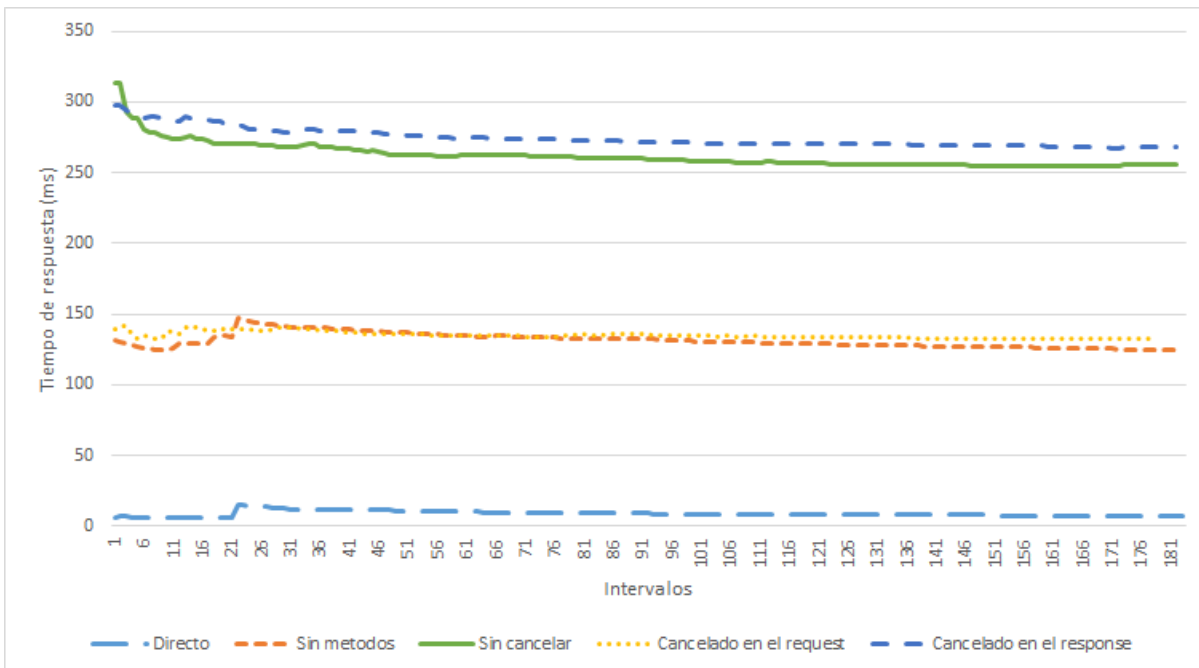


Figura 6.13. Resumen del *overhead* generado en la primera prueba.

En la Figura 6.13 se puede apreciar gran similitud en cuanto a los tiempos de respuesta de los mensajes no cancelados y los cancelados en el *response*. Esto permite apreciar que el hecho de cancelar un mensaje prácticamente no implica *overhead* ante igual cantidad de Métodos evaluados. Se puede observar también que el tiempo agregado por la plataforma ronda los 260 ms, cuando se evalúa una cantidad de 34 Métodos, tres de ellos resueltos mediante Métodos personalizados que son cargados por Reflection.

6.6.2 Prueba de stress

La segunda prueba realizada fue una prueba de sobrecarga, enviando ráfagas de 30 pedidos en 5 segundos, a intervalos de 5 segundos, durante un minuto. Esto significó un total de 420 mensajes por minuto, y se obtuvo un promedio de respuesta de 2096 ms por mensaje.

En la Figura 6.14 se puede observar la evolución en el promedio de los tiempos de respuesta, que llega a un máximo de 3051 ms, y un mínimo de 2079 ms.

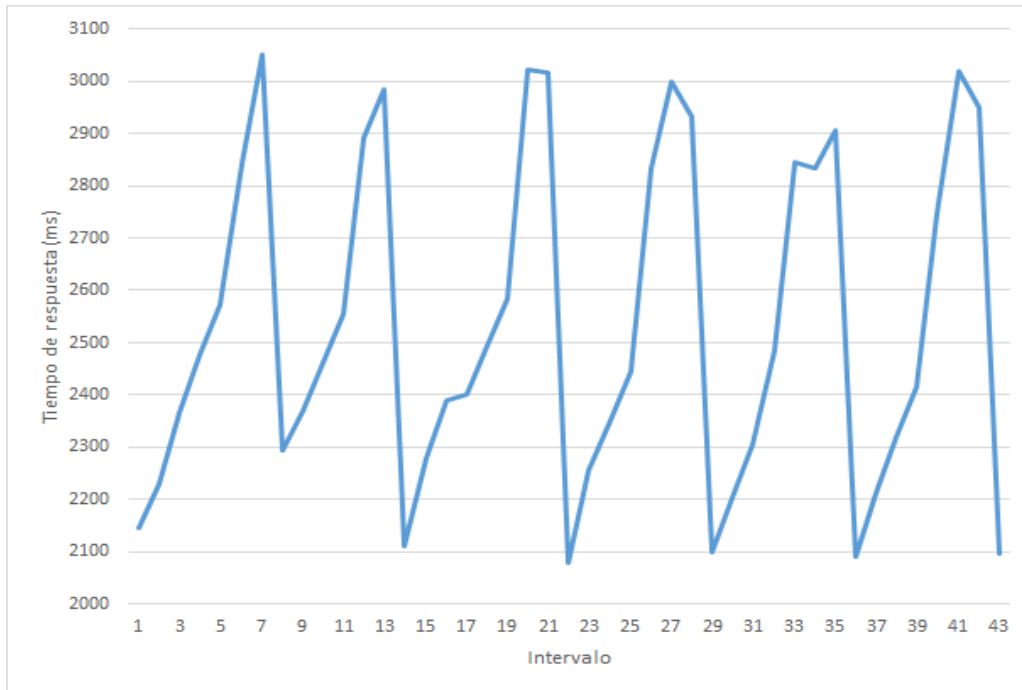


Figura 6.14. Promedio de tiempos de respuesta obtenidos en las pruebas por ráfagas.

En la anterior figura puede observarse cómo el promedio aumenta a medida que se envía una ráfaga de solicitudes, y disminuye en el tiempo en que la siguiente ráfaga comienza. Esto puede explicarse debido a que cuando la ráfaga comienza se dan los tiempos de respuesta más rápidos por estar el servidor menos sobrecargado, pero a medida que las solicitudes se van procesando, el tiempo de respuesta empeora cada vez más, hasta que la ráfaga termina.

En la Figura 6.15, pueden observarse los tiempos de respuestas para ráfagas de pedidos similares, variando la cantidad de solicitudes simultáneas:

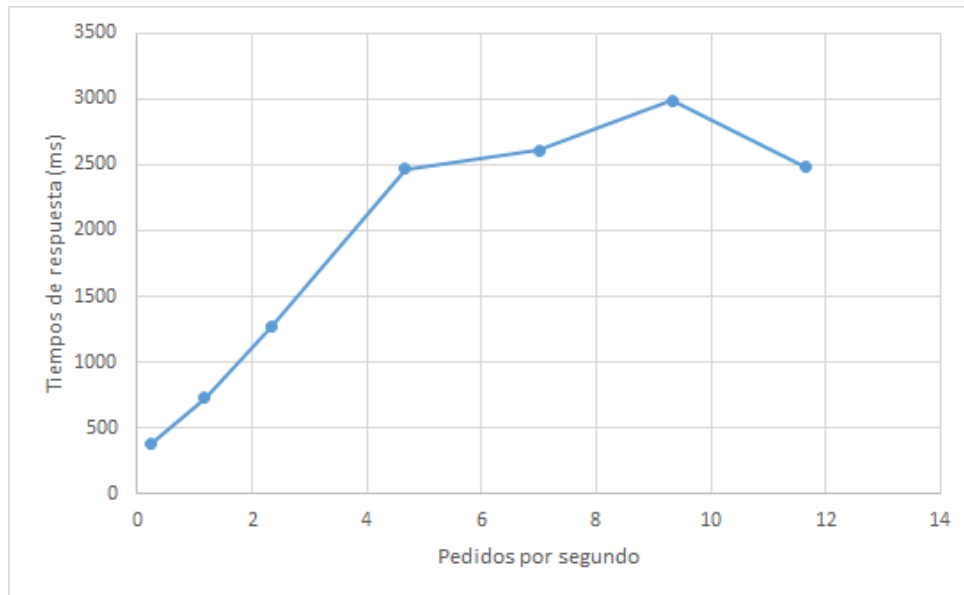


Figura 6.15. Tiempos de respuesta en pruebas de pedidos por ráfaga.

