

INSTITUTO DE COMPUTACIÓN - FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA
MONTEVIDEO, URUGUAY

Protección de Datos Personales en Plataformas de Integración

Informe de Proyecto de Grado

Andrés Echevarría
Dahiana Morales

Tutora: Ing. Laura González

Abril, 2015

Resumen

Los Sistemas de Información Cooperativos (SIC) son sistemas de información a gran escala que interconectan los sistemas de *software* de diferentes organizaciones autónomas, distribuidas geográficamente y con objetivos comunes. Los SICs son sistemas fuertemente distribuidos y altamente coordinados, que a menudo presentan patrones de interacción interorganizacional y requieren compartir y acceder de forma distribuida a recursos de cómputo e información. En general, los SICs basados en servicios se apoyan fuertemente en plataformas de interoperabilidad las cuales brindan una capa de procesamiento intermedio entre las organizaciones.

En Uruguay, por ejemplo, la Agencia de Gobierno Electrónico y Sociedad de la Información (AGESIC) provee, mediante su Plataforma de Gobierno Electrónico (PGE), una plataforma de interoperabilidad que brinda infraestructura (*hardware* y *software*) y servicios utilitarios, que reducen la complejidad de implementar servicios al público y/o accesibles dentro del Estado. Esto permite la implementación de una *Service Oriented Architecture* (SOA) a nivel del Estado, en la cual los servicios ofrecidos por los organismos son descriptos, publicados y descubiertos, invocados y combinados a través de interfaces y protocolos estandarizados.

El gran volumen e intercambio de datos que manejan los SICs y la relevancia y privacidad de los mismos hizo que a nivel mundial fuera necesario comenzar a trabajar en el establecimiento de normativas asociadas a la protección de datos personales. En particular en Uruguay en el 2008 se promulgó la Ley N° 18.331 de Protección de Datos Personales y Acción de “*Habeas Data*”. Este tipo de normativas se vuelven aún más relevantes en áreas como la salud.

En este contexto, resulta de interés que las plataformas de integración para SICs provean mecanismos que permitan la gestión de estas normativas, así como el monitoreo y aseguramiento de su cumplimiento en el marco de las interacciones que se dan entre las organizaciones.

En este proyecto se proponen lineamientos para el diseño de un componente que permita a plataformas como la PGE gestionar las normativas asociadas a la protección de datos personales así como también monitorear y asegurar su cumplimiento en las interacciones que se dan entre los organismos. Para esto se comenzó con un estudio de la ley ya mencionada para conocer los conceptos que se manejan y qué aspectos se deben tener en cuenta para garantizar el cumplimiento de la misma. Luego se examinó la comunicación entre los distintos organismos a través de la PGE y se establecieron mecanismos y acciones a tomar para garantizar el cumplimiento de la ley. Por último se planteó una propuesta de solución que contempla los puntos anteriores y se desarrolló un prototipo para evaluar su viabilidad y validar la solución propuesta.

Palabras Clave: *Protección de Datos Personales, Enterprise Service Bus, web services, Plataformas de Integración, gobierno electrónico.*

Contenido

1	INTRODUCCIÓN	7
1.1.	CONTEXTO, MOTIVACIÓN Y DEFINICIÓN DEL PROBLEMA	7
1.2.	OBJETIVOS.....	8
1.3.	APORTES DEL PROYECTO	9
1.4.	ORGANIZACIÓN DEL DOCUMENTO.....	9
2	MARCO CONCEPTUAL.....	11
2.1	SERVICE ORIENTED ARCHITECTURE (SOA).....	11
2.2	WEB SERVICES	11
2.3	ENTERPRISE SERVICE BUS	14
2.4	SEGURIDAD	16
2.5	PLATAFORMA DE GOBIERNO ELECTRÓNICO DEL ESTADO URUGUAYO	21
3	ANÁLISIS DE REQUERIMIENTOS	27
3.1	CONCEPTUALIZACIÓN DE LA LEY DE PROTECCIÓN DE DATOS PERSONALES ...	27
3.2	REQUERIMIENTOS	29
4	SOLUCIÓN PROPUESTA.....	33
4.1	ARQUITECTURA GENERAL.....	33
4.2	PRINCIPALES CARACTERÍSTICAS	34
4.3	DISEÑO DE COMPONENTES.....	38
4.4	INTERACCIÓN ENTRE COMPONENTES	39
4.5	INTEGRACIÓN CON LA PGE	41
5	IMPLEMENTACIÓN	43
5.1	HERRAMIENTAS UTILIZADAS.....	43
5.2	DETALLES DE IMPLEMENTACIÓN.....	49
6	CASOS DE ESTUDIO	59
6.1	OBJETIVO	59
6.2	VALIDACIÓN DE MENSAJES	59
6.3	SOLICITUD DE ACCESO A DATOS	63
6.4	DASHBOARD DE ADMINISTRACIÓN Y MONITOREO	64
7	CONCLUSIONES Y TRABAJO A FUTURO.....	67
7.1	CONCLUSIONES	67

7.2 TRABAJO A FUTURO	68
8 REFERENCIAS.....	71
APÉNDICE 1. MENSAJES XACML	75
APÉNDICE 2. TRANSFORMACIONES XSLT.....	77
APÉNDICE 3. SERVICIO BÁSICO DE INFORMACIÓN.....	79
APÉNDICE 4. PLATAFORMA DE GOBIERNO ELECTRÓNICO.....	83
APÉNDICE 5. GLOSARIO	87

1 Introducción

En este documento se presenta el proyecto de grado denominado “Protección de Datos Personales en Plataformas de Integración” el cual está enmarcado en las áreas de Bases de Datos y Sistemas de Información. En este capítulo se presenta el contexto y motivación del proyecto y se plantean los objetivos y aportes del mismo.

1.1. Contexto, Motivación y Definición del Problema

Hoy en día los requerimientos operativos de las organizaciones evolucionan rápidamente a lo largo del tiempo. Este dinamismo propone la necesidad de contar con herramientas que se adapten fácilmente y acompañen estos cambios.

Los Sistemas de Información han sido una solución eficaz para este tipo de requerimientos y son cada vez más habituales en contextos organizacionales. Los mismos proveen una solución eficaz al problema de la recolección, almacenamiento, procesamiento y recuperación de información. El gran desarrollo de este tipo de sistemas y la necesidad de compartir información entre distintas organizaciones dio lugar a los Sistemas de Información Cooperativos (SICs) [1].

Los SICs son conjuntos de sistemas de distintas organizaciones que interactúan cooperativamente compartiendo datos, restricciones y reglas de negocio para resolver problemas conjuntamente. Para implementarlos es necesario contar con herramientas que faciliten la coordinación y administración de la comunicación entre las distintas organizaciones y establecer un lenguaje común entre ellas. En general, estas capacidades son provistas por una plataforma de integración, la cual provee una capa de procesamiento intermedio entre las organizaciones. Estas plataformas deben establecer un formato común para la comunicación entre las organizaciones y adaptarse fácilmente a los cambios de cada una de ellas.

En la Figura 1 se presenta un ejemplo de SIC donde se observan distintas organizaciones intercambiando datos a través de servicios y la plataforma de integración que permite este intercambio.



Figura 1- Sistema de Información Cooperativo [2]

Por otro lado, con el avance de la tecnología y la comunicación, las organizaciones manejan grandes volúmenes de datos de diversos tipos los cuales bajo este contexto de colaboración pasan a ser un recurso que puede compartirse. La gran ventaja es que se pueden interconectar datos de diferentes organizaciones logrando datos más completos, variados y vinculados.

Sin embargo, a pesar de las ventajas y facilidades que brinda la cooperación entre organizaciones, este intercambio de datos plantea distintos desafíos relacionados a la privacidad y confidencialidad de los datos que se manejan.

Esto provocó que en Uruguay se decreta la Ley N° 18.331 de Protección de datos Personales y Acción de “*Habeas Data*” [3]. En esta ley se establecen distintas pautas y normas que se deben cumplir para el tratamiento de los datos personales.

En este contexto, resulta de interés que las plataformas de integración para SICs provean mecanismos que permitan gestionar, monitorear y hacer cumplir este tipo de normativas.

1.2. Objetivos

El objetivo general del proyecto es proponer y diseñar mecanismos a incorporar en una plataforma de integración que permitan gestionar, monitorear y asegurar el cumplimiento de normativas asociadas a la protección de datos personales.

En particular el proyecto se centra en la Plataforma de Gobierno Electrónico (PGE) perteneciente a la Agencia para el Desarrollo del Gobierno de Gestión Electrónica y la Sociedad de la Información y el Conocimiento (AGESIC) [4] y la ley de protección de datos personales del Uruguay.

Para cumplir con el objetivo general se plantean los siguientes objetivos específicos:

- Relevar los conceptos y requerimientos que introduce la Ley N° 18.331 de Protección de Datos Personales y Acción de “*habeas data*”.
- Proponer mecanismos a incorporar en la PGE que permitan gestionar, monitorear y garantizar la protección de datos personales de acuerdo a las normativas existentes.
- Realizar un prototipo que permita validar la solución propuesta en el contexto de la PGE.
- Implementar casos de estudio en el contexto de la PGE utilizando servicios específicos.

1.3. Aportes del Proyecto

A continuación se destacan los principales aportes de este proyecto:

- Se identificaron las problemáticas y las necesidades en relación la protección de datos personales y las normativas establecidas en la Ley N° 18.331 de Protección de Datos Personales.
- Se identificaron estándares y herramientas para ser utilizadas en la problemática de la protección de datos personales, en particular XACML.
- Se diseñó una propuesta de solución a la problemática planteada en el contexto de la PGE.
- Se implementó un prototipo basado en la herramienta SwitchYard que permitió validar la solución propuesta y la madurez de las herramientas utilizadas.
- Se implementaron tres casos de estudio en el contexto del gobierno electrónico utilizando servicios que actualmente están ofreciendo los organismos de Uruguay.

1.4. Organización del Documento

El resto del documento se organiza de la siguiente manera:

En el Capítulo 2 se encuentra el marco conceptual en el cual se exhiben los principales conceptos utilizados en este proyecto.

En el Capítulo 3 se presenta el análisis de requerimientos realizado.

En el Capítulo 4 se detalla la solución propuesta. Se comienza con una descripción conceptual para luego presentar la arquitectura y sus principales componentes.

En el Capítulo 5 se presentan las herramientas utilizadas en la realización del prototipo. Además se destacan algunos aspectos de la implementación.

En el Capítulo 6 se desarrollan tres casos de estudio en el marco del gobierno electrónico.

Para finalizar, en el Capítulo 7 se presentan las conclusiones del proyecto y posibles mejoras y trabajos a futuro.

2 Marco Conceptual

En este capítulo se realiza una breve descripción de conceptos, estándares y tecnologías con el objetivo de brindar un marco conceptual que sirva como base para el entendimiento del resto del documento.

2.1 Service Oriented Architecture (SOA)

SOA es una forma lógica de diseñar un sistema de *software* para proveer servicios a aplicaciones u otros servicios distribuidos en la red, a través de interfaces que son publicadas y pueden ser descubiertas de manera dinámica.

Consiste en un enfoque arquitectónico basado en la definición de una colección de servicios, los cuales se comunican entre sí de forma arbitraria.

Las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente operables. La comunicación de estos servicios está basada en definiciones formales independientes de cualquier plataforma o lenguaje de programación. La definición de la interfaz encapsula particularidades de una implementación, permitiendo que sea independiente de la tecnología de desarrollo [5].

2.2 Web services

Un *web service* es una aplicación de *software* identificada por una URI. Sus interfaces y formas de acceso pueden ser definidas, descritas y descubiertas como artefactos XML. Además soporta la interacción directa con otros componentes de *software* utilizando mensajes basados en XML que son intercambiados a través de protocolos basados en internet [6].

Los *web services* se han convertido en la tecnología preferida para la implementación de SOAs debido a que son el principal mecanismo de integración de aplicaciones construidas sobre distintas tecnologías [6].

Los principales estándares en los que se basa la tecnología de *web services* son: *Simple Object Access Protocol* (SOAP), *Web Services Description Language* (WSDL) y *Universal Description Discovery and Integration* (UDDI). En las próximas subsecciones se describen con más detalle estos estándares [6]. Además se presenta el estándar WS-Addressing.

2.2.1 Simple Object Access Protocol (SOAP)

SOAP [7] es un protocolo construido para el intercambio de información estructurada en entornos distribuidos. Está basado en XML y permite construir mensajes de forma independiente a los protocolos subyacentes.

Los mensajes SOAP están formados por un *envelope* que contiene un *header* y un *body*.

Los cabezales contienen información referente a infraestructura y direccionamiento, mientras que el *body* del mensaje contiene información específica del negocio.

El contenido del cuerpo del mensaje se encuentra en formato XML y se ajusta a lo especificado en la descripción del *web service* (WSDL) invocado. En la Figura 2 se presenta un ejemplo de mensaje SOAP donde se invoca la operación “sayHello” de un servicio, pasando como parámetro un nombre.

```
<soapenv:Envelope
  xmlns:q="http://fing.edu.uy/webservices/samples/"
  ...
  <soapenv:Header/>
  <soapenv:Body>
    <q:sayHello>
      <name>Juan</name>
    </q:sayHello>
  </soapenv:Body>
</soapenv:Envelope>
```

Figura 2 - Mensaje SOAP

2.2.2 Web Services Description Language (WSDL)

WSDL [8] es una especificación de un lenguaje basado en XML para describir de forma estándar un *web service*. Describe la forma de comunicación, es decir, los protocolos de mensajería XML soportados, información de tipos de datos para los mensajes, información acerca del transporte específico a utilizar e información de direccionamiento para localizar al *web service* y las operaciones que ofrece [9].

Los documentos WSDL están compuestos por dos grandes partes: una descripción abstracta y una descripción concreta. La descripción abstracta representa la interfaz del *web service* y no referencia a ningún protocolo ni tecnología en particular. Por otro lado, la descripción concreta consiste en la definición de protocolos de comunicación y transporte que permiten la ejecución de las operaciones definidas en la interfaz del servicio [9]. En la Figura 3 se encuentra un ejemplo de la estructura de documento WSDL.

```
<?xml version="1.0">
<definitions ...>

  <types>
    ...
  </types>

  <message>
    ...
  </message>
  <portType name="...">
    <operation name="...">
      <output message="..." />
    </operation>
  </portType>

</definitions>
```

Diagram illustrating the structure of a WSDL document. The document is divided into two main sections within the `<definitions>` element:

- Data Types in XSD:** Indicated by a bracket on the right, this section corresponds to the `<types>` element.
- Interface Description:** Indicated by a bracket on the right, this section corresponds to the `<message>` and `<portType>` elements.

Figura 3 - Esquema documento WSDL [10]

2.2.3 Universal Description, Discovery and Integration (UDDI)

UDDI [11] es una especificación de la OASIS cuyo propósito principal consiste en, facilitar la categorización, descubrimiento y recuperación de *web services*. Los registros UDDI permiten clasificar los servicios de acuerdo al tipo de servicio, tipo de negocio o a las relaciones que poseen con otros servicios. Además, define una API basada en *web services* que permite buscar, recuperar y publicar servicios.

2.2.4 WS-Addressing

WS-Addressing [12] “es un conjunto de especificaciones recomendadas por la W3C, que definen un mecanismo independiente del medio de transporte para referenciar *web services* y “direccionar” mensajes. Dentro de este conjunto de especificaciones se encuentran: *WS-Addressing - Core*, *WS-Addressing - SOAP Binding* y *WS-Addressing – Metadata*” [13].

WS-Addressing Core “es la principal de las especificaciones ya que define qué es un *Endpoint Reference (EPR)*, qué son los *Message Headers* y cómo pueden ser usados para referenciar *web services* o ser útiles para el direccionamiento de mensajes. Una de las principales características de esta especificación es que sus definiciones son totalmente abstractas e independientes del medio de transporte, por lo cual es posible utilizarlas en conjunto con *HTTP*, *SMTP* o cualquier otro protocolo” [13].