En la Figura 6.15 puede observarse la evolución en los tiempos de respuesta para las pruebas en ráfaga a medida que se modifica la cantidad de pedidos enviados por ráfaga. Se puede ver que a pesar de que los tiempos de respuesta aumentan, lo hacen de forma moderada.

6.6.3 Overhead por Métodos, Reglas y Transformaciones

Para esta sección se realizaron una serie de pruebas para determinar cuánto tiempo de ejecución conlleva el uso de Métodos, Reglas y Transformaciones. Para esto se formularon casos de testing representativos, para cada situación.

Se realizaron 9 pruebas, variando entre ellas la cantidad de Reglas, Métodos y Transformaciones que cada una ejecuta, en donde las cantidades son 1, 15 y 30.

Para las pruebas con Reglas, se eligió realizar una validación básica del formato de un campo, utilizando un mismo Método y variando la cantidad de Reglas que este evalúa. De esta manera se puede apreciar, como se ve en la Figura 6.16, que las Reglas no agregan demasiado tiempo de procesamiento en cuanto a la validación de los mensajes.

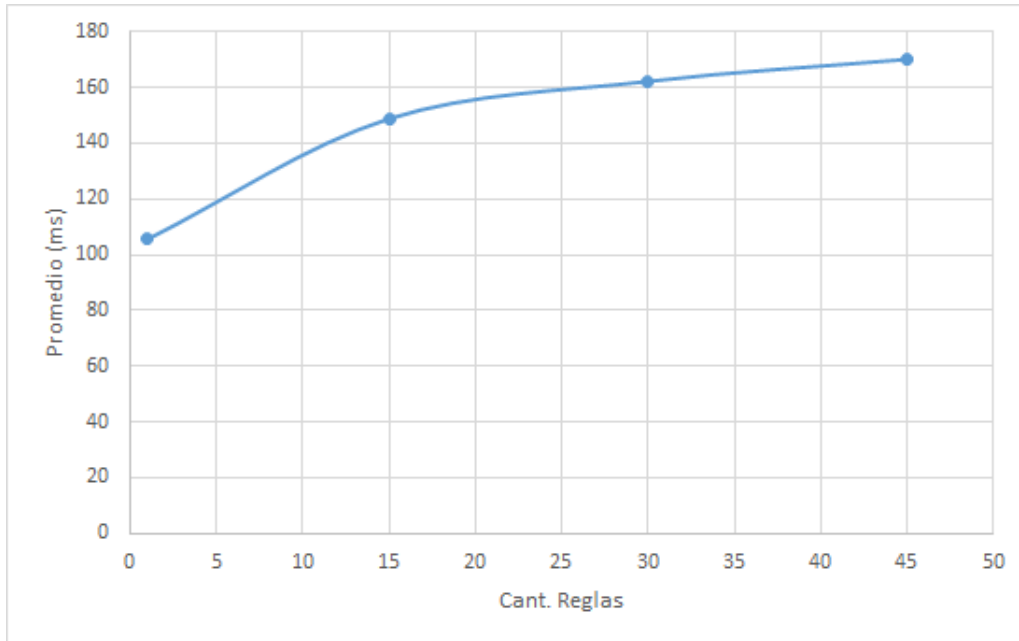


Figura 6.16. Resumen del *overhead* generado según la cantidad de reglas de un Método.

En la anterior figura también puede observarse cómo los tiempos de respuesta tienen un crecimiento menor a medida que se agregan más Reglas. Esto demuestra que la mayor parte del *overhead* generado en la evaluación de Reglas no recae en la cantidad de las mismas, sino en el hecho de que un Método deba ejecutar al menos una de ellas.

Por otra parte, el incremento de Métodos, en donde todos ellos contienen una única Regla que los compone, aumenta considerablemente la cantidad de tiempo que requieren para ser procesados. Este aumento se debe a la forma en cómo se procesan los Métodos, ya que deben ejecutarse en forma serializada para poder ejecutarlas en el orden en que fueron ingresadas al sistema. De esta forma se puede contemplar la posibilidad de aplicar Acciones sobre un mensaje al momento de que cierto Método no se cumpla, sin la necesidad de ejecutar los siguientes.

En cuanto a las transformaciones, se utilizó para las pruebas la transformación *Identidad* (ver en Apéndice 3), que no modifica el mensaje original. Cabe destacar que el tiempo aislado agregado por las transformaciones se debe calcular dejando de lado lo que agrega cada Método de por sí.

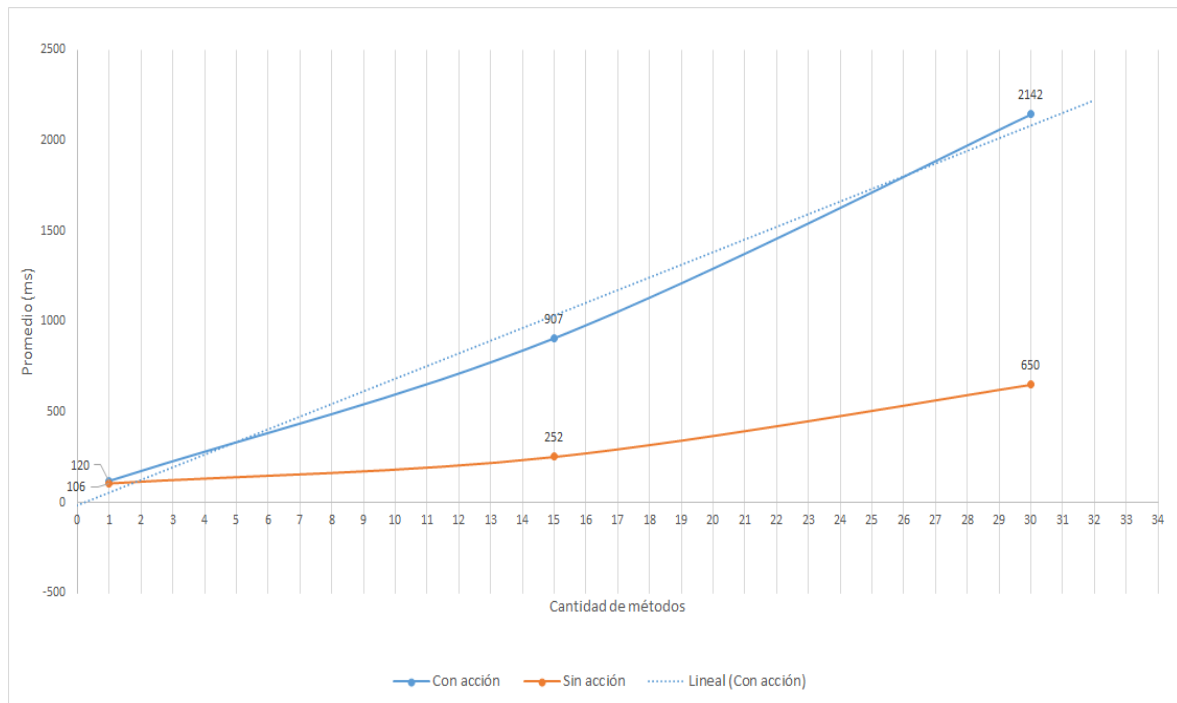


Figura 6.17. Resumen del *overhead* generado según Métodos y Acciones.

Analizando la Tabla 6.6 y la Figura 6.17, con el resumen del tiempo registrado en las pruebas, se puede observar que la relación entre la cantidad de los Métodos y el tiempo promedio está por debajo de lo lineal para una cantidad razonable de Métodos, ya que en realidad se puede observar que el crecimiento es polinómico, pero de un grado muy bajo. De la misma forma influye la ejecución de Acciones. Estas sobrecargas, de un modo, se ven justificadas debido al consumo de recursos limitados, como CPU para procesar la gran cantidad de Métodos y Transformaciones por segundo, debido a las condiciones de las pruebas.

Estrategia	Promedio (ms)
1 Método - 0 Acción - 0 Transformación	76.8
1 Método - 1 Acción - 0 Transformación	105.5
1 Método - 0 Acción - 1 Transformación	119.2
1 Método - 15 Acción - 0 Transformación	148.75
1 Método - 45 Acción - 0 Transformación	170.1
1 Método - 30 Acción - 0 Transformación	162.14
15 Método - 1 Acción - 0 Transformación	251.9
30 Método - 0 Acción - 0 Transformación	650.1
15 Método - 0 Acción - 15 Transformación	906.9
30 Método - 0 Acción - 30 Transformación	2142.4

Tabla 6.6. *Overhead* generado según las estrategias.

6.7 Conclusiones

Al ejecutar las pruebas, se pueden diferenciar dos claros flujos principales por los cuales puede pasar un mensaje en la solución, siendo estos el monitoreo y la validación. El flujo de monitoreo no agrega ninguna lógica sobre el mensaje que llega a la solución, más que realizar un registro y re direccionar el mismo hacia el destino pertinente. Por otro lado, la validación requiere de la ejecución de diversos cálculos y evaluaciones, que determinan qué Métodos y Acciones deben ser invocados.

Con respecto al *overhead* generado por la solución, se puede observar que en relación a la demora obtenida por consumir el servicio directamente, o sea, sin pasar por la plataforma, es considerable, ya que el tiempo por sólo pasar por la plataforma es más de 15 veces mayor al de consumir el servicio directamente. No obstante, se puede relativizar este incremento en el tiempo de respuesta de los pedidos que pasan por la plataforma, con el hecho de que las tareas realizadas dentro de la misma en este caso son de mayor complejidad que las tareas realizadas por el servicio final, que para las pruebas se encuentran simplificados totalmente. Por lo tanto, un incremento de algunos cientos de milisegundos, incluso de algunos segundos, sería aceptable en ambientes reales.

Por otra parte, se puede observar que se genera un crecimiento con un exponente tan bajo que se necesita un gran número de Métodos/Acciones (más de 30 para un mensaje) para que el tiempo de respuesta supere la franja de lo lineal. El *overhead* es generado mayormente

por el hecho de tener que ejecutar Métodos y Acciones, más que por la cantidad de los mismos, al involucrar los componentes que se encargan de realizar dichas tareas en el flujo del mensaje.

Con respecto a la capacidad de respuesta de la plataforma bajo *stress*, se puede observar que no existe un gran deterioro en los tiempos de respuesta de la misma. Por ejemplo, para cargas de dos pedidos por segundo se obtienen tiempos de respuesta que rondan el segundo y medio. Mientras tanto, para cargas de casi 12 pedidos por segundo, los tiempos de respuesta obtenidos rondan los 2.5 segundos, observándose un incremento por debajo de lo lineal.

Por último, cabe comentar que se notó que parte del *overhead* podía ser disminuido sensiblemente (hasta en un 20%) utilizando componentes switchyard fijos, y no invocándolos dinámicamente según el contexto o el contenido del mensaje. De todas formas, implementando una invocación dinámica de los mismos se da la flexibilidad de introducir nuevos componentes de forma más ágil que si estas invocaciones no pudieran ser configurables.

7. Conclusiones y Trabajo a Futuro

En este capítulo se presentan las conclusiones del trabajo realizado. En la sección 7.1 se presentan las conclusiones finales del trabajo y en la sección 7.2 se comentan posibles trabajos a futuro.

7.1 Conclusiones

En este proyecto, se realiza el análisis y diseño de un sistema que permite efectuar tareas pertinentes al control de calidad de los datos que se intercambian en la invocación de servicios entre organizaciones, a través de una plataforma ESB, utilizando y extendiendo las funcionalidades típicas que se pueden encontrar en un producto de esta índole. Además, se desarrollaron pruebas sobre un contexto reducido de una Plataforma de Salud, que demuestran que la solución implementada contribuye con la mejora de calidad de los datos que reciben y devuelven los servicios publicados.

Se analizó la problemática en base a dos temas fundamentales, por un lado, calidad de datos, y por otro, los mecanismos típicos brindados por plataformas ESB. En cuanto a calidad de datos se contempló la dificultad de gestionar múltiples Métodos asociados a los diferentes elementos de los servicios brindados a través de la plataforma. Para lo cual se requiere un complejo manejo del concepto de Método de un modelo de calidad, así como de sus relaciones con los elementos.

Con respecto a los mecanismos brindados por plataformas ESB, se pudo apreciar cuán aplicables son para resolver algunos de los problemas, en la medida que se analizaron los requerimientos. Por ejemplo el ruteo entre componentes y los patrones para modificar el mensaje transmitido a lo largo de la plataforma, con el objetivo de seguir un flujo de Acciones dinámicas según el mensaje y los modelos de calidad especificados. También fueron útiles para determinar la calidad del mensaje que se transporta, y, en caso de ser necesario, corregirlo de acuerdo a ciertas transformaciones de aseguramiento.

Dada la naturaleza dinámica de los datos y la forma en que se representan, la solución está pensada para que las tareas específicas del control y aseguramiento de la calidad de los mismos, puedan ser modificadas e incluso reemplazadas a lo largo del tiempo. Para esto se diseñó una arquitectura en la cual se tienen componentes principales para la estructura de la solución, encargados de manejar la lógica del encaminamiento de los mensajes a lo largo de la plataforma. Mientras que por otro lado, se tienen componentes cuya tarea es llevar a cabo la evaluación de la calidad de los datos que se procesan, o ejecutar Acciones según la calidad medida para los mismos. Estos últimos pueden ser modificados, agregados o removidos de manera sencilla y definida, sin alterar el correcto funcionamiento del sistema.

Dada la necesidad de facilitar la administración y configuración de la plataforma, se implementó una consola de administración que permite de forma gráfica configurar, administrar y monitorear los distintos aspectos de la misma.

Se presenta además un prototipo realizado en una plataforma ESB reconocida, como lo es Switchyard, que permite exponer la factibilidad de llevar a cabo la solución conceptual brindada con tecnologías *Open Source* disponibles en el mercado. Si bien Switchyard es una herramienta bastante completa, que cuenta con muchas funcionalidades interesantes, de acuerdo a la experiencia adquirida en el presente proyecto, algunos puntos aún pueden ser mejorados.

En la implementación del prototipo, se pudo apreciar claramente la dinámica y beneficios de las tecnologías de código abierto, basadas fuertemente en el soporte y contribuciones de la comunidad. En particular, uno de los problemas encontrados a la hora de la implementación se resolvió gracias a que la comunidad detectó la limitación de la plataforma, implementó la solución y actualizó la herramienta con las correcciones necesarias.

Con respecto a la *performance* obtenida en las pruebas realizadas sobre el prototipo, se puede apreciar que el *overhead* generado por la solución es aceptable, no siendo esto una limitante en cuanto a la experiencia de usuario una vez introducida la solución a la plataforma ESB.

Por último, se quiere resaltar que la implementación de la plataforma fue realizada desde un principio teniendo en cuenta un alto nivel de extensibilidad, lo que permite incorporar fácilmente nuevas funcionalidades.

Como conclusión general, el trabajo realizado demuestra que es viable la implementación de una plataforma donde se realice el control de calidad de los datos que fluyen por ella, siguiendo el diseño propuesto como solución y utilizando las capacidades de mediación brindadas por los ESB.

7.2 Trabajo a Futuro

A continuación se mencionan algunas limitaciones del trabajo realizado y se proponen una serie de posibles mejoras identificadas.

7.2.1 Extender la implementación del prototipo

Llevar la totalidad de la solución conceptual a una implementación más allá de un prototipo. Por ejemplo, generar Acciones con implementaciones particulares para usuarios avanzados, construir reglas más complejas que permitan el ingreso de múltiples parámetros, entre otros.

7.2.2 Mejorar Procesador de documentos WSDL en prototipo

El mecanismo desarrollado para procesar documentos WSDL presenta varias limitaciones. Se plantea como línea de trabajo a futuro extender la implementación actual de ese mecanismo, con el propósito de soportar aquellos casos que no son contemplados actualmente, detallados en la sección 5.7.1.