Un EPR se compone de una dirección, parámetros secundarios y *metadata*. Los *Message Headers* o *Message Addressing Properties* son un conjunto de propiedades que permiten direccionar los mensajes brindando soporte a distintos patrones de comunicación para *web services*. Algunas de las propiedades definidas por *WS-Addressing Core* son:

- To: Es una IRI absoluta que representa la dirección del receptor del mensaje.
- From: Es un EPR que indica quién envió el mensaje.
- ReplyTo: Es un EPR que indica a dónde se deben enviar las respuestas del mensaje.
- FaultTo: Es un EPR que indica a dónde se deben enviar los errores relacionados al mensaje enviado.
- Action: Es una IRI absoluta que identifica de forma unívoca la semántica del mensaje.
- MessageID: Es una IRI absoluta que identifica de forma unívoca el mensaje.

La especificación *WS-Addressing – SOAP Binding* define cómo utilizar las propiedades definidas de forma abstracta en el Core, en mensajes SOAP.

Finalmente, *WS-Addressing – Metadata*, indica cómo describir las propiedades utilizando un WSDL, cómo incluir metadata del WSDL en un *Endpoint Reference* y cómo utilizar *WS-Policy* para indicar que un *web service* soporta *WS-Addressing* [13].

2.3 Enterprise Service Bus

Un *Enterprise Service Bus* (ESB) consiste en una plataforma de integración basada en estándares que combina, *web services*, transformación de datos, mensajería y ruteo inteligente para conectar y coordinar, de forma confiable, la interacción de diversas aplicaciones con integridad transaccional [14].

Si bien la tecnología de *web services* constituye una base sólida para la implementación de SOAs, existen algunos aspectos, como su naturaleza punto a punto, que pueden afectar la flexibilidad y escalabilidad de soluciones que usan exclusivamente esta tecnología [13].

El ESB brinda una capa de integración intermedia que provee lógica de integración y comunicación reutilizable, para posibilitar la interacción entre clientes y servicios en una SOA. El ESB acepta pedidos en la forma de mensajes, sobre los cuales se pueden realizar distintas operaciones de mediación [14] para solucionar heterogeneidades entre clientes y servicios. A modo de ejemplo, el ESB puede interactuar con servicios a través de protocolos de comunicación no soportados por los clientes, o puede transformar los mensajes enviados por el cliente para que se ajusten al formato que espera el servicio [13].

La utilización de un ESB promueve el bajo acoplamiento entre clientes y servicios, dividiendo la lógica de comunicación e integración en pequeños componentes de *software* administrables. Permite además tener una clara separación entre la lógica de integración y comunicación, y la lógica de negocio implementada por los servicios [13].

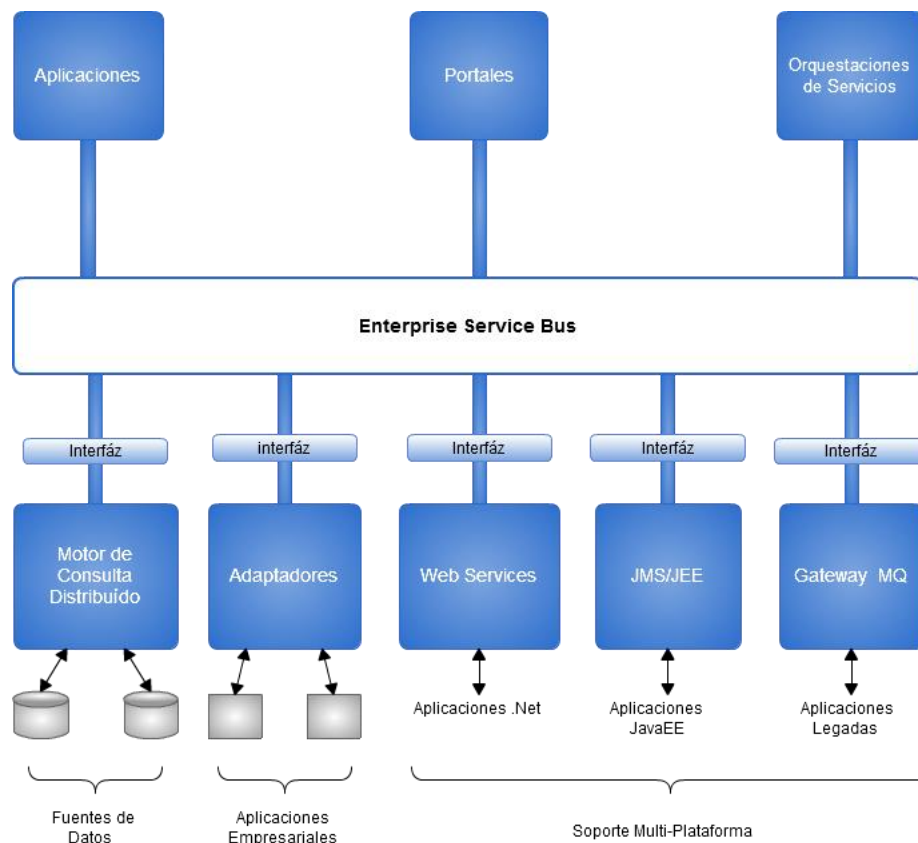


Figura 4 - Enterprise Service Bus [49]

En la Figura 4 se pueden observar, cómo distintas aplicaciones y servicios que utilizan tecnologías, formatos de datos y protocolos de comunicación distintos, se comunican con el ESB a través de servicios con interfaces bien definidas. Dichos servicios pueden ser orquestados o utilizados por otras aplicaciones o servicios.

Funcionalidades de ESB

Las principales funcionalidades provistas generalmente por los productos ESB son: la transformación de protocolos de comunicación, la transformación y enriquecimiento de mensajes, el ruteo de mensajes y mecanismos de mensajería confiable [13].

Asimismo, en general también cuentan con funcionalidades que permiten garantizar propiedades de calidad de servicio, como performance, disponibilidad y seguridad [13] [14] [16]. A continuación se detallan los más relevantes para este proyecto.

Transformación y Enriquecimiento de Mensajes

Los productos ESB brindan una serie de mecanismos para transformar y enriquecer mensajes intercambiados por clientes y servicios, por ejemplo, utilizando el estándar XSLT [15]. Estos mecanismos de transformación pueden utilizarse para afrontar distintos requerimientos, como la resolución de diferencias entre los formatos de datos intercambiados [13].

Además, las transformaciones podrían ser utilizadas para minimizar el impacto, en las aplicaciones cliente, ante posibles cambios en los servicios. Por ejemplo, si un servicio cambiara su interfaz funcional, en ciertas ocasiones se podrían utilizar transformaciones para que los mensajes enviados por clientes se ajusten a la nueva interfaz. Dado que esto se resuelve en el ESB, los cambios en los servicios no tendrían impacto en las aplicaciones cliente [13].

Ruteo de mensajes

Los ESB poseen la capacidad de determinar, en tiempo de ejecución, el destino del mensaje de acuerdo a distintos factores. Algunos tipos de ruteo utilizados por los ESB son el ruteo basado en contenido, ruteo para balanceo de carga, ruteo basado en contexto y ruteo basado en itinerario.

El ruteo basado en contenido tal como lo indica su nombre determina el destino del mensaje en base a su contenido. En el caso de mensajes SOAP, la lógica de ruteo puede basarse en el contenido de los encabezados del mensaje o del cuerpo [17].

El ruteo para el balanceo de carga determina el destino del mensaje utilizando una estrategia de balanceo de carga como Round-Robin, Random y LeastLoad, entre otras [18] [17].

El ruteo basado en contexto determina el destino del mensaje en base a propiedades del contexto de ejecución, como por ejemplo el momento del día o la carga actual del sistema [17].

Por último, el ruteo basado en itinerario (*Routing Slip*) determina el destino del mensaje en base a un itinerario. El itinerario es incluido en el propio mensaje y especifica al ESB el camino que debe recorrer el mismo [19] [17].

Monitoreo

Los productos de ESB cuentan con varias funcionalidades nativas que permiten el monitoreo de distintos tipos de información como contenido de los mensajes, tiempos de respuesta de los servicios, cantidad de invocaciones a los servicios, etc [13].

Patrón Modelo de Datos Canónicos (MDC)

Es un patrón de integración implementado en algunos productos ESB que permite la comunicación entre sistemas con diferentes modelos de datos, estableciendo una forma estándar de comunicación. En general es utilizado para resolver la integración entre distintas organizaciones con varios sistemas legados con diferentes esquemas de base de datos. La utilización de este patrón implica la definición de un modelo común a todos los sistemas involucrados, donde se mapean las principales entidades. A modo de ejemplo, cuando un sistema A desea enviar el objeto Ciudadano a un sistema B, deberá transformar su objeto Ciudadano al objeto Ciudadano del MDC para poder enviarlo a la aplicación destino. La aplicación destino convertirá el objeto Ciudadano del MDC a su modelo de Ciudadano [50].

2.4 Seguridad

A continuación se realiza una breve introducción de algunos conceptos básicos de seguridad.

2.4.1 Conceptos Básicos

- **Autenticación:** Consiste en verificar la identidad de un usuario, proceso o dispositivo, a menudo como un prerrequisito para permitir el acceso a recursos en un sistema de información [20].
- **Confidencialidad de datos:** Hace referencia a que la información sensible no es revelada a individuos, entidades o procesos no autorizados [20].

Los datos intercambiados a través de una red deben ser protegidos para evitar que usuarios no autorizados puedan tener acceso a los mismos. La técnica estándar para asegurar la confidencialidad de los datos intercambiados es la encriptación [21].

- **Integridad:** Es la propiedad de que los datos no han sido alterados de manera no autorizada. Esto aplica a los datos en su almacenamiento, durante el procesamiento y en su transmisión [20].
- **No repudio:** Consiste en tener protección contra el hecho de que un sujeto niegue falsamente haber realizado una acción. Provee medios para determinar que el sujeto realizó cierta acción tal como crear información,

enviar un mensaje, aprobar información o recibir un mensaje [20]. Una técnica habitual para brindar integridad y no repudio, es el uso de firmas digitales.

- **Autorización y control de acceso:** Una autorización es el derecho o permiso dado a una entidad para acceder a un recurso. El proceso de autorización es un procedimiento para conceder estos derechos [22]. El control de acceso consiste en la protección de los recursos del sistema contra acceso no autorizado. Es un proceso por el cual el uso de los recursos es regulado de acuerdo a una política y es permitido solamente a las entidades autorizadas (usuarios, procesos u otros sistemas) de acuerdo a dicha política [22].
- **Disponibilidad:** La propiedad de que un recurso esté accesible y usable a demanda por una entidad autorizada. El ataque habitual a la disponibilidad es la denegación de servicio, que consiste en evitar o demorar los accesos autorizados a los recursos [23].
- **Auditabilidad:** La información de auditoría debe ser mantenida y protegida de forma que las acciones que afecten a la seguridad puedan ser vinculadas a la parte responsable [24]. Para poder hacer esto, el sistema debe previamente identificar y autenticar a los usuarios y mantener un registro de auditoría de los eventos que afecten a la seguridad [23].

2.4.2 Estándares de seguridad

Con el fin de garantizar la seguridad en los *web services* se han desarrollado estándares basados en XML. A los efectos de este proyecto los más importantes son:

2.4.2.1 WS-Security

“WS-Security [25] es un estándar de la OASIS que propone un conjunto estándar de extensiones a SOAP que pueden ser utilizadas para enviar tokens de seguridad como partes del mensaje SOAP y proveer integridad y confidencialidad del contenido de los mismos [9]. WS-Security describe principalmente cómo incluir tokens de seguridad en mensajes SOAP y cómo utilizar XML Encryption y XML Signature para cifrar y firmar esos tokens, así como otras partes del mensaje” [13].

WS-Security define un conjunto de *security tokens* y el mecanismo para adjuntarlos, identificarlos y referenciarlos dentro de un mensaje. El *UserNameToken*, por ejemplo, permite especificar un nombre de usuario y opcionalmente una contraseña. A su vez, el *BinarySecurityToken* permite incluir certificados X.509 o tickets Kerberos. Finalmente, a través de los *XMLTokens*, es posible adjuntar *security tokens* basados en XML utilizando distintos formatos como *Security Assertion Markup Language (SAML)* y *eXtensible Rights Markup Language (XrML)* [13].

WS-Security provee un modelo de seguridad extensible que brinda soporte a múltiples tokens de seguridad, dominios de confianza, firmas digitales y algoritmos

de encriptación, que junto con un modelo de seguridad específico (PKI, Kerberos, SAML, etc), permite lograr una solución completa y segura de punta a punta [13].

2.4.2.2 WS-Trust

WS-Trust [26] es un estándar de la OASIS que extiende *WS-Security* y define un modelo de seguridad para *web services* basado en relaciones de confianza y mecanismos de emisión, renovación y validación de *tokens* de seguridad [13].

El modelo de seguridad definido por *WS-Trust* se basa en un proceso por el cual los servicios requieren que todo mensaje recibido contenga un conjunto de *claims* para ser aceptado. Todo mensaje que no las tenga, es rechazado. Los servicios generalmente utilizan *WS-Policy* [27] para especificar qué *claims* requieren [13].

En casos en los cuales los clientes no tengan posesión de las *claims* para consumir un servicio, pueden contactar a autoridades apropiadas para así obtenerlas. Dentro de la especificación *WS-Trust*, se conoce a estas autoridades como los *Security Token Services* (STS). Los clientes pueden contactar a los STS para que éstos les emitan los *claims* y así poder consumir el servicio [13].

2.4.2.3 eXtensible Access Control Markup Language (XACML)

XACML [28] es un estándar de OASIS basado en XML que define un lenguaje de políticas de acceso y un lenguaje *request/response* para la toma de decisiones de control de acceso. El lenguaje de políticas es utilizado para describir los requerimientos generales de control de acceso, y tiene algunos puntos de extensión para la definición de nuevas funciones y tipos de datos entre otros. El lenguaje permite realizar consultas para determinar si una cierta acción puede o no realizarse e interpretar el resultado. Las respuestas siempre incluyen uno de los siguientes cuatro valores: *Permit*, *Deny*, *Indeterminate* (ocurrió algún error o falta algún valor que hace que la decisión no pueda ser tomada), *Not Applicable* (la petición no puede ser respondida por este servicio).

A continuación se detalla el modelo de flujo de información XACML y una breve descripción de los componentes que intervienen [28].

Policy Administration Point (PAP): entidad del sistema que crea las políticas o conjuntos de políticas de acceso.

Policy decision point (PDP): entidad del sistema que evalúa las políticas aplicables y emite una decisión de autorización.

Policy enforcement point (PEP): entidad del sistema que ejecuta el control de acceso, realizando solicitudes de decisión de autorización y haciendo cumplir las decisiones tomadas.

Policy information point (PIP): entidad del sistema que actúa como fuente de atributos.

Context handler: entidad del sistema que convierte las solicitudes de decisión del formato nativo de solicitud a la forma canónica XACML y convierte las decisiones de

autorización en la forma canónica XACML al formato nativo de respuesta. Las Figura 5 y Figura 6 presentan un ejemplo de *request* y de *response* respectivamente.



Figura 5 - Request XACML [28]

En el *request* se identifican los siguientes elementos:

Subject: es la entidad que solicita acceso. XACML soporta varios *Subject* y cada uno de ellos admite una lista de atributos.

Resource: especifica información sobre el recurso para el que se solicita el acceso.

Action: especifica la acción que se realizará sobre el recurso.