7.2.3 Reglas con diccionarios o listas en el prototipo

Permitir al usuario dar de alta Reglas que puedan acceder a una lista de valores externa, ya sea cargando un archivo o invocando un servicio web que los contenga, y no tener que ingresar estos valores uno por uno desde la interfaz web. Esto facilitaría en gran medida el ingreso de validaciones que manejen listas.

7.2.4 Mecanismos de alerta

Actualmente la plataforma registra información que posibilita una implementación de mensajes de alerta. Se podrían implementar por ejemplo, alertas que notifiquen la presencia de patrones o mensajes que no cumplen con modelos de calidad de forma muy recurrente. Además, las alertas podrían ser detectadas y enviadas de forma automática a los administradores de los distintos organismos.

7.2.5 Tipo de notificaciones

La solución dispone, como una de las Acciones posibles, el envío de emails para advertir sobre mensajes que no cumplen ciertos Métodos y por lo tanto modelos de calidad configurados. Se considera que podrían implementarse nuevos tipos o vías de notificaciones, como por ejemplo envíos de mensajes SMS, *push notifications* a dispositivos móviles, o notificaciones nativas del *Backoffice*.

7.2.6 Seguridad en *Backoffice*

En cuanto a temas de seguridad, sería indispensable encriptar con algún algoritmo seguro como SHA-2 ¹⁴ las contraseñas de los usuarios de la consola de administración.

Otra mejora, también para el componente web, sería incorporar el protocolo Hypertext Transfer Protocol Secure (HTTPS).

7.2.7 Agregar funcionalidades en *Backoffice*

Con el objetivo de mejorar y hacer más completo el *Backoffice* de administración, se identificó como trabajo a futuro la implementación de bajas y modificaciones de entidades, que quedó pendiente. Por ejemplo la eliminación de un organismo o de un Web Service.

¹⁴ SHA-2 es un conjunto de funciones hash criptográficas diseñadas por la Agencia de Seguridad Nacional (NSA).

Además se podrían agregar nuevas Métricas al *dashboard*, permitiendo generarlas de forma dinámica con filtros y parámetros, brindando la posibilidad de exportar los reportes como planillas de datos o archivos PDF.

7.2.8 Mejoras en ejecución de Métodos y Acciones

En cuanto al mecanismo de ejecución de Métodos y Acciones implementado en el prototipo, podría mejorarse permitiendo configurar el orden de los mismos para contemplar casos más complejos como por ejemplo, que se vuelvan a ejecutar los Métodos luego de una Acción de transformación.

8 Referencias

- [1] Mecella, M., Scannapieco, M., Virgillito, A., Baldoni, R., Catarci, T., Batini, C.: "Managing Data Quality in Cooperative Information Systems. On the Move to Meaningful Internet Systems". Springer, Londres, 2002.
- [2] González, L., Ruggia, R., "Towards a Compliance-Aware Inter-organizational Service Integration Platform". Springer, Berlin, 2014.
- [3] Kamyar Rasta, Trinh Hoang Nguyen, Andreas Prinz. "A Framework for Data Quality Handling in Enterprise Service Bus". Grimstad, Noruega, 2013
- [4] B. Otto, K. Hüner and H. Österle, "Identification of Business Oriented Data Quality Metrics", ICIQ'09, 2009.
- [5] "Data Quality and Systems Theory". Communications of the ACM, vol. 4, no. 2, 1998.
- [6] I. Caballero, "Calidad y Medición de Sistemas de Información." Universidad de Castilla La Mancha, España, Castilla de la Mancha, 2009.
- [7] Wang, R.; Strong, D.: "Beyond accuracy: What data quality means to data consumers". Journal on Management of Information Systems, Vol. 12, 4:5-34, 1996.
- [8] Bruno Bianchi Gallo, María Carolina Valverde Corrado. "Un caso de estudio en Calidad de Datos para Ingeniería de Software Empírica". Montevideo, Uruguay, 2009.
- [9] C. Batini and M. Scannapieco, "Data quality: concepts, methodologies and techniques". Springer, 2006.
- [10] Monica Scannapieco, "Data Quality under the Computer Science perspective", 2002.
- [11] L. L. Pipino, Y. W. Lee and R. Y. Wang, "Data quality assessment", Abril 2002.
- [12] Chappell, David. 2004. "Enterprise Service Bus: Theory in Practice". O'Reilly Media. <https://www.safaribooksonline.com/library/view/enterprise-service-bus/0596006756/ch01s07.html>. Último acceso: 06 de marzo de 2016.
- [13] M. P. a. W. Heuvel, "Service oriented architectures: approaches".
- [14] Patrón Dynamic Router - <http://camel.apache.org/dynamic-router.html>. Último acceso: 10 de mayo de 2016.
- [15] Tutorial Java EE. "What are Web Services?" - <https://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html>. Último acceso: 06 de marzo de 2016.

- [16] W3C Working Group Note. "Web Services Architecture" - <http://www.w3.org/TR/ws-arch/>. Último acceso: 06 de marzo de 2016.
- [17] W3C. "Guía Breve de Tecnologías XML" - <http://www.w3c.es/Divulgacion/GuiasBreves/TecnologiasXML>. Último acceso: 06 de marzo de 2016.
- [18] W3C. "Web Services Description Language 1.1" - http://www.w3.org/TR/wsdl#_introduction. Último acceso: 06 de marzo de 2016.
- [19] W3C. "Web Services Addressing" - <https://www.w3.org/Submission/ws-addressing>. Último acceso: marzo de 2016.
- [20] Tecnologías avanzadas para el desarrollo de Web Services. Introducción al Middleware, Facultad de Ingeniería, Universidad de la República - <https://eva.fing.edu.uy/mod/resource/view.php?id=41819>. Último acceso: 19 de marzo de 2016.
- [21] AGESIC. Servicio "Consulta del comprobante de defunción electrónico" - <http://www.agesic.gub.uy/innovaportal/v/1794/9/agesic/consulta-del-comprobante-de-defuncion-electronico.html>. Último acceso: 06 de marzo de 2016.
- [22] Andrés Echevarría, Dahiana Morales. "Protección de Datos Personales en Plataformas de Integración". Montevideo, Uruguay, 2015.
- [23] Laura González. "Plataforma ESB Adaptativa para Sistemas Basados en Servicios". Montevideo, Uruguay, 2011.
- [24] Jorge Luis Laborde, Mauricio Fenoglio y Matías Galnares. "Implementación de una Plataforma ESB Adaptativa". Montevideo, Uruguay, 2012.
- [25] Maven - <https://maven.apache.org/>. Último acceso: 13 de marzo de 2016.
- [26] Hibernate - <http://hibernate.org/>. Último acceso: 08 de marzo de 2016.
- [27] El lenguaje de consulta de Hibernate - <https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html>. Último acceso: 08 de marzo de 2016.
- [28] Hibernate Mapping Entities - https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/#entity. Último acceso: 08 de marzo de 2016.
- [29] PostgreSQL - <http://www.postgresql.org/>. Último acceso: 08 de marzo de 2016.

- [30] Switchyard - <http://switchyard.jboss.org/>. Último acceso: 08 de marzo de 2016.
- [31] Switchyard, guía de usuario - <https://docs.jboss.org/author/display/SWITCHYARD/User+Guide>. Último acceso: 08 de marzo de 2016.
- [32] Camel - <http://camel.apache.org/>. Último acceso: 08 de marzo de 2016.
- [33] Enterprise Integration Patterns - <http://camel.apache.org/enterprise-integration-patterns.html>. Último acceso: 08 de marzo de 2016.
- [34] Domain Specific Language - <http://camel.apache.org/dsl.html>. Último acceso: 08 de marzo de 2016.
- [35] SoapUI - <http://www.soapui.org/>. Último acceso: 08 de marzo de 2016.
- [36] W3C XSLT - <https://www.w3.org/TR/xslt>. Último acceso: 13 de marzo de 2016.
- [37] W3C XPATH - <https://www.w3.org/TR/xpath/>. Último acceso: 13 de marzo de 2016.
- [38] W3C SOAP Envelope - <https://www.w3.org/2001/06/soap-envelope>. Último acceso: 13 de marzo de 2016.
- [39] Foro de JBoss. Problema de invocación dinámica a servicios externos, desde Switchyard - <https://issues.jboss.org/browse/SWITCHYARD-1916>. Último acceso: 13 de marzo de 2016.
- [40] Jorge Abin, Horacio Nemeth, Ignacio Friedmann. "Arquitectura de Sistemas para un Sistema de Salud de alcance Nacional". Montevideo, 2015.