Environment: es un conjunto de atributos que son relevantes para la decisión de autorización independiente de *Subject*, *Resource* o *Action*.

```

<Response>
  <Result>
    <Decision>NotApplicable</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:1.0:status:processing-error"/>
      <StatusMessage>Authentication required</StatusMessage>
    </Status>
    <Obligations/>
  </Result>
</Response>

```

Figura 6 - Response XACML [28]

El *response* contiene:

Decision: decisión de autorización, los posibles resultados son “Permit”, “Deny”, “Indeterminate” o “NotApplicable”.

Status: indica si se han producido errores durante la evaluación de la solicitud de decisión y opcionalmente información acerca de esos errores.

Obligations: lista de operaciones que deben ser ejecutadas por el PEP en relación a una decisión de autorización.

En la Figura 7 se presenta el flujo de información XACML.

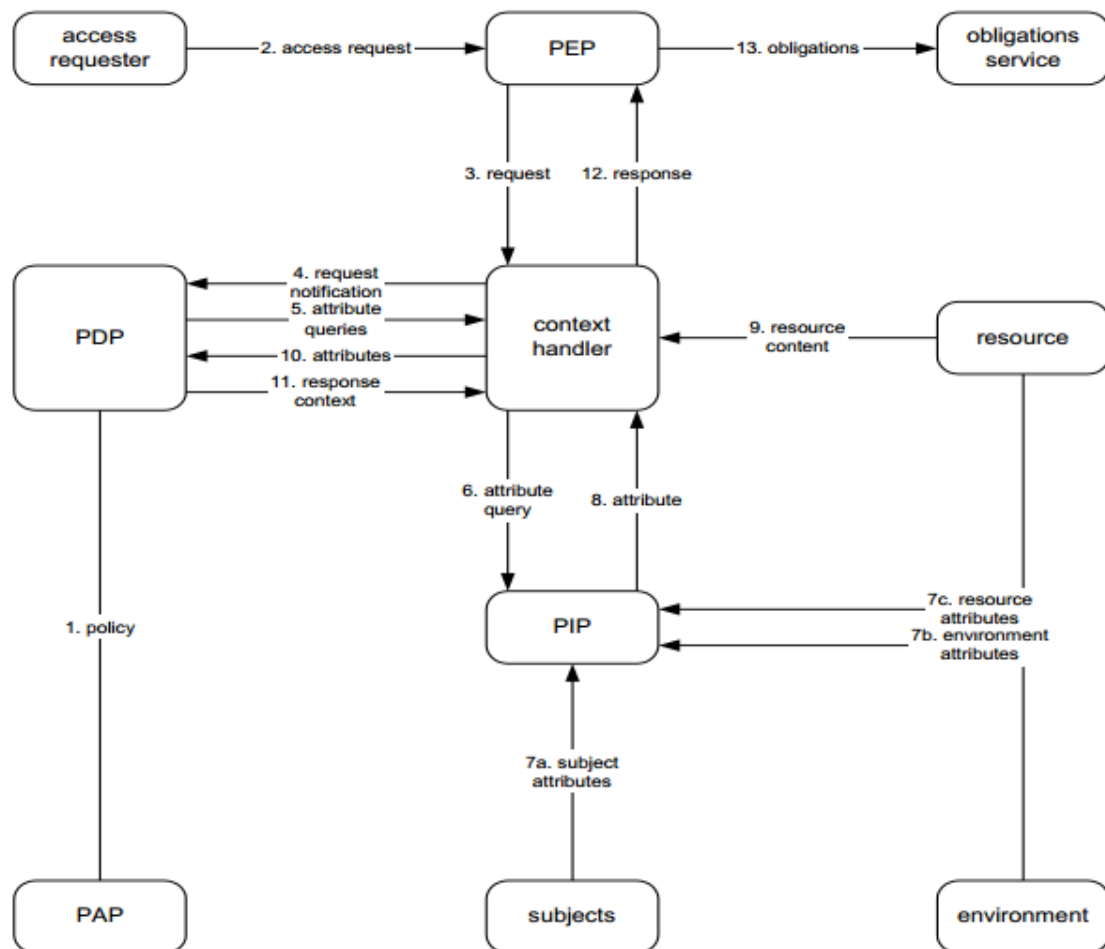


Figura 7 - Flujo de información XACML [28]

El modelo opera de la siguiente manera:

1. El Administrador de políticas crea las políticas y conjuntos de políticas en el PAP y los deja disponibles para el PDP.
2. El solicitante de acceso (*access requester*) envía una solicitud de acceso al PEP.
3. El PEP envía la solicitud de acceso al *context handler* en su formato nativo, incluyendo en forma opcional atributos del sujeto, recurso, acción y entorno.
4. El *context handler* construye una solicitud XACML y se la envía al PDP.
5. El PDP solicita atributos adicionales del sujeto, recurso, acción y entorno al *context handler*.
6. El *context handler* solicita los atributos a un PIP.
7. El PIP obtiene los atributos solicitados.
8. El PIP retorna los atributos solicitados al *context handler*.
9. Opcionalmente el *context handler* puede incluir el recurso en el contexto.
10. El *context handler* envía los atributos solicitados y opcionalmente el recurso al PDP. El PDP evalúa las políticas.
11. El PDP retorna la respuesta de contexto al *context handler* incluyendo la decisión de autorización.
12. El *context handler* traduce la respuesta de contexto al formato de respuesta nativo del PEP. El *context handler* retorna la respuesta al PEP.
13. El PEP hace cumplir las obligaciones (operaciones especificadas en las políticas que deben ser realizadas por el PEP al momento hacer cumplir la decisión de autorización) que implican las políticas que se están controlando.
14. Si la decisión indica que el acceso es permitido, el PEP permite el acceso al recurso de lo contrario el acceso es denegado.

2.5 Plataforma de Gobierno Electrónico del Estado Uruguayo

Como ya se mencionó este proyecto se enmarca bajo el contexto de la Plataforma de Gobierno del Estado Uruguayo. A continuación se presenta una breve introducción sobre este tipo de plataformas y se detallan los aspectos y componentes más importantes de la misma.

2.5.1 Introducción

El Gobierno Electrónico puede definirse como el uso de las TICs para reinventar los procedimientos del gobierno, con el fin de promover la difusión y disponibilidad de la información [29].

La implementación del Gobierno Electrónico se basa en una combinación de tecnologías y especificaciones basadas en modelos que varían según los distintos países. Las Plataformas de Gobierno Electrónico (PGEs) constituyen uno de los tipos de herramienta clave para desarrollar el Gobierno Electrónico y consisten en infraestructura y servicios que facilitan la conectividad de los sistemas de los organismos del Estado. Ofrecen servicios comunes generando economía de escala y haciendo posible el uso de tecnologías avanzadas y promueven la implementación de servicios multi-organismo [13].

En Uruguay, la Plataforma de Gobierno Electrónico (PGE) de AGESIC [13] tiene como objetivo general facilitar y promover la implementación de servicios de Gobierno Electrónico. El soporte tecnológico a la PGE está dado por la Plataforma de Interoperabilidad (PDI), la cual provee a los organismos del Estado de una infraestructura que facilita la implementación de servicios al público así como el acceso a servicios de otros organismos del Estado. Otros componentes de la PGE son los Servicios Transversales que se brindan a los organismos, los cuales son actualmente el Portal y Buscador del Estado Uruguayo, el Sistema de Expediente Electrónico y el Geoportal. En las siguientes subsecciones se especifican los componentes de la PDI [13].

2.5.2 Plataforma de Middleware

La plataforma de *middleware* provee mecanismos para facilitar el desarrollo, despliegue e integración de servicios y aplicaciones. Estos mecanismos brindan a su vez, la infraestructura base para la implementación de la SOA a nivel del Estado. Los organismos pueden utilizar esta plataforma para publicar y consumir servicios. Les permite contar con un ambiente de ejecución para servicios o aplicaciones que requieran infraestructura de *hardware* o *software* no disponible en los organismos. También pueden utilizar las diferentes capacidades de mediación, las cuales permiten desacoplar clientes y servicios [13].

En la plataforma de *middleware* se pueden identificar tres grandes bloques: entornos de ejecución para Aplicaciones y Servicios, un Registro de Servicios y productos de tipo *Enterprise Service Bus* [13].

2.5.3 Sistema de Seguridad

El Sistema de Seguridad provee servicios de seguridad al resto de los componentes. Brinda mecanismos que permiten realizar el control de acceso a los servicios de la plataforma, de acuerdo a políticas definidas, para que el acceso a los mismos sea seguro. También se encarga del auditado de los eventos relacionados con la seguridad [13].

2.5.4 Sistema de Gestión de Metadatos

El Sistema de Gestión de Metadatos provee una especificación de alto nivel de los conceptos relativos a servicios públicos, de forma de evitar, o eventualmente

resolver, ambigüedades en el manejo de estos conceptos por parte de los organismos [13].

En el Apéndice 4. Plataforma de Gobierno Electrónico se encuentra más información sobre la PGE.

2.5.5 Invocación de servicio de la PGE

Una aplicación cliente en un organismo debe realizar tres pasos para consumir un servicio de la PGE.

El primer paso que debe realizar es obtener un *token* de seguridad SAML (*Security Assertion Markup Language*) firmado digitalmente por el organismo. SAML es un *framework* que permite el intercambio de información de autorización y autenticación entre dominios de seguridad¹ [13].

El segundo paso consiste en obtener un *token* de seguridad firmado por la PGE. Para ello debe realizar una solicitud al STS de la PGE utilizando el estándar WS-Trust.

Por último, para invocar un servicio de la PGE, la aplicación cliente debe enviar un mensaje SOAP al Servicio *Proxy*² del servicio con la siguiente información:

1. *Token* SAML emitido por el STS de la PGE, especificado a través del estándar WS-Security.
2. Servicio y método a consumir, especificados a través de estándar WS-Addressing [13].

En la Figura 8 se presenta un ejemplo de un mensaje SOAP para consumir el *web service* “Servicio Básico de información” provisto por la Dirección Nacional de Identificación Civil (DNIC).

¹ Un dominio de seguridad es una colección de sistemas que tienen una política de seguridad única y un sólo conjunto de controles de seguridad.

² Un servicio *proxy* es un servicio que actúa como intermediario entre un cliente y un servicio destino.

```

<env:Envelope xmlns:env='... '>
<env:Header xmlns:wsa='... '>
  <wsse:Security env:mustUnderstand='1' xmlns:ds='...'
    xmlns:wsse='...' xmlns:wsu='...'>
    <saml:Assertion AssertionID='...' IssueInstant='...'
      Issuer='...' MajorVersion='1' MinorVersion='1' xmlns:saml='...'>
      <saml:Conditions NotBefore='...' NotOnOrAfter='...'>
        <saml:AudienceRestrictionCondition>
          <saml:Audience>
            Servicio http://192.168.40.190:9000/wsServicioDeInformacionBasico
          </saml:Audience>
        </saml:AudienceRestrictionCondition>
      </saml:Conditions>
      <saml:AuthenticationStatement
        AuthenticationInstant='...' AuthenticationMethod='...'>
        <saml:Subject>
          <saml:NameIdentifier>
            uid=rolPruebaDoctor,cn=BPS
          </saml:NameIdentifier>
        </saml:Subject>
        </saml:AuthenticationStatement>
        <saml:AttributeStatement>
          <saml:Subject>
            <saml:NameIdentifier>
              uid=rolPruebaDoctor,cn=BPS
            </saml:NameIdentifier>
          </saml:Subject>
          <saml:Attribute AttributeName='User'
            AttributeNamespace='urn:nac'>
            Nombre de usuario <saml:AttributeValue>Juan</saml:AttributeValue>
          </saml:Attribute>
        </saml:AttributeStatement>
        <ds:Signature Id='uuid2765608c-0128-1428-9ce4-8913af9af38d'
          xmlns:ds='http://www.w3.org/2000/09/xmldsig#'> ...
        </ds:Signature>
      </saml:Assertion>
    </wsse:Security>
  </env:Header>
  <env:Body>
    <wsa:To>http://192.168.40.190:9000/wsServicioDeInformacionBasico</wsa:To>
    <wsa:Action>
      http://.../certificadoCNVEWSDLPortType/ObtPersonaPorDoc
    </wsa:Action>
  </env:Body>
</env:Envelope>

```

Cabecales WS-Security: token SAML emitido por la PGE

Cabecales WS-Addressing: To (Servicio) y Action (Método)

Información de Negocio

Figura 8 - Ejemplo Mensaje SOAP

2.6 Ley de Protección de Datos Personales del Uruguay

En los distintos países se han implementado diferentes soluciones con el fin de abordar la problemática del control de acceso a datos personales de los ciudadanos. En particular en Uruguay se creó la Ley Nº 18.331 de Protección de Datos Personales y Acción de “*Habeas Data*” [3], la cual se consideró como caso de estudio para la realización este proyecto.

En esta sección se analizan los aspectos más relevantes de dicha ley ya que varios de ellos fueron utilizados al momento de relevar los requerimientos del sistema.

La ley establece que “*El derecho a la protección de datos personales es inherente a la persona humana*” y que “*El derecho a la protección de los datos personales se aplicará por extensión a las personas jurídicas, en cuanto corresponda*” [30]. En nuestro caso lo acotaremos a las personas humanas.

En la ley se establece que el derecho de protección de datos personales alcanza a todos los datos que identifican o pueden llegar a identificar a las personas. Un dato personal puede incluir sonidos, imágenes o datos biométricos entre otros. A modo de ejemplo se consideran datos personales al nombre, apellido, correo electrónico, una fotografía, una huella dactilar, la voz o el ADN.

Además de datos personales, la ley define los datos sensibles. Son datos personales que revelan el origen racial, étnico, preferencias políticas, convicciones religiosas o morales, afiliación sindical e informaciones referentes a la salud o a la vida sexual. La ley establece que nadie está obligado a proporcionar este tipo de datos y para obtenerlos es necesario contar con el consentimiento expreso del titular.

El consentimiento del titular debe ser:

- Libre: es decir, lo debe dar de forma voluntaria.
- Previo: debe ser pedido antes que los datos.
- Expreso: se debe dar de forma explícita.
- Documentado: debe estar registrado en algún tipo de soporte.
- Informado: al titular de los datos hay que informarle para que se usarán sus datos.

Derecho de acceso

Todo titular tiene derecho a saber qué datos se tienen de sí mismo. Para ello el titular podrá pedir toda la información que le pertenece. La información debe ser brindada en un lenguaje claro. El derecho de acceso podrá ser ejercido a intervalos de seis meses y los datos pueden ser suministrados por escrito o por medios electrónicos. El plazo máximo para dar respuesta a estos derechos es de cinco días hábiles.

Principios

La ley establece un conjunto de principios que deben ser aplicados por todos y aseguran la protección de datos personales.

Principio de legalidad: Para que una base de datos sea legítima el responsable de la misma debe inscribirla en el Registro de Bases de Datos Personales.

Principio de veracidad: Los datos deben ser verdaderos, objetivos, no excesivos y adecuados.

Principio de finalidad: Los datos personales no pueden ser utilizados para fines diferentes para los que se obtuvieron.

Principio del previo consentimiento informado: Es necesario contar con el consentimiento del titular de los datos para realizar el tratamiento de los mismos.

Principio de seguridad de los datos: El responsable de la base de datos deberá adoptar medidas para garantizar que los datos personales permanezcan seguros y confidenciales. Entre las medidas de seguridad a adoptar se encuentran las auditorías, los respaldos, el control y registro de acceso y forma de transferencia.

Principio de reserva: Los datos personales deben usarse de forma reservada. El responsable de la base de datos tiene el deber del secreto y se extiende a todo su personal. Solo se puede relevar del secreto si se cuenta con una orden judicial o con el consentimiento del titular.

Principio de responsabilidad: Establece que el responsable de la base de datos es responsable de las violaciones de las disposiciones de la ley.

3 Análisis de Requerimientos

En esta sección se presenta el análisis realizado a partir de la ley de Protección de Datos Personales y las necesidades y problemáticas planteadas en el capítulo 1.

3.1 Conceptualización de la ley de Protección de Datos Personales

A continuación se presentan los conceptos básicos para el entendimiento de los requerimientos identificados y la solución propuesta. Los mismos fueron obtenidos a partir del análisis de la ley de Protección de Datos y de una reunión llevada a cabo con Horacio Nemeth, Gerente Interino del departamento de Arquitectura y Normas de AGESIC.

- **Dato Personal**

Cuando se habla de dato personal se hace referencia a cualquier información que identifique o pueda llegar a identificar a las personas.

- **Ciudadano**

La ley establece que *“El derecho de protección de datos personales es inherente a la persona humana”*. Se utilizará el concepto “ciudadano” para referirse a toda persona física de la cual se desea proteger sus datos personales. Pese a que la ley también expresa que el derecho a la protección de datos personales se aplicará por extensión a las personas jurídicas, estas no serán consideradas en el alcance de esta solución.

- **Consentimiento**

Además de datos personales, se define un subconjunto de los mismos denominados datos sensibles. Se establece que nadie está obligado a brindar este tipo de datos y para obtenerlos es necesario contar con el consentimiento expreso del titular.

- **Organismo**

Este concepto hace referencia a los distintos organismos del Estado que se encuentran integrados a la PGE, como por ejemplo la Dirección Nacional De Identificación Civil.

Cada organismo expone un conjunto de *web services* y a su vez cada uno de ellos cuenta con una serie de métodos disponibles.

- **Finalidad**

Cuando un ciudadano brinda su consentimiento para la utilización de sus datos personales, la ley determina que el mismo debe ser informado. Esto significa que el titular de los datos debe conocer para qué serán utilizados. Se define como finalidad aquello para lo que el ciudadano proporciona sus datos personales.

Un posible uso de esos datos puede ser la realización de un trámite en un determinado organismo.

- **Acción**

Se define “acciones” como un conjunto de tareas a realizar en caso de que se detecte el incumplimiento de alguno de los aspectos establecidos por la ley de Protección de Datos Personales.

Además de los conceptos identificados a partir de la ley se introduce el concepto de *web service*. Los *web services* son provistos por los distintos organismos y están formados por un conjunto de métodos. A su vez, cada método contiene un grupo de elementos. A modo de ejemplo podemos señalar el *web service* denominado “CertificadosService” provisto por el Ministerio de Industria, Energía y Minería (MIEM). Este *web service* contiene dos métodos: “getCertificadoPymeByRUT” y “getCertificadoCCPByCodigo”. Algunos de los elementos del método “getCertificadoPymeByRUT” son: rut, code, desc, razonSocial, tipoDeSociedad, categoría y vigencia [31].

En la Figura 9 se presenta un modelo conceptual basado en los conceptos anteriores. En el mismo se puede observar que un ciudadano puede otorgar consentimientos para compartir sus datos personales con un organismo en el contexto de una determinada finalidad. La realización de una finalidad puede requerir la invocación de uno o más métodos de distintos *web services*. A cada método es posible asociarle un conjunto de acciones a realizar en caso de que se detecte el incumplimiento de alguno de los aspectos establecidos por la ley al invocar el mismo. La relación entre los elementos y los datos personales se explica más adelante.

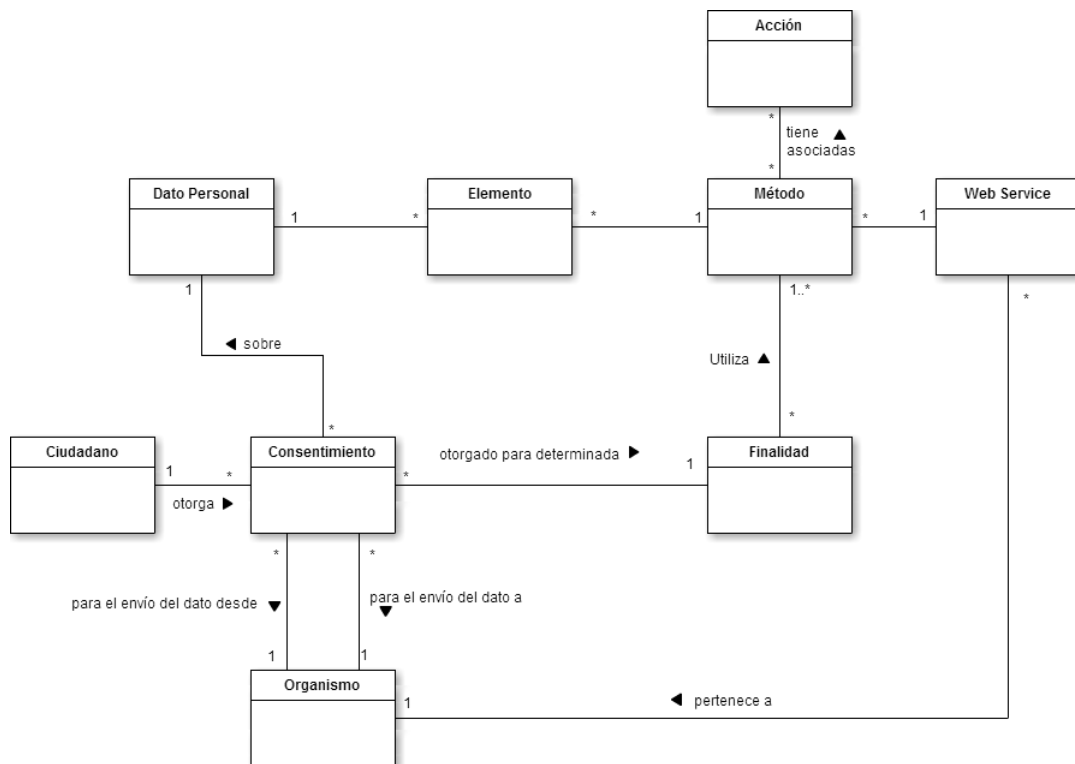


Figura 9 - Modelo Conceptual

3.2 Requerimientos

En base a los objetivos del proyecto y el análisis de la ley realizado en la sección 3.1 se definen los siguientes requerimientos.

El sistema deberá controlar los mensajes intercambiados entre los distintos organismos y garantizar el cumplimiento de la ley de protección de datos.

También deberá proveer a los ciudadanos herramientas para poder gestionar los consentimientos brindados a los distintos organismos. Además los ciudadanos deberán contar con mecanismos que les permitan obtener los datos personales que cada organismo dispone de ellos en tiempo y forma.

Por otra parte los organismos deberán ser capaces de agregar, modificar o eliminar los consentimientos de los ciudadanos si cuentan con la autorización explícita de los mismos. A su vez deberán ser capaces de gestionar los distintos pedidos de obtención de datos generados por los ciudadanos.

Por último será necesario contar con una herramienta para la configuración del sistema que permita a su vez monitorearlo.

A partir de estos requerimientos se definen los siguientes tres tipos de usuarios, Administrador General del Sistema, Administrador de Organismo y Ciudadano.

A continuación se detallan los requerimientos específicos para cada tipo de usuario.

Administrador General del Sistema

RBA-1 Alta de Organismo

El sistema debe permitir el alta de organismos en el sistema.

RBA-2 Creación de Administrador

El sistema debe permitir la creación de usuarios que se desempeñen como administradores del sistema. Cada uno de los administradores debe pertenecer a un organismo.

RBA-3 Creación de Dato personal

El sistema debe permitir la creación de datos personales.

RBA-4 Creación de Finalidad

El sistema debe permitir la creación de finalidades. Las mismas serán utilizadas para conocer el fin con el cual se utilizarán los datos personales de un ciudadano. Un ejemplo de finalidad podría ser la realización de un determinado trámite.

RBA-5 Creación de Acción

El sistema debe permitir la creación de acciones que se ejecutarán en caso de detectar que se quiere intercambiar un dato personal para el cual no se cuenta con un consentimiento, tal como lo establece la Ley N° 18.331.

RBA-6 Creación de Método

El sistema debe permitir la creación de métodos pertenecientes a *web services* que serán invocados por los distintos organismos. Cada método debe pertenecer a un organismo y estar asociado a una finalidad bajo la cual será invocado. Durante el proceso de creación de un método debe ser posible configurar las acciones a ejecutar en caso de detectar la ausencia de consentimientos.

Además debe ser posible realizar el mapeo entre los elementos del método (tanto de entrada como de salida) con datos personales. Dicho mapeo se explica en el capítulo 4.

RBA-7 Consulta, modificación y eliminación de consentimientos

El sistema debe permitir crear, eliminar y visualizar los consentimientos brindados por un ciudadano. Un consentimiento implica la autorización explícita por parte del ciudadano hacia un determinado organismo a compartir sus datos personales con otro organismo, bajo determinada finalidad y durante determinado período de tiempo.

RBA-8 Monitoreo de la aplicación

El sistema debe permitir monitorear los mensajes intercambiados entre los diferentes organismos, saber cuáles de ellos finalizaron de forma exitosa y cuáles no. Además debe permitir obtener distintas métricas del sistema.

Administrador de Organismo

RFO-1 Consulta y modificación de consentimientos

El sistema debe permitir al administrador de cada organismo obtener los consentimientos brindados por un ciudadano.

Además, dado un ciudadano y una finalidad, debe ser posible obtener los consentimientos faltantes para la realización de la finalidad, brindando la posibilidad de crearlos cuando el ciudadano lo autorice.

El sistema también debe permitir crear y eliminar consentimientos.

RFO-2 Solicitudes de acceso a datos

El sistema debe permitir al administrador de cada organismo obtener las solicitudes de acceso a datos generadas por los ciudadanos. Por otra parte, debe ser posible finalizar dichas solicitudes y se debe notificar al ciudadano que su solicitud ha sido finalizada.

También el sistema debe notificar al administrador de cada organismo cuando existan solicitudes que estén próximas a alcanzar el período máximo de cinco días hábiles según lo establece la Ley N° 18.331.

Ciudadano

RFC-1 Consulta y modificación de consentimientos

El sistema debe permitir al ciudadano ver, modificar y/o eliminar los consentimientos brindados sobre sus datos personales a los distintos organismos.

RFC-2 Solicitudes de acceso a datos

El sistema debe permitir al ciudadano crear solicitudes de acceso a datos. Cada solicitud está dirigida a un organismo del cual el ciudadano desea obtener sus datos personales.

Además le debe permitir ver el estado de sus solicitudes tanto las pendientes como las finalizadas.

Requerimientos relacionados al intercambio de mensajes

RFS-1 Verificación de políticas de seguridad

El sistema debe interceptar todos los mensajes intercambiados entre los organismos que llegan a la PGE y realizar las validaciones correspondientes para determinar si dichos mensajes satisfacen las restricciones de seguridad definidas por la Ley N° 18.331.

La verificación se realiza teniendo en cuenta los organismos que participan en el intercambio de los datos personales, el ciudadano dueño de los mismos, la finalidad bajo la cual se desean intercambiar los datos, la fecha de envío del mensaje y los consentimientos brindados por el titular de los datos.

En caso de encontrar errores, se deben ejecutar las acciones pre-configuradas por el administrador del sistema.

Requerimientos no Funcionales

RNF-1 El prototipo debe ser implementado en lenguaje Java y utilizar servidor JBoss.

4 Solución Propuesta

En este capítulo se presentan las principales características de la solución propuesta denominada *Clouseau*. Esta aplicación interactúa con la PGE monitoreando los mensajes intercambiados por los organismos y tomando acciones sobre los mismos. En las siguientes secciones se detalla su arquitectura y se describen sus componentes. Luego se explica la interacción entre sus componentes, en especial, la que se da entre los módulos PDP y PEP mediante mensajes XACML.

4.1 Arquitectura General

La solución propuesta consiste en un componente que se incluirá en la PGE y se encargará de manejar los tres tipos de usuarios identificados en el capítulo anterior. Por un lado permitirá al administrador del sistema configurar los aspectos necesarios para el control de mensajes intercambiados entre organismos así como también monitorear el sistema. Por ejemplo será necesario crear en el sistema los organismos que se desean controlar. A su vez, permitirá a los administradores de cada organismo manejar los consentimientos de cada ciudadano y gestionar los pedidos de información creados por los mismos. Por último, cada ciudadano será capaz de gestionar mediante un sitio *web* sus consentimientos y solicitar a cada organismo la obtención de sus datos.

La idea general de *Clouseau* es interceptar todos los mensajes que se reciben y se envían desde y hacia la PGE. Cada mensaje será examinado por la aplicación la cual verificará su contenido teniendo en cuenta los aspectos de la Ley N° 18.331 previamente analizados. Se realizarán validaciones de los tipos de datos personales compartidos teniendo en cuenta los consentimientos disponibles para su utilización y las configuraciones realizadas. Como resultado de ese análisis se realizarán distintas transformaciones al mensaje original y eventualmente se tomarán distintas acciones pre-configuradas.

Para poder realizar las tareas antes mencionadas el sistema deberá almacenar un conjunto de datos. Entre otras cosas se almacenarán los consentimientos brindados por cada ciudadano, la configuración de cada *web service* expuesto por los distintos organismos y las distintas acciones a tomar. En la Figura 10 se muestra un esquema conceptual de la solución propuesta.

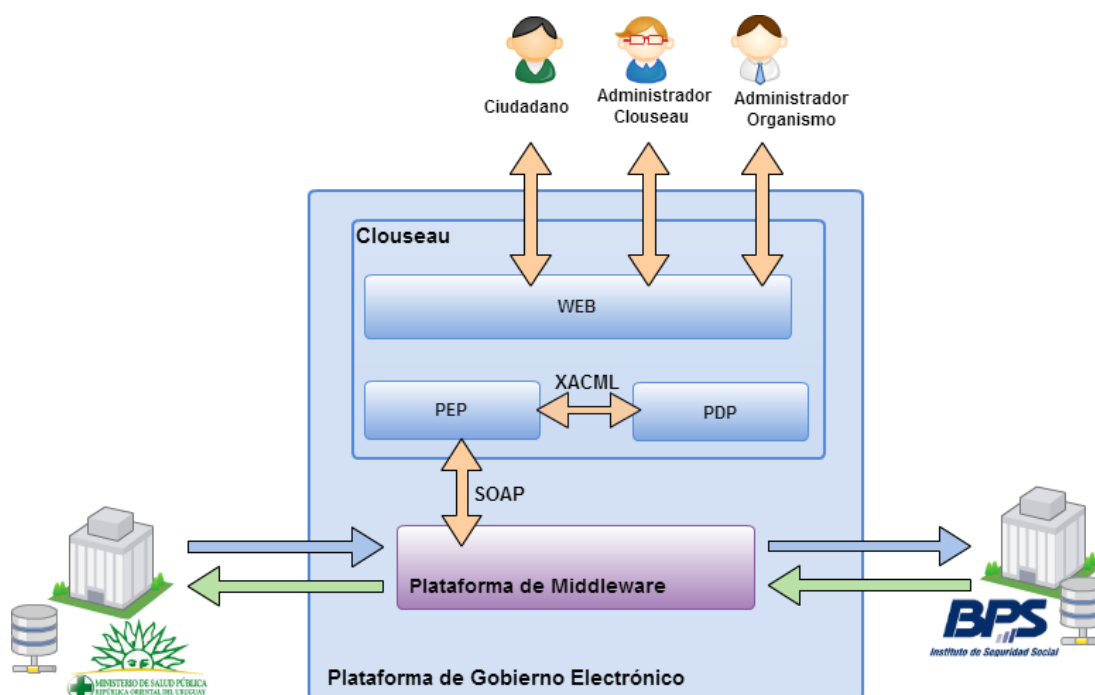


Figura 10 - Solución General

En la figura se presenta un ejemplo de comunicación entre el Ministerio de Salud Pública (MSP) y el Banco de Previsión Social (BPS). Como se mencionó anteriormente dicha comunicación se da a través de la Plataforma de Middleware de la PGE. La solución de basa en incluir un módulo dentro de la PGE por el cual pasan todos los mensajes intercambiados entre organismos. Este módulo consta básicamente de tres componentes. Un componente PEP el cual es el encargado de recibir los mensajes y enviar la solicitud de acceso XACML al componente PDP para la evaluación de consentimientos. Un componente PDP el cual se encarga de evaluar la petición XACML y generar la respuesta al módulo PEP en base a los consentimientos disponibles. Por último, se cuenta con un componente *web* mediante el cual el administrador del sistema, los administradores de cada organismo y los ciudadanos pueden configurar y gestionar distintos aspectos del sistema.