Apéndice 1. Registro para reportes

Las tablas A.1.1 y A.1.2 presentan y describen las tablas y datos almacenados para la creación de los reportes y estadísticas.

<i>RegistroMensaje</i>		
<i>Dato</i>	<i>Tipo</i>	<i>Descripción</i>
Request	Boolean	Toma el valor <i>true</i> si el mensaje es <i>request</i> y <i>false</i> si es <i>response</i> .
From	String	En caso de ser <i>request</i> , almacena la URN que identifica al organismo cliente del servicio. Si es <i>response</i> , almacena la URN que identifica al servicio web que está respondiendo.
ToUrn	String	En caso de ser <i>request</i> , almacena la URN que identifica al servicio web que se consume. Si es <i>response</i> , almacena la URN que identifica al organismo al que se envía la respuesta.
ToUrl	String	En caso de ser <i>request</i> , almacena el <i>endpoint</i> del servicio web que se consume. Si es <i>response</i> , queda vacío.
Action	String	Nombre de la operación del servicio web que se consume.
Message	String	Se almacena el Mensaje original en formato XML.
TimeStamp	Timestamp	Fecha y hora en que se recibe el mensaje en la plataforma.
Resultado	ResultadoMetodo	Representa un enumerado que toma el valor: EVALUA_OK si todos los Métodos de control de calidad aplicados al mensaje retornaron <i>true</i> . EVALUA_ERROR si al menos uno de los Métodos aplicados al mensaje evalúa <i>false</i> . NO_EVALUA si no existen Métodos para aplicar al mensaje, o estos no llegan a evaluarse porque el mensaje de entrada no cumple con los requisitos de la plataforma.
Valido	Boolean	Para el caso que el mensaje sea <i>request</i> , este campo toma el valor <i>true</i> si el mensaje contiene el cabezal WS-Addressing exigido y si existe el servicio web identificado con el <i>ToUrn</i> asociado al organismo identificado en el <i>From</i> . En caso contrario toma el valor <i>false</i> .

Tabla A1.1. Datos almacenados en la tabla RegistroMensaje.

<i>RegistroMetodo</i>		
<i>Dato</i>	<i>Tipo</i>	<i>Descripción</i>
Metodo	Long	Identificador del Método ejecutado.
Resultado	Boolean	Almacena el resultado del Método aplicado al mensaje.
RegistroMensaje	Long	Clave foránea del Registro Mensaje al cual está asociado el Método que se almacena.

Tabla A1.2. Datos almacenados en la tabla RegistroMetodo.

Apéndice 2. Inicialización de la estructura interna

A continuación se describe cómo deben ser inicializados los valores de la estructura interna para el intercambio de mensajes.

- **idRegistroMensaje:** una vez creada la estructura, se genera un registro del mensaje en la tabla de estadísticas. El identificador de ese registro es guardado en la estructura para que más adelante puedan actualizarse los datos de ese registro, por ejemplo con el resultado de ejecución de los Métodos.
- **toUrn:** cuando se recibe un mensaje *request*, se obtiene del cabezal WS-Addressing la URN del destino al que es enviado, y se guarda en este campo. Internamente en la plataforma, se manejan los datos del mensaje original (sin el cabezal) por eso es importante guardar hacia quién está dirigido el mensaje. Cuando se recibe una respuesta, se intercambia este valor de la estructura con el del campo *from*, y viceversa.
- **toUrl:** es utilizado en los mensajes *request*. Según el destino indicado en *toUrn*, se obtiene la URL del servicio (mapeada en la base de datos) y se guarda en este campo, para que luego el componente encargado de consumir el servicio sepa a qué dirección enviar el mensaje.
- **from:** al igual que el campo *toUrn*, se obtiene del cabezal WS-Addressing original de un *request*. Si el mensaje es un *response*, entonces se obtiene del campo *toUrn* del *request* al que corresponde. Este campo sirve para registrar quién es el cliente que invoca el servicio, en un *request*, o cuál es el servicio que responde, en un *response*.
- **isRequestMessage:** este campo toma los valores *true* o *false* según si el mensaje manejado es un *request* o un *response*, respectivamente. Es necesario para diferenciar si luego de procesado, un mensaje debe ser dirigido al componente que consume el servicio, o devuelto para ser enviado como respuesta al cliente.
- **method:** este campo puede aparecer varias veces en una estructura, y sirve para guardar los identificadores de los Métodos que deben ser ejecutados para el mensaje que se está procesando.
- **action:** de forma parecida al campo anterior, y con objetivos similares, almacena la información de las Acciones que deben ser ejecutadas luego de que un Método no se cumple. Son quitados a medida que las Acciones asociadas se van ejecutando.
- **error:** este campo es usado para guardar información cuando surge un error en el procesamiento del mensaje a lo largo de la plataforma, y luego devolver el mensaje correspondiente al cliente.
- **finish:** toma valores *true* o *false*, determinando si luego de la ejecución de los Métodos y Acciones, se debe continuar con el flujo del mensaje, o si este debe ser cancelado.
- **methodFail:** también toma valores *true* o *false*, y se actualiza luego de la ejecución de cada Método. Según este campo se sabe si el último Método fue evaluado de forma

positiva o negativa, y por consiguiente, si se deben cargar las Acciones necesarias en la estructura, o proseguir con la evaluación de los Métodos restantes.

- **body:** contiene el mensaje sin el cabezal WS-Addressing. El contenido puede ser modificado por las Acciones de aseguramiento que se ejecuten (por ejemplo, transformaciones XSL). Luego de finalizado el procesamiento de un mensaje, si no se debe retornar un error, se envía el contenido de este campo, ya sea para consumir el Web Service destino, si es un mensaje *request*, o para enviar como respuesta al cliente original, en caso de ser un *response*.

Apéndice 3. Transformaciones XSL

En las pruebas realizadas al prototipo se utilizó la transformación *identidad.xsl*. La misma reproduce su entrada como su salida. Por un lado puede parecer una transformación poco útil, pero la identidad proporciona la base de toda una clase de transformaciones útiles.

La Figura A2.1 presenta una implementación recursiva típica utilizando el Método `xsl:copy`.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

Figura A2.1. Transformación *identidad.xsl*.

A partir de la transformación presentada anteriormente se implementó la transformación de la Figura A2.2 para el servicio ObtenerCertificado presentado en el caso de estudio. La transformación mantiene todos los elementos del mensaje incambiados, menos el elemento `sexo`, que dependiendo de su valor, si no es ninguno de los valores válidos (definidos en la Tabla de Sexo del Apéndice 12), se le asigna el valor 4 que representa el valor "Se Desconoce". De esta forma se asegura que el valor del campo `sexo` siempre sea válido.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

  <xsl:template match="sexo/text() [!='1'] [!='2'] [!='3'] [!='4'] [!='9']">4</xsl:template>
</xsl:stylesheet>
```

Figura A2.2. Transformación *aseguramientoSexo.xsl*.

La transformación de la Figura A2.3 realiza una corrección sobre los mensajes que pertenecen al Método *obtenerCertificadoResponse*. Primero, verifica que exista el campo *estadoCivil*, y en caso de que no exista, lo agrega al cuerpo del mensaje, ingresando el valor por defecto "I", que representa el valor "Se Desconoce" según la Tabla de Estados Civiles definida en el Apéndice 12. En caso de que el campo ya se encuentre en el mensaje, la transformación no altera su contenido.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:web="http://www.fing.edu.uy/pgrado/services/ConsultaCertificadoDefuncion">

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

  <xsl:template match="web:obtenerCertificadoResponse">
    <xsl:copy>
      <xsl:apply-templates/>
      <xsl:if test="not(estadoCivil)">
        <estadoCivil>I</estadoCivil>
      </xsl:if>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Figura A2.3. Transformación *aseguramientoEstadoCivil.xsl*.

Apéndice 4. Manejo de errores

A continuación se pueden observar algunos ejemplos de mensajes de error devueltos por la plataforma. Primero se devuelve un mensaje de error indicando que no se cumplen las restricciones de calidad, y un detalle que contiene código y descripción de cada uno de los Métodos que no se cumplieron.