4.2 Principales Características

A continuación de presentan algunos de los aspectos más importantes de la solución propuesta.

Intercepción de requests

Uno de los principales requerimientos del sistema es el análisis de todos los mensajes que se intercambian entre los distintos organismos en la PGE. Para llevar este requerimiento a cabo se optó por exponer un *endpoint* en el cual se reciben todos los mensajes intercambiados, tanto los *requests* como los *responses*. Como respuesta a ese análisis se retornará un mensaje que cumpla con las validaciones necesarias o un error dependiendo del caso.

Configuración de web services

Para poder evaluar los distintos tipos de mensajes intercambiados en la PGE era necesario conocer los detalles de los *web services* disponibles. También era necesario conocer qué organismos proveía cada uno de ellos. Para solucionar este problema se optó por crear un *backoffice* de administración en el cual, entre otras cosas, se pueda configurar cada *web service* de la PGE. Analizando el conjunto de *web services* que se encuentran en el catálogo de servicios [32] brindado por AGESIC se observó que cada *web service* está compuesto por un conjunto de métodos. A partir de esto se decidió que la configuración en el sistema sea a nivel de métodos.

Para la configuración de un nuevo método el administrador deberá indicar a qué organismo pertenece y subir el archivo WSDL correspondiente.

Tipos de Elementos

Cada vez que se expone un nuevo método en la PGE, el mismo debe ser registrado en el sistema. De cada método se deben especificar datos básicos como su nombre y organismo al cual pertenecen pero fundamentalmente se deben especificar qué elementos del método se desean verificar. Para eso es posible clasificar sus elementos en una de las siguientes categorías:

- *Free*: Este tipo de elementos no requieren consentimientos para ser compartidos.
- *Limited*: Es necesario contar con el consentimiento explícito del titular para compartir este tipo de datos.
- *Denied*: Este tipo de datos nunca es compartido.

Este mapeo permite, dado un mensaje, identificar qué elementos del mismo deben ser verificados a la hora de comprobar los consentimientos disponibles.

Transformaciones y acciones a tomar

Como se mencionó en la sección de arquitectura general, cuando se detecta que se quiere compartir datos de tipo "LIMITED" sin el consentimiento explícito del titular de los datos, se realizarán distintas transformaciones al mensaje y se tomarán determinadas acciones. Se propone como solución que, para cada método configurado en el sistema, sea posible agregar una transformación XSLT la cual será aplicada al mensaje en caso de no contar con los consentimientos necesarios. Una posible transformación podría ser filtrar del mensaje los datos no permitidos por el titular de los datos. En la Figura 11 se observa un ejemplo en el cual se filtran los datos personales Documento y Sexo.

Adicionalmente se podrá seleccionar entre un conjunto de acciones disponibles en el sistema las cuales se ejecutarán en los mismos casos que la transformación. Un ejemplo de acción podría ser la notificación vía mail de la falta de consentimientos.

Además se decidió implementar la creación de acciones de forma extensible para que se puedan registrar nuevas acciones en el sistema cuando se desee.

El conjunto de acciones a ejecutar y el archivo con la transformación XSLT se podrán configurar al momento de dar de alta cada método.



Figura 11 - Ejemplo de filtrado de mensaje

Aplicación Web

Se consideró importante que el ciudadano cuente con una aplicación *web* en la cual pueda consultar y gestionar sus consentimientos brindados a los distintos organismos. De esta forma el ciudadano puede tener un control mayor sobre sus datos personales y por ejemplo dejar de compartirlos cuando desee. También desde la *web* podrá crear solicitudes de accesos a datos sin necesidad de concurrir al organismo correspondiente. Estas solicitudes se crearán en cada organismo y una vez finalizadas se le notificará vía mail al usuario que la misma ha finalizado.

Modelo de Datos Canónicos (MDC)

Dado que cada organismo utiliza sus propias *tags* para representar los datos personales de los ciudadanos, fue necesario diseñar algún mecanismo que permita unificar esa diversidad de nomenclaturas. Por ejemplo, el *tag* que contiene el nombre de un ciudadano puede llamarse “nomPersona” en un método de un *web service* de un organismo y “nombre” en otro organismo o incluso en otro método del mismo organismo. Para resolver este problema se decidió crear un modelo de datos canónicos para representar los datos personales, siguiendo el patrón Modelo de Datos Canónicos y teniendo en cuenta el modelo canónico de Persona definido y utilizado por AGESIC [33].

Para poder realizar ese modelo se optó por tener almacenado en el sistema un conjunto de datos personales definidos y a la hora de dar de alta un nuevo método

se debe especificar, para los elementos deseados, a qué dato personal corresponden. En la Figura 12 se puede observar un ejemplo de mapeo entre elementos de un mensaje y datos personales. Este mapeo será realizado por el administrador del sistema al momento de dar de alta un nuevo método.

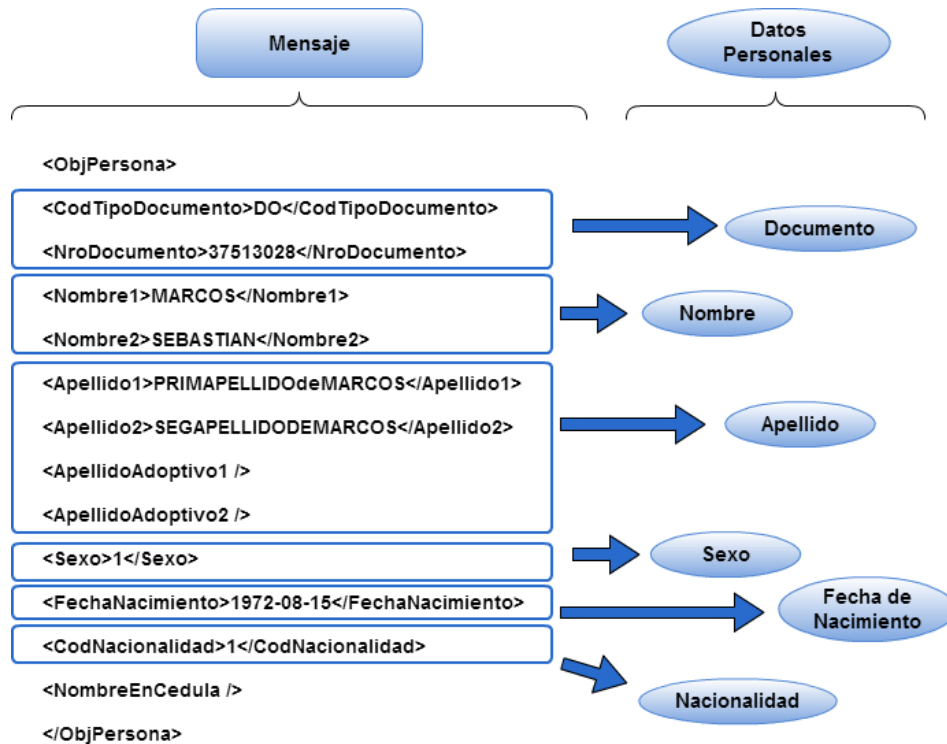


Figura 12 - Correspondencia entre elementos y datos personales

Cálculo de consentimientos

Cuando un ciudadano desea realizar un trámite en un organismo puede ser que el mismo requiera la invocación de uno o más métodos de varios *web services*. A su vez cada uno de estos métodos puede requerir que el ciudadano brinde distintos consentimientos para el pasaje de sus datos personales. Para resolver este problema se decidió contar con una funcionalidad de cálculo de consentimientos.

La idea es que dado un ciudadano y una finalidad el administrador de un organismo pueda obtener los consentimientos que hacen falta en el sistema para poder realizar la finalidad deseada. Para poder calcular los consentimientos faltantes se obtiene la lista de datos personales que intervienen en todos los métodos de la finalidad y se verifica que el ciudadano haya dado el consentimiento para utilizar esos datos personales. De cada dato personal se busca un consentimiento para los organismos involucrados y que el consentimiento esté dentro del período de validez.

A partir de esta verificación se obtiene la lista de consentimientos faltantes y se brinda la posibilidad al administrador de agregarlos luego de contar con la aprobación explícita del ciudadano.

Validez de Consentimientos

De cada consentimiento es posible especificar una fecha de comienzo y otra de fin de su validez. Bajo este contexto, dado que los consentimientos pueden poseer un período de validez, es posible que existan almacenados en el sistema consentimientos que ya se encuentran expirados. En consecuencia, se implementó un purgador que se encarga de eliminar consentimientos ya expirados de la base de datos. Los detalles de su implementación se encuentran en el capítulo 5.

4.3 Diseño de Componentes

Clouseau cuenta con los siguientes módulos: Clouseau-web, Clouseau-service, Clouseau-repo, Clouseau-flow y Clouseau-service-flow. Se decidió separar la aplicación en estos cinco módulos para tener un bajo nivel de acoplamiento y separarlo teniendo en cuenta el grupo de responsabilidades de cada uno.

A continuación se detallan los componentes de la solución y la función de cada uno de ellos. En la Figura 13 se muestra el diagrama de componentes.

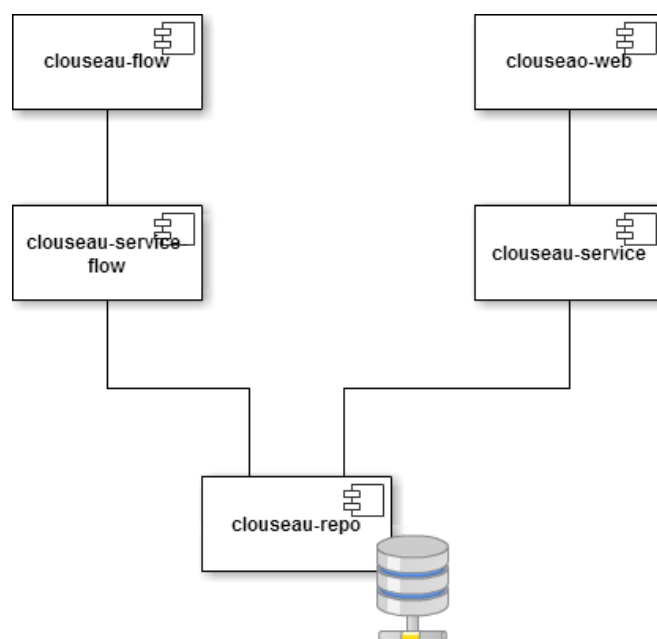


Figura 13 - Diagrama de Componentes

A. Clouseau-web

Es el componente *web* de la solución. El mismo será utilizado por los administradores para tareas de configuración y monitoreo y por los ciudadanos para la gestión de sus consentimientos.

B. Clouseau-service

Contiene la lógica de negocio utilizada por la aplicación *web*. Se comunica con el módulo *repo* para el manejo de datos.

C. Clouseau-repo

Este módulo se encarga de manejar el acceso a la base de datos.

D. Clouseau-service-flow

Es quien contiene la lógica relacionada con la verificación de consentimientos. Es el componente PDP de la solución ya que es quien evalúa los consentimientos disponibles y emite una decisión de autorización o no de los mensajes. Se comunica con el módulo repo para acceder a los consentimientos almacenados en la base de datos.

Para evaluar los consentimientos disponibles se utilizan los datos provenientes del módulo PEP. Estos datos son el documento del ciudadano, el organismo origen y destino del mensaje y la finalidad para la cual quiere ser utilizado el dato personal. Este módulo obtiene los consentimientos del ciudadano que coinciden con los datos anteriores y se encuentran dentro del período de validez. A partir de los mismos toma la decisión de autorización que es enviada como respuesta al módulo PEP.

E. Clouseau-flow

Es el componente PEP de la solución. Está basado en ESB y consta de un *endpoint* a través del cual se interceptan todos los mensajes SOAP que llegan a la PGE. Está basado en los patrones “*Routing Slip*” y “Ruteo basado en contenido”. En el capítulo de implementación se detalla el funcionamiento de este componente.

4.4 Interacción entre Componentes

Como se comentó en la descripción de componentes, los módulos Clouseau-flow y Clouseau-service-flow cumplen la función de PEP y PDP del estándar XACML respectivamente. En la Figura 14 se encuentra un diagrama de secuencia en el cual se muestra la interacción que se da entre estos componentes.

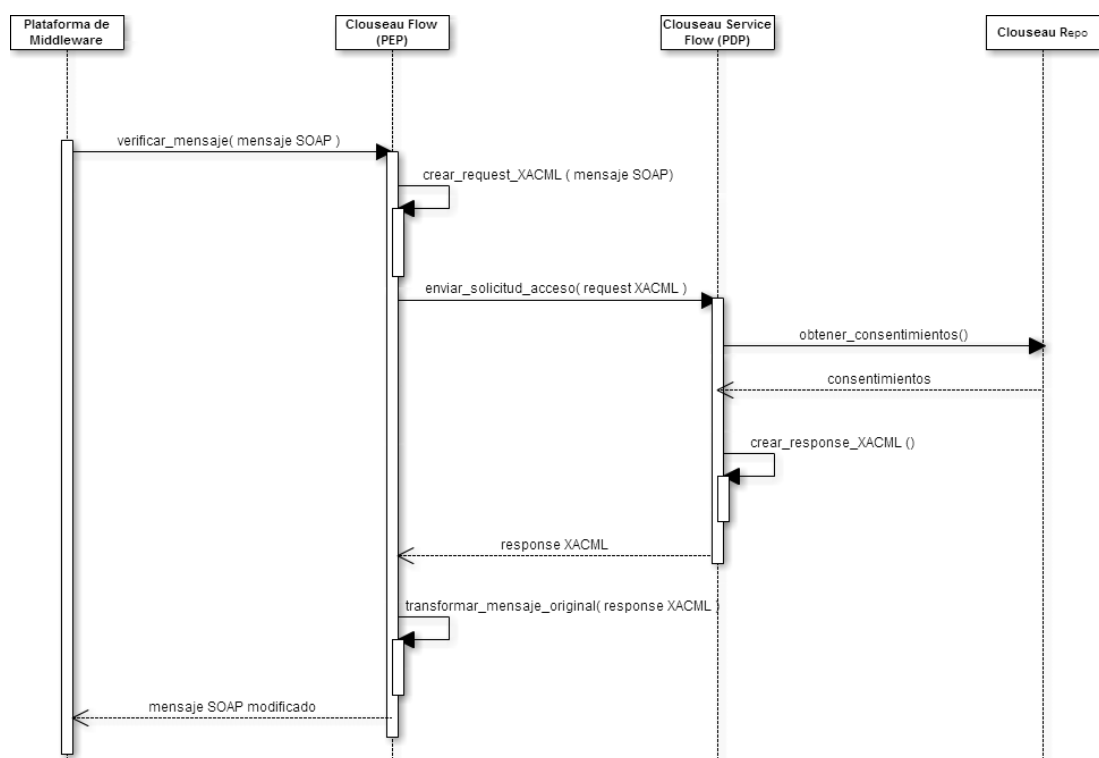


Figura 14- Diagrama de Secuencia

A continuación se describe a alto nivel cada uno de los pasos que se realizan a la hora de procesar un mensaje:

1. El mensaje a procesar es enviado al módulo Clouseau-flow (PEP)
2. El módulo Clouseau-flow valida la estructura del mensaje y a partir del mismo construye un *request* XACML.
3. El *request* XACML previamente construido es enviado al módulo Clouseau-service-flow para obtener una respuesta de acceso.
4. En módulo Clouseau-service-flow obtiene del módulo Clouseau-repo los consentimientos disponibles en el sistema y en base a éstos construye una respuesta XACML.
5. La respuesta XACML es enviada al módulo Clouseau-flow el cual a partir de la misma realizará las transformaciones necesarias al mensaje.
6. El mensaje modificado es retornado hacia la plataforma de middleware.

Como se explicó anteriormente, la comunicación entre los módulos Clouseau-flow y Clouseau-service-flow se da a través de mensajes XACML. A continuación se muestra cómo se utilizan cada uno de los elementos de los mensajes XACML en nuestra solución.

En el *request* se utilizan los elementos de la siguiente forma:

- **Subject:** es la entidad que solicita el acceso. En nuestro caso será el identificador del organismo que desea enviar el mensaje.

- *Resource*: especifica información sobre el recurso para el que se solicita el acceso, en nuestro caso el recurso es el propio mensaje que se desea enviar.
- *Action*: especifica la acción que se realizará sobre el recurso. En este prototipo la acción siempre será “Send” ya que lo que se desea es enviar el mensaje.
- *Environment*: es un conjunto de atributos que son relevantes para la decisión de autorización independiente de *Subject*, *Resource* o *Action*. Se utilizará esta sección para indicar el propósito por el cual se requiere enviar el mensaje, identificador del organismo destino del mensaje y documento del ciudadano del cual se quiere enviar datos.

En cuanto al response, estos son los elementos que se utilizaron:

- *Decision*: decisión de autorización. Se utilizan los valores “Permit” y “Deny” definidos en el estándar XACML.
- *Status*: Indica si se han producido errores durante la evaluación de la solicitud de decisión y opcionalmente información acerca de esos errores.
- *Obligations*: lista de operaciones que deben ser ejecutadas por el PEP en relación a una decisión de autorización. En esta sección agregamos las acciones que se deben realizar en caso de que no se cumpla con alguno de los consentimientos necesarios para enviar el mensaje. Por ejemplo, el envío de un mail notificando la falta de consentimientos. Además se agrega el nombre de la transformación XSLT que se debe realizar al mensaje si es necesario. Por último se envía la lista de parámetros que no pasaron la evaluación de consentimientos.

En el Apéndice 1. Mensajes XACML se muestra un ejemplo de *request* y de *response* XACML.

4.5 Integración con la PGE

Para el diseño de la solución se asumió que los mensajes SOAP que llegan a la PGE poseen un cabezal WS-Security donde se indica el organismo emisor y un cabezal WS-Addressing donde se especifica el *web service* y el método invocado.

Además se definió un cabezal denominado “Clouseau-Info” el cual es requerido y contiene la finalidad bajo la cual se está invocando el *web service*, el documento y el tipo de documento del ciudadano propietario de los datos personales incluidos en el cuerpo del mensaje SOAP. En la Figura 15 se muestra un ejemplo de este cabezal.

```
<cl:clouseauInfo xmlns:cl='http://schemas.xml.clouseu'>
  <cl:purpose>T201</cl:purpose>
  <cl:document>4728398</cl:document>
  <cl:type_document>CI</cl:type_document>
</cl:clouseauInfo>
```

Figura 15- Cabezal Clouseau-Info

Para poner en funcionamiento la solución en AGESIC se deberá incluir Clouseau como un módulo dentro de la Plataforma de Middleware de la PGE. Además se deberá modificar la estructura actual de los mensajes que llegan a la PGE añadiendo el cabezal "Clouseau-Info" con la información antes mencionada. En la Figura 16 se puede observar la estructura modificada de los mensajes.



Figura 16 - Mensaje SOAP

Por otro lado se asume que previo a que el mensaje llegue a Clouseau, en el Sistema de Seguridad de la PGE, se realiza un control de acceso por método. Es decir que se verifica que el usuario del organismo origen del mensaje posee autorización para invocar dicho método.

Una vez realizadas las validaciones de seguridad correspondientes los mensajes deberán ser redirigidos a Clouseau, donde se verificará el cumplimiento de las restricciones definidas por la Ley N° 18.331. En función del resultado retornado, los mensajes seguirán con el itinerario preestablecido dentro de la Plataforma de Middleware o serán enviados al organismo emisor indicando el error correspondiente.

5 Implementación

En este capítulo se presentan las herramientas utilizadas en el prototipo. De cada una de ellas se mencionan sus características y sus principales ventajas. Luego se destacan algunos aspectos propios de la implementación. Se plantean también algunos problemas encontrados y se describe cómo fueron solucionados.

5.1 Herramientas Utilizadas

Como se señaló anteriormente, en esta sección se presenta una breve descripción de las tecnologías utilizadas en la implementación del prototipo. Debido al requerimiento no funcional RNF-1 se implementó en JavaEE 7 y el *deploy*³ se realizó sobre la plataforma JBossEAP. La implementación de la solución se basó fuertemente en SwitchYard y para el componente *web* se utilizó fundamentalmente JSF y Bootstrap. En la Figura 17 se encuentra un esquema con los componentes implementados y las principales herramientas utilizadas en cada uno de ellos.

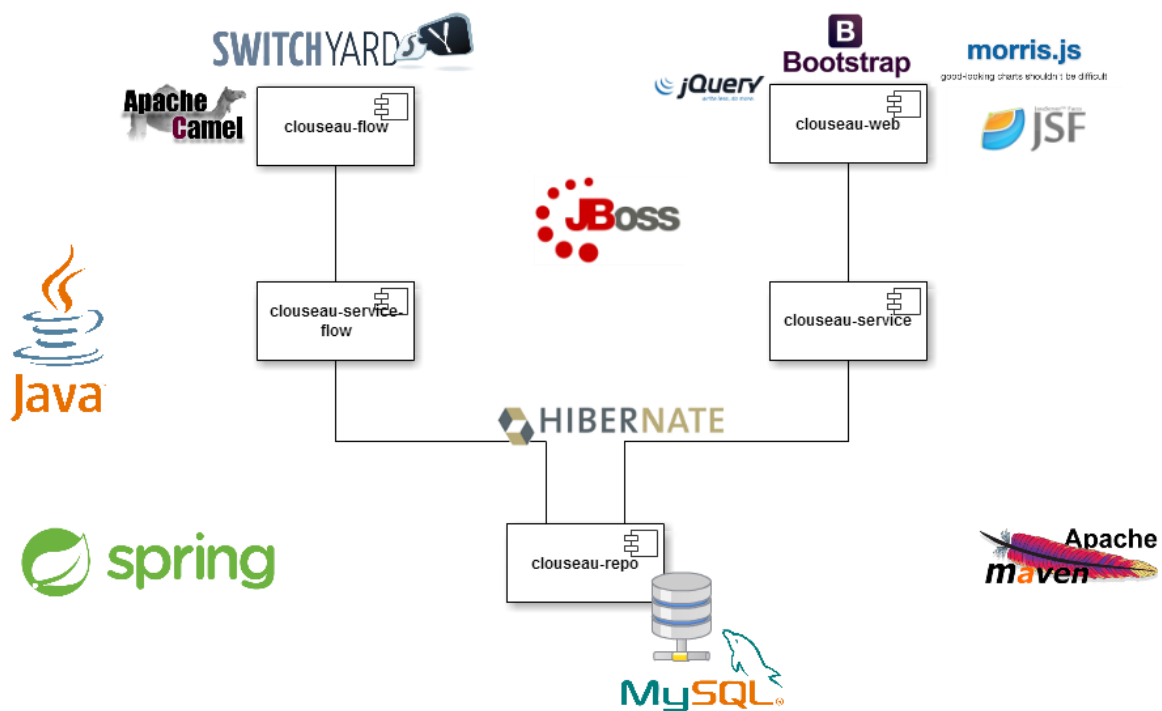


Figura 17 - Herramientas utilizadas

Spring Framework

Spring Framework [34] es una plataforma que facilita el desarrollo de aplicaciones Java y surgió como alternativa a los *Enterprise JavaBeans* (EJBs). En particular se

³ Implantación en un servidor.

utilizó para resolver el problema del manejo del ciclo de vida de los objetos, la inyección de dependencias y el manejo de la transaccionalidad.

Su utilización permitió lograr un prototipo con capas bien delimitadas y buenas prácticas de programación. Se utilizó la versión 4.1.1 de Spring.

Spring está formado por una serie de módulos lo cual hace que no sea necesario agregar todos sino solo los que sean necesarios. En este proyecto se utilizaron los módulos *Core* para iniciar los objetos y manejar sus ciclos de vida; el módulo *Context* para acceder a los *beans*⁴; Spring DAO para acceder a datos almacenados y el módulo ORM para la integración con Hibernate y manejar las transacciones.

Inversión de control e inyección de dependencias

En Spring existen dos formas de utilizar inyección de dependencias, declarando en un XML los componentes de la aplicación y sus dependencias o mediante anotaciones. En este proyecto se utilizó la inyección mediante anotaciones. En la Figura 18 podemos ver como se inyecta la clase *AdministrationDAO* utilizando la anotación *@Autowired*.

```
@Service
@Transactional(propagation = Propagation.REQUIRED)
public class AdministratorBean {

    private static final String EMAIL = "email";

    @Autowired
    private AdministratorDAO administratorDAO;

    @Autowired
    private OrganizationDAO organizationDAO;

    @Transactional
    public void create(AdministratorDTO administratorDTO) {

        Administrator administrator = new Administrator();
        administrator.setEmail(administratorDTO.getEmail());
        administrator.setEncryptedPassword(administratorDTO
            .getEncryptedPassword());
        administrator.setSalt(administratorDTO.getSalt());
        administrator.setActive(true);
    }
}
```

Figura 18 - Anotaciones de Spring

Transaccionalidad

En este prototipo manejamos la transaccionalidad utilizando la anotación *@Transactional* a nivel de clases. Spring ofrece varios tipos. En particular, se utilizó *Propagation=REQUIRED*. Si el método es invocado desde un contexto transaccional, entonces el método será invocado en el mismo contexto transaccional. Si el método no es invocado desde un contexto transaccional, entonces el método creará una nueva transacción e intentará hacer *commit* de la transacción cuando el método termine su ejecución.

⁴ Componente de *software* reutilizable.

Hibernate

Hibernate [35] es una herramienta que se encarga del mapeo entre objetos y el modelo de base de datos. Además facilita el manejo de consultas y recuperación de datos. Su configuración fue muy sencilla. Solo se especificó en un archivo XML los datos de conexión para la base de datos y el *driver*⁵ utilizado. También se especificó en el archivo cuales son las clases que se mapean con entidades de la base de datos. En el prototipo desarrollado se utilizó la versión 4.2.0 y se utilizó en el componente Clouseau-repo para el acceso a la base de datos y el mapeo de entidades.

MySQL

Es un servidor de bases de datos SQL. MySQL [36] es un proyecto *Open Source* y se optó por ésta herramienta ya que se caracteriza por su gran velocidad a la hora de realizar las operaciones y su bajo costo en requerimientos para la elaboración de base de datos.

JavaServer Faces (JSF)

JSF [37] es un *framework* y estándar utilizado para implementar interfaces de usuario del lado del servidor de aplicaciones *web* basado en Java. Proporciona facilidades para la representación de componentes UI y manejar su estado; manejar eventos; validación del lado del servidor y conversión de datos; definir navegación entre páginas y manejo de errores entre otras cosas. Su principal beneficio es que permite separar la lógica de negocio de las vistas y proporciona un método fácil para enlazar ambas partes.

Se decidió utilizar JSF ya que se integra muy fácilmente con el componente *web* de Spring y juntos proporcionan una clara separación entre las vistas y los controladores. En este prototipo se utilizó la versión 2.2.8 de JSF para el manejo de los *beans* que se encargan de controlar las páginas *web*. JSF es capaz de crear las instancias,

controlar sus ciclos de vida y llamar a los métodos *getters* y *setters* ahorrando tiempo de desarrollo.

Bootstrap

Esta herramienta hace que el desarrollo de aplicaciones *front-end*⁶ sea más fácil y rápido. Permite construir páginas *web responsives*⁷ sin mucho esfuerzo y cuenta con varios componentes ya diseñados. Bootstrap [38] es de código abierto y es compatible con los principales navegadores *web*. Se utilizó esta herramienta para el desarrollo del sitio *web* del sistema.

⁵ *Software* que permite la comunicación entre un sistema y un dispositivo.

⁶ Componente del *software* que interactúa con el o los usuarios.

⁷ Diseño web adaptable.

Maven

Apache Maven [39] es una herramienta que permite administrar proyectos de *software* desde su creación hasta la generación de un binario del mismo. Brinda la posibilidad de generar proyectos modulares en lugar de un único proyecto monolítico. En general se cuenta con un proyecto padre y varios hijos. Un proyecto en Maven se define mediante un archivo POM (*Project Object Model*). En particular se utilizó Maven para definir aspectos sobre la compilación del proyecto, la estructura modular del mismo y las librerías utilizadas. En la Figura 19 se muestra la estructura elegida para el prototipo realizado.

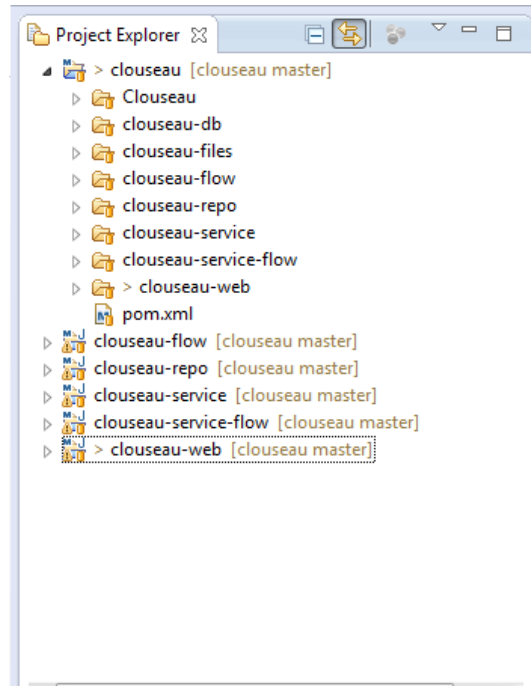


Figura 19 - Estructura modular del prototipo

Apache Camel

Apache Camel [40] es un *framework* Java de código abierto enfocado en facilitar la integración de aplicaciones sobre distintas plataformas.

Para ello provee implementaciones concretas de los Patrones de integración Empresariales (EIPs) más utilizados. Además brinda facilidades a la hora de utilizar Domain Specific Language (DSL) [41].

En particular para este proyecto se utilizaron los patrones Ruteo Basado en Contenido y *Routing Slip*. Además de utilizó la *interfaz endpoint* para recibir los mensajes a procesar por el sistema.

SwitchYard

SwitchYard [51] es un *framework* diseñado para proporcionar soporte durante la implementación y administración de aplicaciones SOA. Contiene todas las

características esperables en un ESB, además está integrado con Apache Camel añadiendo a JavaEE funcionalidades de ruteo y transformación de mensajes y orquestación de servicios. Es desplegado sobre JBoss EAP.

SwitchYard permite transformar rápidamente una clase java en un *web service*. El prototipo implementado cuenta con un *endpoint* SOAP que expone un servicio con el método “processMessage”. Es decir recibe un mensaje SOAP, y retorna dicho mensaje transformado en caso de ser necesario.

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

Ruteo de Mensajes

Para realizar el ruteo de los mensajes en el ESB se utilizaron principalmente dos patrones de ruteo: Ruteo Basado en Contenido y *Routing Slip*.