```
<soap:Envelope xmlns:soap="https://www.w3.org/2001/06/soap-envelope">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>400</soap:faultcode>
      <soap:faultstring>El mensaje no cumple las restricciones de calidad</soap:faultstring>
      <soap:detail>
        <error>13251 - Fecha de nacimiento invalida</error>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figura A3.1. Request que no cumple con las restricciones de calidad impuestas para ese servicio.

Luego la salida para un mensaje que no contiene los cabecales WS-Addressing requeridos por la plataforma:

```
<soap:Envelope xmlns:soap="https://www.w3.org/2001/06/soap-envelope">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>400</soap:faultcode>
      <soap:faultstring>Error en el cabecal WS-Addressing</soap:faultstring>
      <soap:detail>
        <error>11 - El cabecal addressing no contiene el campo "From"</error>
        <error>12 - El cabecal addressing no contiene el campo "To"</error>
        <error>13 - El cabecal addressing no contiene el campo "Action"</error>
        <error>14 - El cabecal addressing no contiene el campo "MessageID"</error>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figura A3.2. Mensaje de error indicando que el cabecal WS-Addressing es incorrecto y, en el detalle, cuáles son los errores que contiene.

Por último una solicitud que intenta acceder con parámetros incorrectos, por ejemplo: un destinatario no existente o inhabilitado en la plataforma:

```
<soap:Envelope xmlns:soap="https://www.w3.org/2001/06/soap-envelope">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>400</soap:faultcode>
      <soap:faultstring>Los parametros enviados no son validos para la plataforma.
      Comuniquese con el administrador del sistema</soap:faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figura A3.3. Mensaje de error para una solicitud que intenta acceder a un organismo no habilitado en la plataforma.

Apéndice 5. Métodos

En esta sección se presentan reglas implementadas para el prototipo. Estas fueron pensadas para abarcar una cantidad de casos de uso básicos que pueden presentarse al momento de querer realizar validaciones sobre los mensajes.

La implementación que se llevó a cabo apunta a que la tarea de agregar nuevas reglas por medio de código sea sencilla. Para esto, se implementó una clase *Validator*, la cual contiene una función que busca evaluar las reglas de la forma más genérica posible.

Teniendo en cuenta que una gran mayoría de las evaluaciones de Métodos se realizan bajo la comparación de 2 elementos, cualesquiera sean estos, se desarrolló una función genérica que toma como parámetros dos elementos cualesquiera, y un enumerado que representa el tipo de regla a aplicar. Por ejemplo, `validate(<nombre>, "Carlos", EQUALS)`, donde `<nombre>` es el valor del elemento *nombre* en el mensaje.

En la Tabla A5.1 se presentan las operaciones separadas por tipos.

String		
Identificador	Tipo de Comparación	Descripción
LARGER_THAN	String x, String y	Largo de x mayor que largo de y
SHORTER_THAN	String x, String y	Largo de x menor que largo de y
EQUALS_LENGTH	String x, String y	Largo de x igual que largo de y
EQUALS	String x, String y	x es igual a y
NOT_EQUALS	String x, String y	x no es igual a y
SUB_STRING	String x, String y	y está contenido como <i>substring</i> en x
REG_EXP	String x, String y	x concuerda con la expresión regular definida en y
CONTANED_S	String x, [String] y	x está contenido en y
LENGTH_LARGER_THAN	String x, Integer y	Largo de x es mayor que y
LENGTH_SHORTER_THAN	String x, Integer y	Largo de x es mayor que y
LENGTH_EQUALS_THAN	String x, Integer y	Largo de x es igual que y
TEXT_FORMAT	String x	x está compuesto solo por caracteres alfabéticos

Tabla A5.1 Operaciones implementadas para el tipo *String*.

Comparable		
Identificador	Tipo de Comparación	Descripción
GREATER	T x, T y (*)	x es mayor que y
GREATER_E	T x, T y	x es mayor o igual que y
LESS	T x, T y	x es menor que y
LESS_E	T x, T y	x es menor o igual que y
EQUALS	T x, T y	x es igual a y
NOT_EQUALS	T x, T y	x es distinto de y
CONTAINS	T x, [T] y	x está contenida en y
NUMBER_FORMAT	Number x	x está compuesto solo por caracteres numéricos

Tabla A5.2. Operaciones implementadas para las clases que extienden de *Comparable*.(*) El tipo genérico T es cualquier clase que extienda de la clase *Comparable*.

Date		
Identificador	Tipo de Comparación	Descripción
BEFORE_TODAY	Date x	La fecha x es anterior a hoy
RECENT_DATE	Date x	El día de la fecha x es de ayer u hoy.
DATE_FORMAT	Date x, String y (opcional)	La fecha x cumple con el formato opcional y, o cumple con el formato genérico para mensajes SOAP

Tabla A5.3. Operaciones implementadas para el tipo Date.

Genérica		
Identificador	Tipo de Comparación	Descripción
EXISTS	Object a	a es distinto de null

Tabla A5.4. Operación que chequea si un elemento específico existe.

Apéndice 6. Implementación de una Acción

A continuación se puede observar un segmento de código del componente *ActionMail*, utilizado para ejecutar las Acciones de tipo *AccionMail*, que se encargan de enviar un mail a una serie de destinatarios definidos en la Acción, con un asunto y cuerpo configurables para cada una.

Este componente puede modificar los campos *body*, *finish* y *errores* del mensaje recibido. El *body*, para el caso de transformaciones que se deseen realizar, la bandera *finish* en caso de que sea una Acción de cancelamiento, para indicar a la plataforma que el mensaje debe ser descartado, y la lista *errores* para agregar los mensajes de error que se deseen mostrar, en caso de que suceda alguno, o, por ejemplo, se deba cancelar el mensaje.



Figura A6.1. Segmento de código de la clase *ActionMail*, implementación del componente de ejecución de las Acciones de tipo *AccionMail*.

Apéndice 7. Ejemplo de Método por Reflection

A continuación puede observarse el ejemplo de un código personalizado para resolver un Método que controla que las cédulas de un mensaje tengan el formato válido según el algoritmo de cédula uruguaya. Luego de compilado, debe ser colocado en el *path* indicado para los Métodos a ser utilizados por Reflection, y se debe asociar el Método que lo debe utilizar, a la clase `ControlarCedula`, para que el componente `ReflectionDispatcher` pueda invocarlo correctamente.

```
package uy.edu.fing.pgrado.CalidadDeDatosESB.ReflectionMethods;

public class ControlarCedula {

    public boolean resolveMethod(String body){
        boolean cedulasValidas = true;
        ArrayList<String> cedulas = obtenerCedulas(body);
        for (int i = 0; i < cedulas.size(); i++) {
            String cedula = cedulas.get(i);
            cedulasValidas &= validarCI(cedulaVal);
        }

        return cedulasValidas;
    }

    private ArrayList<String> obtenerCedulas(String body){
        ...
    }

    private boolean validarCI(String cedula){
        ...
    }
}
```

Figura A7.1. Ejemplo de implementación de un Método a ser invocado por *Reflection*.

Apéndice 8. Operaciones utilizadas en el caso de estudio

ObtenerCertificado

A continuación se detalla la primera de las operaciones utilizadas para validar el prototipo, comenzando por una breve descripción, continuando con los parámetros de entrada y salida esperados.

Descripción

Retorna el certificado resumido correspondiente con el número o documento del fallecido pasado por parámetro.

Parámetros de entrada

En la Tabla A8.1 se puede observar el detalle de los parámetros de entrada.

Parámetros	Tipo de dato	Obligatorio	Valor por defecto	Descripción
obtenerCertificado				
numeroCertificado	String	NO	N/A	Número del certificado
tipoDocumento	Integer	NO	N/A	Tipo Documento del Fallecido (Tabla A12.3 de Tipo de Documento)
paisDocumento	String	NO	N/A	País del Documento del Fallecido (Código Alpha3 ISO 3166-1)
numeroDocumento	String	NO	N/A	Número del Documento del Fallecido

Tabla A8.1. Detalle de la operación *obtenerCertificado*.

Parámetros de salida

En la Tabla A8.2 se detallan los elementos de salida del servicio.

Parámetros	Tipo de dato	Descripción
obtenerCertificadoResponse		
numeroCertificado	String	Número del certificado
tipoDocumento	Integer	Tipo Documento del Fallecido (ver Tabla A12.3 de Tipo de Documento)
paisDocumento	String	País del Documento del Fallecido (Código Alpha3 ISO 3166-1)
numeroDocumento	String	Número del Documento del Fallecido
primerNombre	String	Primer Nombre difunto
primerApellido	String	Primer Apellido difunto
lugarDefuncion	Integer	Indica el lugar de defunción (ver Tabla A12.4 de Lugar de Defunción)
codigoInstitucion	Integer	Código de Institución de salud según registro del MSP
nombreInstitucion	String	Nombre de Institución de salud según registro del MSP
codigoEstablecimiento	Integer	Código de Establecimiento de salud según registro del MSP
nombreEstablecimiento	String	Nombre de Establecimiento de salud según registro del MSP
codigoDepartamento	String	Código de Departamento de defunción (Código Alpha2 según ISO 3166-2)
codigoLocalidad	String	Código de Localidad de defunción (Código Alpha7, referencia al código de centros poblados según el criterio de codificación del SGM)
direccionDefuncion	String	Dirección de defunción en caso de que la defunción no ocurra en el centro asistencial (calle, número, ruta, km, etc)
estadoCivil	String	Código de Estado Civil del Fallecido (ver Tabla A12.2 de Estado Civil)

nacionalidad	String	País de Nacimiento del Fallecido (Código Alpha3 ISO 3166-1)
fechaDefuncion	Date	Fecha en formato “yyyy-MM-dd HH:mm:ss”
fechaNacimiento	Date	Fecha en formato “yyyy-MM-dd HH:mm:ss”
sexo	Integer	Sexo fallecido (ver Tabla A12.1 de Sexo)
fechaFirmaCertificado	Date	Fecha en formato “yyyy-MM-dd HH:mm:ss”
cedulaMedicoFirmante	String	Cédula de identidad del médico firmante
cjppuMedicoFirmante	String	Número de caja profesional del médico firmante

Tabla A8.2. Detalle de la operación *obtenerCertificadoResponse*.

ObtenerListaCertificados

A continuación se detalla la segunda de las operaciones utilizadas para validar el prototipo, comenzando por una breve descripción, continuando con los parámetros de entrada y salida esperados.

Descripción

Retorna una lista de certificados de defunción con Fecha de Defunción comprendida entre las fechas pasadas por parámetro y con la inicial del Primer Apellido del Fallecido comprendido entre las iniciales pasadas por parámetro.

Parámetros de entrada

En Tabla A8.3 se muestran los parámetros de entrada.

Parámetros	Tipo de dato	Obligatorio	Valor por defecto	Descripción
obtenerListaCertificados				
fechaInicio	Date	SI	N/A	Fecha Inicio de Defunción
fechaFin	Date	SI	N/A	Fecha de Fin de Defunción
inicialInicio	String	NO	A	Inicial Primer Apellido Fallecido Inicio
inicialFin	String	NO	Z	Inicial Primer Apellido Fallecido Fin

Tabla A8.3. Detalle de la operación *obtenerListaCertificados*.

Parámetros de salida

En la Tabla A8.4 se detallan los elementos de salida del servicio. Puede observarse que se tiene una lista de elementos complejos del tipo “certificado”, compuestos por un conjunto de elementos simples.

Parámetros	Tipo de dato	Descripción
obtenerListaCertificadosResponse		
certificados	List<certificado>	Colección de certificados
certificado		
numeroCertificado	String	Número del certificado
tipoDocumento	Integer	Tipo Documento del Fallecido (ver Tabla A12.3 de Tipo de Documento)
paisDocumento	String	País del Documento del Fallecido (Código Alpha3 ISO 3166-1)
numeroDocumento	String	Número del Documento del Fallecido

Tabla A8.4. Detalle de la operación *obtenerListaCertificadosResponse*.

Apéndice 9. Propiedades globales

A continuación se presentan algunas de las propiedades utilizadas en el prototipo, definidas en archivo “calidaddedatos.properties”.

Email

La cuenta de email desde la cual se desea enviar las notificaciones a los usuarios pertinentes (debiendo especificar también el *password* de la cuenta, el *host* y el puerto por el cual se realizarán los envíos).

Modo de cancelación de los mensajes

Se pueden configurar dos formas de cancelar los mensajes cuando se ejecuta una Acción-Cancelar. Estas son: cancelar inmediatamente o cancelar al final. La primera puede resultar de utilidad en casos de tener Métodos que de antemano se sabe que no serán necesarios evaluar, evitando ejecutar operaciones innecesarias. La segunda puede ser de utilidad para ejecutar todos los Métodos antes de cancelar, pudiendo recibir de esta forma, la notificación de todos los Métodos que no validan el mensaje.

Rutas de archivos externos al sistema

Algunos componentes del sistema hacen uso de archivos alojados fuera del mismo, para lo cual es necesario definirles una ruta. Algunos de estos son los archivos de transformaciones para el aseguramiento de calidad, las clases de validación de calidad implementadas por el usuario, o las Acciones de la misma índole.

Apéndice 10. Métricas del caso de estudio

A continuación se listan todas las Métricas que son empleadas en el caso de estudio, agrupando las mismas por el tipo de dato al que se aplica.

Dimensión	Factor	Métrica
Fecha Evento		
Exactitud	Correctitud Sintáctica	FormatoFechaHora() Verifica si la variable tiene formato “yyyy-MM-dd HH:mm:ss” Tipo de Resultado: (0,1) Granularidad: elemento
Exactitud	Correctitud Semántica	FechaHoraMenorHoy() Verifica que la Fecha sea menor a la Fecha de hoy. Tipo de Resultado: (0,1) Granularidad: elemento
Frescura	Oportunidad	RangoFechaHora() Verifica si la Fecha/Hora es reciente para determinar si el dato fue oportuno. Tipo de Resultado: (0,1) Granularidad: elemento
Institución de Salud		
Consistencia	Integridad del Dominio	ListaInstitucion() Verifica si la institución se encuentra dentro de la lista de instituciones autorizadas. Tipo de Resultado: (0,1) Granularidad: elemento
Establecimiento de Salud		
Consistencia	Integridad del Dominio	ListaEstablecimiento() Verifica si el establecimiento se encuentra dentro de la lista de establecimientos autorizados. Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad Intra-relación	VerificaEstablecimientoInstitución() Verifica que el establecimiento pertenezca a la institución a la que se hace referencia en el mismo conjunto de datos. Tipo de Resultado: (0,1) Granularidad: elemento

Personal de Salud		
Consistencia	Integridad del Dominio	ListaPersonalSalud() Verifica si el personal de salud se encuentra dentro de la lista de profesionales habilitados. Tipo de Resultado: (0,1) Granularidad: elemento
Documento de Identidad		
Exactitud	Correctitud Sintáctica	VerificaCIUruguay() Si el Tipo de Documento y País es C.I. Uruguay, verifica que el dato sea del tipo numérico, que el número tenga entre 7 y 8 cifras, que no sea superior a 9.999.999 y que el último sea el dígito verificador correcto. Tipo de Resultado: (0,1) Granularidad: elemento
Lugar Defunción		
Exactitud	Correctitud Sintáctica	FormatoNumero () Verifica si la variable tiene formato numérico. Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad del Dominio	ListaLugarDefuncion() Verifica si el parámetro se encuentra dentro la lista de lugares de defunción (ver Tabla A12.4 de Lugar de Defunción). Tipo de Resultado: (0,1) Granularidad: elemento
Sexo		
Exactitud	Correctitud Sintáctica	FormatoSexo () Verifica si la variable tiene formato numérico. Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad del Dominio	ListaSexo() Verifica si el sexo se encuentra definido en la Tabla de Sexo del Apéndice 12. Tipo de Resultado: (0,1) Granularidad: elemento
Estado Civil		
Exactitud	Correctitud Sintáctica	FormatoEstadoCivil() Verifica si la variable tiene formato char(1). Tipo de Resultado: (0,1) Granularidad: elemento