En la Figura 20 se muestra un ejemplo de ruteo basado en contenido utilizando Java DSL.

```
public class MessageValidatorRouter extends RouteBuilder {
    @Override
    public void configure() {
        from("switchyard://ClouseauService")
            .choice()
            .when()
                .method("MessageValidatorBean", "isCorrect")
            .to("switchyard://PEPService")
            .otherwise()
            .to("switchyard://ErrorManagerService");
    }
}
```

Figura 20 -Ruteo SwitchYard

En el ejemplo se define el *router* “MessageValidatorRouter”, que intercepta todos los mensajes recibidos por el *endpoint* “switchyard://ClouseauService” y utiliza el patrón de ruteo Basado en Contenido para redirigir el mensaje al *endpoint* “switchyard://PEPService” en caso de que el mensaje sea estructuralmente válido y que se continúe con el procesamiento, o redirige el mensaje a “switchyard://ErrorManagerService”, donde se añade un mensaje de error indicando lo ocurrido.

En la Figura 21 se muestra un ejemplo de *Routing Slip* utilizando Java DSL.

```

public class ActionRouter extends RouteBuilder {
    @Override
    public void configure() {
        from("switchyard://ActionService")
            .routingSlip(bean("RouteActionBean", "calculateRoute"));
    }
}

```

Figura 21- Routing Slip SwitchYard

En el ejemplo se define el router “ActionRouter”, que intercepta todos los mensajes recibidos por el endpoint “switchyard://ActionService”. “RouteActionBean” se encarga de calcular la lista de *endpoints* que debe recorrer el mensaje. A modo de ejemplo asumimos que el componente PDP retornó como *Obligations* que se debe ejecutar la transformación XSLT incluida en el archivo “Transformer.xml” y la acción “Email” por lo que el método “calculateRoute” retornará lo siguiente: “xslt:file:TransformerFilter.xml, switchyard://EmailActionService”. A partir del resultado anterior y utilizando *Routing Slip* el mensaje recorrerá secuencialmente la lista de *endpoints* especificada en su cabecal. En este caso al mensaje se le aplicaría la transformación XSLT y luego se enviará el email correspondiente.

Todos los *endpoints* mencionados previamente fueron definidos como interfaces Java.

Transformaciones XSLT

XSLT [15] es un lenguaje especificado por W3C para realizar transformaciones sobre documentos XML. Se basa en expresiones XPath [42] para seleccionar y examinar la estructura del documento de entrada de la transformación. SwitchYard permite nativamente ejecutar transformaciones XSLT, a partir de archivos XSL.

En el Apéndice 2. Transformaciones XSLT se encuentran dos transformaciones implementadas a modo de ejemplo.

Morris.js

Morris.js [43] es una librería basada en JQuery especializada en la construcción de gráficas lineales y de barras entre otras. Fue utilizada para la implementación del *Dashboard* de Administración. Se optó por esta librería ya que permite fácilmente la creación de gráficas agradables estéticamente.

JBoss XACML

Para la creación de mensajes XACML se utilizó la librería JBoss XACML [44]. La misma provee un modelo de objetos para construir mensajes basados en dicho estándar. Se utilizó tanto en el módulo PEP para la construcción de *requests* XACML como en el PDP para su interpretación y para la construcción de *responses* XACML.

5.2 Detalles de Implementación

En esta sección se presentan detalles sobre la implementación de alguna de las funcionalidades. Se describen tanto aspectos técnicos como problemas encontrados a la hora de implementarlos y como fueron resueltos. Cabe destacar que se implementaron la totalidad de los requerimientos del capítulo 3.

ESB

Como ya se mencionó Clouseau-flow es el componente ESB de la solución. El mismo consta de un *endpoint* por el cual se interceptan todos los mensajes SOAP que llegan a la Plataforma de Middleware de PGE. Funciona como PEP ya que es quien realiza las solicitudes de decisión de autorización y se encarga de que se cumplan las decisiones tomadas.

En la Figura 22 se muestra un esquema con los sub-componentes del módulo.

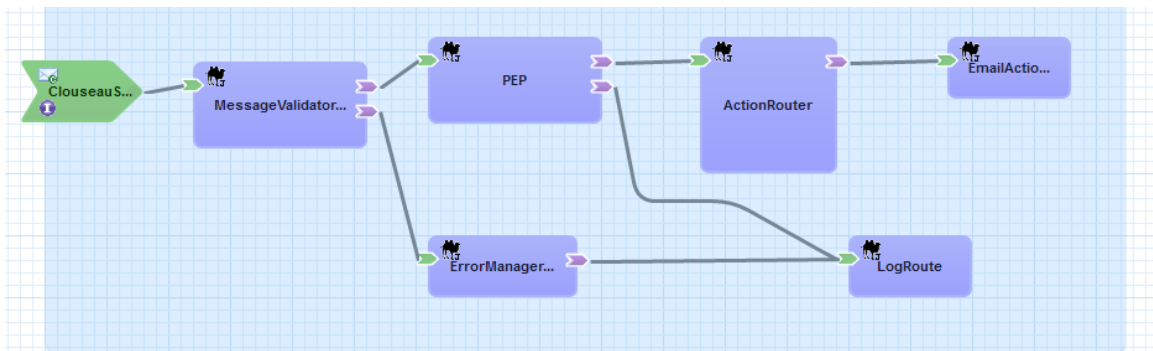


Figura 22 - Flujo ESB

A continuación se describe cada uno de sus componentes.

Message Validator Router

Es el encargado de verificar si el mensaje es estructuralmente válido, es decir si posee todos los datos necesarios para realizar el proceso de validación de permisos.

Un mensaje es estructuralmente válido si contiene un cabezal *WS-Security* donde se especifique el código del organismo que desea compartir el dato personal.

Además debe contener un cabezal *WS-Addressing* donde se especifique la *URI* del *web service* invocado. También debe incluir el cabezal Clouseau-Info definido.

En caso de que el mensaje sea correcto es redirigido al componente “PEP”, de lo contrario es redirigido al componente “ErrorManagerRouter”.

Error Manager Router

Es el componente encargado de agregar al cuerpo del mensaje el error correspondiente. Luego el mensaje es redirigido al componente “LogRouter”.

PEP

Componente encargado de extraer del mensaje SOAP recibido los datos necesarios para crear el *request* XACML que será enviado a Clouseau-service-flow (PDP).

En la respuesta XACML del módulo PDP se especifica si los organismos involucrados (tanto el emisor como el receptor) están autorizados para intercambiar los datos personales incluidos en el mensaje. En caso de no estar autorizados se incluye en la respuesta las acciones a ejecutar.

Si el módulo PDP determina que los organismos poseen los consentimientos necesarios para el intercambio de los datos personales, se redirige el mismo al componente “LogRouter”. De lo contrario es redirigido al componente “ActionRouter”.

Action Router

Es el encargado de extraer del resultado generado por el módulo PDP las acciones que deben ser ejecutadas y la transformación XSLT que se debe aplicar al mensaje. Tanto las acciones como la transformación que se desea aplicar deben ser configuradas al momento de dar de alta un nuevo método en el sistema.

Una posible acción podría ser un componente que envíe un email. Y un ejemplo de transformación podría ser una que, dado un mensaje SOAP y una lista de parámetros, se retorne el mensaje recibido quitando los valores en las etiquetas cuyos nombres se encuentren en la lista recibida.

Para ejecutar la transformación y las acciones configuradas se utiliza el patrón de ruteo *Routing Slip*. Para ello se añade en el *header* del mensaje la lista de los componentes que debe recorrer.

En la Figura 23 se muestra un ejemplo del funcionamiento del patrón *Routing Slip*.

En el ejemplo se determina en tiempo de ejecución los componentes que el mensaje debe recorrer. Luego se añade al cabezal una lista con las URIs de dichos componentes (“A” y “C”). El cabezal del mensaje es procesado y recorre secuencialmente la lista de componentes indicada.

En el contexto de Clouseau un posible ejemplo de uso es, añadir al cabezal una lista que incluye, el nombre del archivo que posee la transformación XSLT, seguido de las URIs de los componentes que poseen la implementación de las acciones.

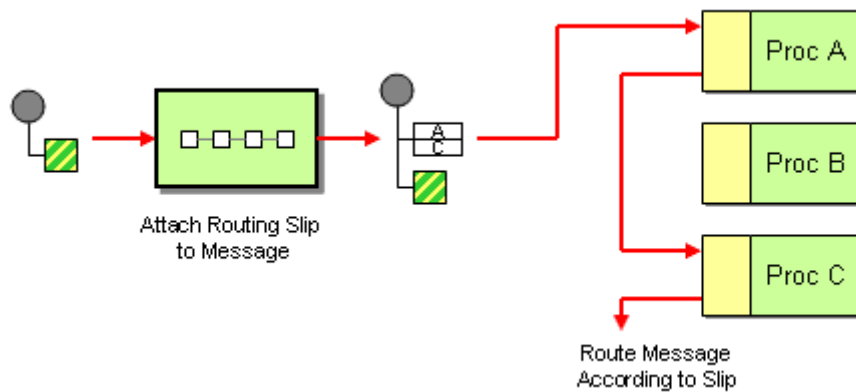


Figura 23 - Routing Slip [45]

Email Action

Componente encargado de enviar emails al administrador de Clouseau cada vez que se detecte la ausencia de consentimientos en los mensajes recibidos por el sistema.

Log Router

Componente encargado de registrar en el sistema datos de cada mensaje interceptado. Los datos más importantes almacenados son: organismo emisor, organismo destino, nombre del método invocado, mensaje recibido y mensaje retornado.

Parseo de archivos WSDL

A la hora de dar de alta un nuevo método en Clouseau, se cuenta con una interfaz que permite al administrador subir el archivo WSDL del *web service*. A partir del archivo se parsea su contenido y se obtienen automáticamente datos del *web service* como el identificador de sus métodos, su descripción y la lista de elementos que se utilizará para determinar cuáles de esos elementos corresponden a datos personales.

La obtención de datos directamente del archivo hace que sea más fácil para el administrador la creación de un nuevo método de un *web service* y en consecuencia reducir la posibilidad de errores manuales. Para el parseo del archivo WSDL se utilizó la librería *javax.wsdl*. Cabe destacar que en caso de que ocurra un error en el procesamiento del archivo WSDL se da la posibilidad al usuario de que complete los campos de forma manual y de esta forma no bloquear la creación de un nuevo método. En la Figura 24 se observa la interfaz para la creación de un nuevo método en el sistema.

Figura 24 - Alta de método

Creación de Acciones

Para cumplir con el requerimiento RBA-5 el cual hace referencia a la creación de acciones que se ejecutarán en caso de detectar la falta de consentimientos, fue necesario diseñar algún mecanismo que permita agregar nuevas acciones fácilmente. Principalmente se buscaba agregar acciones realizando la menor cantidad de cambios en la aplicación. Para resolver este problema se utilizó el soporte de *reflexion* provisto por Java.

Reflexion es la capacidad que tiene un programa para conocer y en ocasiones modificar su estructura de alto nivel. Cuando el código fuente de un programa se compila, pierde la información sobre su estructura ya que se genera código de bajo nivel. *Reflexion* nos permite, en tiempo de ejecución, volver a conocer la estructura de alto nivel.

A la hora de crear una nueva acción debemos agregar al proyecto el nuevo *bean* y gracias a *reflexion* se creará, en tiempo de ejecución, la instancia de este *bean* solo conociendo el nombre del mismo el cual se tomará de la base de datos. El único requisito es que los nuevos *beans* extiendan de la clase “ActionBean” el cual contiene un método denominado “getActionDTO()” que debe ser implementado. Este método es el encargado de generar un DTO con datos que se enviarán al módulo PEP en la respuesta XACML. Estos datos se utilizan en el componente PEP para ejecutar la acción. Por ejemplo si la acción es enviar un mail, en ese DTO se encontrará la lista de destinatarios del mail a enviar.

Del lado del componente PEP se debe crear una interfaz respetando la siguiente nomenclatura: NombreAccion + "ActionService". La interfaz debe contener la definición del método "String executeAction(String message)".

Además se deberá definir un *router* que intercepte todos los mensajes de la interfaz definida previamente.

Siguiendo con el ejemplo anterior en que la acción es enviar un mail el código del *router* se vería como se muestra en la Figura 25.

```
public class EmailActionRoute extends RouteBuilder {  
  
    @Override  
    public void configure() {  
        from("switchyard://EmailActionService").process(new Processor() {  
            @Override  
            public void process(Exchange exchange) throws Exception {  
                sendEmail();  
            }  
        });  
    }  
}
```

Figura 25 - Action Router

Scheduling Tasks

Para implementar el purgador de consentimientos expirados se utilizó el mecanismo de *schedule task* que brinda Spring. Con una simple anotación en el método deseado, Spring ejecuta esa tarea en intervalos de tiempo configurables mediante expresiones crons.

A pesar de que se implementó solo una *schedule task*, el diseño se realizó considerando la posibilidad de agregar más tareas fácilmente. Para esto se creó una tabla en base de datos donde se almacenan todas las tareas. En particular de cada tarea se almacena:

- Nombre
- Fecha de última ejecución de la tarea.
- Fecha de ultima ejecución exitosa de la tarea
- Estado (Activo/Inactivo)
- Ultimo mensaje de error obtenido

Estos datos permiten monitorear las tareas en caso de fallas así como también poder habilitar o deshabilitar su ejecución por base de datos cuando se desee, sin necesidad de tener que volver a *deployar* la aplicación. En la Figura 26 se encuentra el código del purgador.

```

@Scheduled(cron = "0 0/1 * * * ?")
public void purge() {

    CronData cronData = this.cronDataDAO.loadByProperty("name", CRON_NAME);

    if (cronData.isActive()) {
        cronData.setLastRun(new Date());
        try {

            this.purgeConsentBean.purgeConsents();
            cronData.setLastSuccess(new Date());

        } catch (Exception e) {
            cronData.setLastFailure(new Date());
            cronData.setLastFailureMessage(e.getMessage());
        }
        this.cronDataDAO.save(cronData);
    }
}

```

Figura 26 - Purgador de consentimientos expirados

Envío de emails

Tanto para notificar a los ciudadanos que su solicitud de pedido de datos ha sido finalizada como para notificar a los administradores que se está intentando enviar datos sin el consentimiento de su titular se utilizan emails. Un problema encontrado fue que el envío de emails insumía mucho tiempo en comparación con las otras tareas realizadas. Por ejemplo, a la hora de procesar un mensaje, el envío de un email hacia que su procesamiento tardara tiempos no tolerables.

Para resolver este problema el envío de emails se realizó de forma asincrónica, es decir, al momento de enviar un email se crea un hilo aparte encargado del envío del mismo pudiendo así continuar en paralelo con la ejecución de otras actividades.

Dashboard

El *back-end* de administración ofrece un *dashboard* donde es posible visualizar varias métricas sobre el tráfico de mensajes tomadas por el sistema. Como se mencionó anteriormente, el sistema guarda un registro con los datos más relevantes de cada mensaje SOAP recibido. Dicho registro es almacenado en una tabla en la base de datos.

Para la construcción del *dashboard* se realizan consultas sobre dicha tabla y se muestran en el sitio *web* utilizando la API Morris.js.