Consistencia	Integridad del Dominio	ListaEstadoCivil() Verifica si el estado civil se encuentra definido en la Tabla de Estado Civil del Apéndice 12. Tipo de Resultado: (0,1) Granularidad: elemento
País		
Exactitud	Correctitud Sintáctica	FormatoPais() Verifica si la variable tiene formato char(3). Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad del Dominio	ListaPais() Verifica si el país se encuentra dentro de la lista de países definidos en la ISO 3166-1. Tipo de Resultado: (0,1) Granularidad: elemento
Departamento de Uruguay		
Exactitud	Correctitud Sintáctica	FormatoDepartamento() Verifica si la variable tiene formato char(2). Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad del Dominio	ListaDepartamento() Verifica que si existe el departamento, este se encuentra dentro de la lista de departamentos de Uruguay (ISO 3166-2). Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad Intra-relación	VerificaDepartamentoPais() Verifica que si existe departamento es porque, para el mismo conjunto de datos, el país de referencia es Uruguay. Tipo de Resultado: (0,1) Granularidad: elemento
Localidad Departamento de Uruguay		
Exactitud	Correctitud Sintáctica	FormatoLocalidad() Verifica si la variable tiene formato texto (no contiene números y símbolos). Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad Intra-relación	VerificaLocalidadDepartamento() Verifica que si la localidad se encuentra entre la lista de localidades del departamento para el mismo conjunto de datos. Tipo de Resultado: (0,1)

		Granularidad: elemento
Fecha pasada		
Exactitud	Correctitud Semántica	FechaHoraMenorHoy() Verifica que la Fecha sea menor o igual a la Fecha de hoy. Tipo de Resultado: (0,1) Granularidad: elemento
Campo texto		
Exactitud	Correctitud Sintáctica	FormatoTexto() Verifica si la variable tiene formato texto (no contiene números y símbolos). Tipo de Resultado: (0,1) Granularidad: elemento
Campo numérico		
Exactitud	Correctitud Sintáctica	FormatoNumero() Verifica si la variable tiene formato número (sólo dígitos). Tipo de Resultado: (0,1) Granularidad: elemento
Tipo Documento		
Exactitud	Correctitud Sintáctica	FormatoTipoDocumento() Verifica si la variable tiene formato número (sólo dígitos). Tipo de Resultado: (0,1) Granularidad: elemento
Consistencia	Integridad del Dominio	ListaTipoDocumento() Verifica si el tipo de documento se encuentra dentro de los valores indicados en la Tabla A12.3 Tipo de Documento. Tipo de Resultado: (0,1) Granularidad: elemento
Texto libre, Fecha de Evento, Institución de Salud, Establecimiento de Salud, Personal de Salud, Documento de Identidad, Sexo, Estado Civil, País, Fecha Pasada, Tipo Documento y Campo Texto		
Compleitud	Densidad	Nulo() Verifica si la variable tiene valor nulo. Tipo de Resultado: (0,1) Granularidad: elemento

Tabla A10.1. Métricas definidas en el modelo.

Apéndice 11. Métodos implementados con reglas

En este apéndice se detalla la implementación de las Métricas usadas en el caso de estudio, describiendo las Reglas utilizadas y la relación lógica entre ellas, demostrando la capacidad del prototipo para aplicar un escenario real. Asimismo se detallan las Acciones asociadas a cada Método.

Para facilitar la comprensión, los Métodos son creados con el mismo nombre que la Métrica que implementan.

Nulo

Descripción	Verifica si la variable tiene valor nulo		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	Variable	EXISTS	-
Relación entre reglas	R1		

FormatoFechaHora

Descripción	Verifica si la <i>Fecha</i> tiene formato “yyyy-MM-dd HH:mm:ss”		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	<i>Fecha</i>	DATE_FORMAT	“yyyy-MM-dd HH:mm:ss”
Relación entre reglas	R1		

FechaHoraMenorHoy

Descripción	Verifica si la <i>Fecha</i> es menor al día en que se efectúa la evaluación.		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	<i>Fecha</i>	BEFORE_TODAY	-
Relación entre reglas	R1		
Acciones	CancelarMensaje		

RangoFechaHora

Descripción	Verifica si la variable <i>Fecha</i> cumpla: Hoy() > <i>Fecha</i> >= Hoy() -1		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	<i>Fecha</i>	RECENT_DATE	-
Relación entre reglas	R1		

ListaInstitucion

Descripción	Verifica si la <i>Institución</i> se encuentra dentro de la lista <i>Instituciones Autorizadas</i>		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	<i>Institución</i>	CONTAINED	<i>Instituciones Autorizadas (*)</i>
Relación entre reglas	R1		

De forma análoga se implementan los Métodos: ListaEstablecimiento, ListaPersonalSalud, ListaSexo, ListaEstadoCivil, ListaPais, ListaDepartamento, ListaTipoDocumento, ListaLocalidad y ListaLugarDefuncion.

FormatoNumero

Descripción	Verifica si la variable tiene formato numérico.		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	Variable	NUMBER_FORMAT	-
Relación entre reglas	R1		

FormatoEstadoCivil

Descripción	Verifica si la variable <i>EstadoCivil</i> tiene es de largo 1 y de tipo texto sin contener números ni símbolos.		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	<i>EstadoCivil</i>	TEXT_FORMAT	-
2	<i>EstadoCivil</i>	LENGTH_EQUALS_THAN	1
Relación entre reglas	R1 and R2		

De forma análoga se implementan los Métodos: FormatoLocalidad, FormatoDepartamento y FormatoPais.

VerificaDepartamentoPais

Descripción	Verifica que si existe departamento es porque, para el mismo conjunto de datos, el país de referencia es Uruguay.		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	Departamento	EXISTS	-
2	País	EQUALS	“URY”
Relación entre reglas	(R1 and R2) or (not R2)		

FormatoTexto

Descripción	Verifica si la variable tiene formato texto (no contiene números ni símbolos).		
Reglas			
Id Regla	Parámetro 1	Función	Parámetro 2
1	Variable	TEXT_FORMAT	-
Relación entre reglas	R1		

(*) Para ingresar una lista, sus elementos se deben agregar separados por una coma, ejemplo:
Maldonado, Montevideo, Canelones.

Apéndice 12. Tablas de referencia del servicio Obtener Certificados

En esta sección se describen las tablas utilizadas como referencia en los parámetros de las operaciones brindadas por servicio.

Tabla de Sexo

A continuación se muestran los valores disponibles para los elementos “Sexo”.

Id	Valor
1	MASCULINO
2	FEMENINO
3	INDETERMINADO O AMBIGUO
4	SE DESCONOCE
9	NO APLICA

Tabla A12.1. Valores posibles del elemento Sexo.

Tabla de Estado Civil

A continuación se muestran los valores disponibles para los elementos “Estado Civil”.

Id	Valor
C	CASADO
D	DIVORCIADO
V	VIUDO
S	SOLTERO
I	SE DESCONOCE

Tabla A12.2. Valores posibles del elemento Estado Civil.

Tabla de Tipo de Documento

A continuación se muestran los valores disponibles para los elementos “Tipo de Documento”.

Id	Valor
68909	CÉDULA DE IDENTIDAD
68912	PASAPORTE
68944	CREDENCIAL CÍVICA
69012	LIBRETA DE CONDUCIR EXTRANJERA
69011	LIBRETA DE CONDUCIR NACIONAL
69024	OTRO DOCUMENTO DE IDENTIFICACIÓN PERSONAL
68945	SIN DOCUMENTO
68910	DOCUMENTO DE IDENTIDAD

Tabla A12.3. Valores posibles del elemento Tipo de Documento.

Tabla de Lugar de Defunción

A continuación se muestran los valores disponibles para los elementos “Lugar de Defunción”.

Id	Valor
1	Centro Asistencial
2	Domicilio
3	Vía Pública
4	Casa de Salud/Asilos
5	Trabajo
6	Cárcel
7	Otro

Tabla A12.4. Valores posibles del elemento Lugar de Defunción.