A modo de ejemplo la Figura 27 muestra el código javascript necesario para la creación de una gráfica de barra donde se muestra la cantidad de mensajes autorizados y no autorizados durante los últimos siete días.

```
var graphB = Morris.Bar({
  element: 'last-week',
  data: [ #{dashboardController.chartDataLastWeek}
],
  xkey: 'date',
  ykeys: ['a', 'n'],
  labels: ['AUTHORIZED', 'NOT AUTHORIZED']
});
```

Figura 27 - Código Dashboard

Los datos de la gráfica se obtienen del atributo “chartDataLastWeek” de “DashboardController”. Un posible valor de dicho atributo sería:

```
chartDataLastWeek = [{"date": '2015-02-17', a: 100, n: 9}
, {"date": '2015-02-18', a: 52, n: 32}
, {"date": '2015-02-19', a: 74, n: 62}
, {"date": '2015-02-20', a: 13, n: 5}
, {"date": '2015-02-21', a: 0, n: 0}
, {"date": '2015-02-22', a: 0, n: 0}
, {"date": '2015-02-23', a: 302, n: 152}
, {"date": '2015-02-24', a: 1, n: 2}];
```

Figura 28

Creación de *Request* XACML a partir de mensaje SOAP

Como se mencionó en la sección 4.4 el módulo PEP es el encargado de crear a partir del mensaje SOAP recibido, el *request* XACML que será enviado al módulo PDP. Este procesamiento sobre el mensaje es realizado utilizando la herramienta XPath.

```

<env:Envelope xmlns:env='...'>
  <env:Header xmlns:wsa='...'>
    <wsse:Security env:mustUnderstand='1' xmlns:ds='...'
      xmlns:wsse='...' xmlns:wsu='...'>
      <saml:Assertion AssertionID='...' IssueInstant='...'
        Issuer='...' MajorVersion='1' MinorVersion='1' xmlns:saml='...'>
        <saml:Conditions NotBefore='...' NotOnOrAfter='...'>
          <saml:AudienceRestrictionCondition>
            <saml:Audience>
              http://192.168.40.190:9000/wsServicioDeInformacionBasico
            </saml:Audience>
          </saml:AudienceRestrictionCondition>
        </saml:Conditions>
        <saml:AuthenticationStatement
          AuthenticationInstant='...' AuthenticationMethod='...'>
          <saml:Subject>
            <saml:NameIdentifier>
              uid=rolPruebaDoctor,cn=BPS
            </saml:NameIdentifier>
          </saml:Subject>
        </saml:AuthenticationStatement>
        <saml:AttributeStatement>
          <saml:Subject>
            <saml:NameIdentifier>
              uid=rolPruebaDoctor,cn=BPS
            </saml:NameIdentifier>
          </saml:Subject>
          <saml:Attribute AttributeName='User'
            AttributeNamespace='urn:nac'>
            <saml:AttributeValue>Juan</saml:AttributeValue>
          </saml:Attribute>
        </saml:AttributeStatement>
        <ds:Signature Id='uuid2765608c-0128-1428-9ce4-8913af9af38d'
          xmlns:ds='http://www.w3.org/2000/09/xmldsig#'> ...
        </ds:Signature>
      </saml:Assertion>
    </wsse:Security>
    <wsa:To>http://192.168.40.190:9000/wsServicioDeInformacionBasico</wsa:To>
    <wsa:Action>
      http://.../certificadoCNVEWSDLPortType/ObtPersonaPorDoc
    </wsa:Action>
    <cl:clouseulInfo xmlns:cl='http://schemas.xml.clouseu'>
      <cl:purpose>T201</cl:purpose>
      <cl:document>4728398</cl:document>
      <cl:type_document>CI</cl:type_document>
    </cl:clouseulInfo>
  </env:Header>
  <env:Body>
    <wsd:ObtPersonaPorDoc xmlns:wsd='http://wsDNIC/'>
      <wsd:paramObtPersonaPorDoc>
        <wsd:Nrodocumento>4728398</wsd:Nrodocumento>
        <wsd:TipoDocumento>CI</wsd:TipoDocumento>
      </wsd:paramObtPersonaPorDoc>
    </wsd:ObtPersonaPorDoc>
  </env:Body>
</env:Envelope>

```

Figura 29 - Mensaje SOAP

En la Figura 29 se muestra el mapeo de los elementos de un mensaje SOAP de la PGE a un *request* XACML.

A partir del Cabezal WS-Security de los mensajes SOAP de la PGE, se obtiene el código del organismo que desea compartir los datos personales.

A partir del cabezal WS-Addressing se obtiene el nombre del *web service* y del método invocado, permitiendo conocer así el organismo destino del mensaje el cual es enviado como un atributo del “*Environment*”.

Del cabezal de Clouseau, se extrae la finalidad bajo la cual se realiza la invocación y el tipo y número de documento del ciudadano propietario de los datos personales. Dichos datos son mapeados como atributos del “*Environment*” del *request* XACML. El *body* del mensaje será enviado como el “*Resource*” XACML.

Seguridad

Para proteger las contraseñas de usuarios y administradores éstas se guardan encriptadas en la base de datos usando el algoritmo PBKDF2. La implementación estándar de PBKDF2 utiliza SHA-1, un algoritmo de hash seguro. A pesar de que es un algoritmo muy seguro, una desventaja de usar SHA-1 es que es un algoritmo muy rápido y esa rapidez es una debilidad ya que los ataques por fuerza bruta pueden ejecutarse más rápidamente.

6 Casos de Estudio

En esta sección se presentan tres casos de estudios definidos para validar la solución planteada y mostrar el funcionamiento general del sistema.

6.1 Objetivo

El objetivo de los casos de estudios es validar el correcto funcionamiento del prototipo enfocándonos en tres de las principales funcionalidades del mismo. El primer caso permite validar el ciclo completo desde que se registra un método en el sistema hasta que el mismo es invocado y se analizan los mensajes intercambiados en la invocación. El segundo caso presenta el manejo de las solicitudes de acceso a datos creadas por los ciudadanos a través del portal *web*. Por último, en el tercero se analiza el *dashboard* de administración del sistema.

6.2 Validación de mensajes

Para este caso de estudio asumimos que ya fueron dados de alta en el sistema el conjunto de usuarios, administradores, organismos y finalidades utilizados. En este caso de estudio se utiliza el servicio básico de información provisto por la DNIC.

Comenzamos este caso de estudio dando de alta el nuevo método en el sistema. El servicio elegido consta de dos métodos: `ObtPersonaPorDoc` y `ProductDesc`. Utilizaremos el método `ObtPersonaPorDoc` el cual dado un número de cédula de identidad, retorna los datos filiatorios del mismo. En la Figura 30 podemos apreciar la página de creación de un nuevo método. En la misma el administrador debe subir el WSDL del *web service* y automáticamente se obtendrán los datos del método.

Nuevo Método

Paso 1 - Datos Básicos del nuevo Método

Cargue el WSDL del nuevo método

No se ha seleccionado ningún archivo.

Organismo

Fianalidad

WS

Método

Descripción

Transformación xsl

TransformerFilter.xsl

Acciones

☒ Email

Figura 30 - Alta Método

Además, es posible subir un archivo con una transformación XSLT a aplicar en caso de error y se puede seleccionar opcionalmente un grupo de acciones a realizar. En este caso se subió un archivo XSL que filtra los datos personales no permitidos y se seleccionó la acción “Email”.

En la siguiente página es donde el administrador debe determinar qué elementos del método corresponden a datos personales. La configuración de elementos y datos personales fue la siguiente:

Elemento	Dato Personal	Tipo de Parámetro	Tipo
TipoDocumento	documento	ENTRADA	LIMITED
NroDocumento	documento	SALIDA	LIMITED
CodTipoDocumento	documento	SALIDA	LIMITED
Nombre1	nombre	SALIDA	FREE
Nombre2	nombre	SALIDA	FREE
Apellido 1	nombre	SALIDA	FREE
Apellido2	nombre	SALIDA	FREE
ApellidoAdoptivo1	nombre	SALIDA	FREE
ApellidoAdoptivo2	nombre	SALIDA	FREE
Sexo	sexo	SALIDA	LIMITED
FechaNacimiento	fecha de nacimiento	SALIDA	LIMITED
CodNacionalidad	nacionalidad	SALIDA	DENIED

Figura 31 - Configuración de método


Nuevo Método

Paso 2 - Mapeo de Elementos / Datos Personales

Elemento	Tipo de Parámetro	Dato Personal	Tipo	
Apellido2	Salida	nombre	FREE	<button>Agregar</button>

Elementos mapeados

Elemento	Tipo de Parámetro	Dato Personal	Tipo	
NroDocumento	Salida	documento	LIMITED	<button>Eliminar</button>
TipoDocumento	Entrada	documento	LIMITED	<button>Eliminar</button>
CodTipoDocumento	Salida	documento	LIMITED	<button>Eliminar</button>
Nombre1	Salida	nombre	FREE	<button>Eliminar</button>
Nombre2	Salida	nombre	FREE	<button>Eliminar</button>
Apellido1	Salida	nombre	FREE	<button>Eliminar</button>
Apellido2	Salida	nombre	FREE	<button>Eliminar</button>

Finalizar

Figura 32 - Configuración de método vía web

A continuación se muestra un ejemplo de invocación del servicio utilizando la respuesta del método obtPersonaPorDoc.

<pre> <env:Body> <ObtPersonaPorDocResponse xmlns="http://ws.DNIC/"> <ObtPersonaPorDocResult> <ObjPersona> <CodTipoDocumento>DO</CodTipoDocumento> <NroDocumento>37513028</NroDocumento> <Nombre1>MARCOS</Nombre1> <Nombre2>SEBASTIAN</Nombre2> <Apellido1>PRIMAPELLIDODEMARCOS</Apellido1> <Apellido2>SEGAPPELLIDODEMARCOS</Apellido2> <ApellidoAdoptivo1 /> <ApellidoAdoptivo2 /> <Sexo>1</Sexo> <FechaNacimiento>1972-08-15</FechaNacimiento> <CodNacionalidad>1</CodNacionalidad> <NombreEnCedula>juan garcia</NombreEnCedula> </ObjPersona> </ObtPersonaPorDocResult> </ObtPersonaPorDocResponse> </env:Body> </pre> Mensaje original	<pre> <env:Body> <ObtPersonaPorDocResponse xmlns="http://ws.DNIC/"> <ObtPersonaPorDocResult> <ObjPersona> <CodTipoDocumento/> <NroDocumento/> <Nombre1>MARCOS</Nombre1> <Nombre2>SEBASTIAN</Nombre2> <Apellido1>PRIMAPELLIDODEMARCOS</Apellido1> <Apellido2>SEGAPPELLIDODEMARCOS</Apellido2> <ApellidoAdoptivo1/> <ApellidoAdoptivo2/> <Sexo/> <FechaNacimiento/> <CodNacionalidad/> <NombreEnCedula>juan garcia</NombreEnCedula> </ObjPersona> </ObtPersonaPorDocResult> </ObtPersonaPorDocResponse> </env:Body> </pre> Mensaje filtrado
---	---

Figura 33 - Filtro de mensaje 1

Como se puede observar en la Figura 33 los datos CodTipoDocumento, NroDocumento y Sexo, fueron filtrados del mensaje ya que son de tipo LIMITED y no se cuenta con los consentimientos requeridos. CodNacionalidad fue filtrado por ser un elemento de tipo DENIED.

Seguidamente, ingresaremos al sitio de administración del Banco de Previsión Social (BPS) y verificaremos qué consentimientos del ciudadano con documento C.I: 4728398 son necesarios para realizar la finalidad “T201” la cual implica invocar al servicio utilizado en este ejemplo.

En la Figura 34 se puede ver que los consentimientos faltantes son sobre los datos personales configurados como LIMITED como era previsto.



Nuevos Consentimientos

Buscar ciudadano

Tipo Documento:

CI

Documento:

4728398

Finalidad:

Trámite de prueba

Buscar

Consentimientos Faltantes

Dato Personal	Organismo origen	Organismo destino	Fecha inicio	Fecha fin	
documento	Dirección Nacional de Identificación Civil	Banco de Previsión Social	dd/mm/yyyy	dd/mm/yyyy	<button>Agregar</button>
documento	Banco de Previsión Social	Dirección Nacional de Identificación Civil	dd/mm/yyyy	dd/mm/yyyy	<button>Agregar</button>
sexo	Dirección Nacional de Identificación Civil	Banco de Previsión Social	dd/mm/yyyy	dd/mm/yyyy	<button>Agregar</button>
fecha de nacimiento	Dirección Nacional de Identificación Civil	Banco de Previsión Social	dd/mm/yyyy	dd/mm/yyyy	<button>Agregar</button>

Andrés Echevarría - 4728398

Dato Personal	Organismo Origen	Organismo Destino	Fecha inicio	Fecha fin
---------------	------------------	-------------------	--------------	-----------

Figura 34 - Ejemplo cálculo de consentimientos

Asumiendo que el ciudadano otorgó su consentimiento explícito según establece la Ley N° 18.331, se procede a agregar consentimientos para los datos personales documento y sexo. Luego se vuelve a invocar el servicio y éste fue el resultado obtenido:

Mensaje original	Mensaje filtrado
<pre> <env:Body> <ObtPersonaPorDocResponse xmlns="http://ws.DNIC/"> <ObtPersonaPorDocResult> <ObjPersona> <CodTipoDocumento>DO</CodTipoDocumento> <NroDocumento>37513028</NroDocumento> <Nombre1>MARCOS</Nombre1> <Nombre2>SEBASTIAN</Nombre2> <Apellido1>PRIMAPELLIDODEMARCOS</Apellido1> <Apellido2>SEGAPPELLIDODEMARCOS</Apellido2> <ApellidoAdoptivo1 /> <ApellidoAdoptivo2 /> <Sexo>1</Sexo> <FechaNacimiento>1972-08-15</FechaNacimiento> <CodNacionalidad>1</CodNacionalidad> <NombreEnCedula>juan garcia</NombreEnCedula> </ObjPersona> </ObtPersonaPorDocResult> </ObtPersonaPorDocResponse> </env:Body> </pre>	<pre> <env:Body> <ObtPersonaPorDocResponse xmlns="http://ws.DNIC/"> <ObtPersonaPorDocResult> <ObjPersona> <CodTipoDocumento>DO</CodTipoDocumento> <NroDocumento>37513028</NroDocumento> <Nombre1>MARCOS</Nombre1> <Nombre2>SEBASTIAN</Nombre2> <Apellido1>PRIMAPELLIDODEMARCOS</Apellido1> <Apellido2>SEGAPPELLIDODEMARCOS</Apellido2> <ApellidoAdoptivo1 /> <ApellidoAdoptivo2 /> <Sexo>1</Sexo> <FechaNacimiento/> <CodNacionalidad> <NombreEnCedula>juan garcia</NombreEnCedula> </ObjPersona> </ObtPersonaPorDocResult> </ObtPersonaPorDocResponse> </env:Body> </pre>

Figura 35 - Filtro de mensaje 2

La respuesta obtenida esta vez posee, además de los datos de la invocación anterior, los datos personales documento y sexo debido a que se crearon los consentimientos.

A través del portal *web*, los ciudadanos pueden acceder a la lista de consentimientos que han otorgado. En la Figura 36 se muestra la lista de consentimientos existentes para el ciudadano del ejemplo. Como se puede apreciar el ciudadano puede agregar y eliminar consentimientos.

Consentimientos

Andres Echevarria - 4728398

Dato Personal	Organismo	Finalidad	Válido	
documento	Dirección Nacional de Identificación Civil	Trámite de prueba	+	Eliminar
documento	Banco de Previsión Social	Trámite de prueba	+	Eliminar
sexo	Dirección Nacional de Identificación Civil	Trámite de prueba	+	Eliminar

Crear nuevo consentimiento

Dato personal:

Organismo:

Finalidad:

Desde: Hasta:

Figura 36 - Web de Usuarios

Como se ha mostrado en este caso de estudio el prototipo cumple con los objetivos establecidos sobre el control de mensajes.

6.3 Solicitud de acceso a datos

En este caso de estudio se muestra el ciclo de vida de una solicitud de acceso a datos. Como ya se comentó los ciudadanos pueden generar solicitudes a los organismos para obtener todos los datos que el organismo tenga sobre ellos. La Figura 37 muestra la página de creación de una nueva solicitud.

Mis Solicitudes

Solicitudes

Fecha de creación	Organismo	Estado
24-02-2015	Ministerio de Desarrollo Social	PENDIENTE
24-02-2015	Dirección Nacional de Aduanas	PENDIENTE
24-02-2015	Banco de Previsión Social	FINALIZADA

Nueva solicitud

Usted puede solicitar la obtención de toda la información que sobre sí mismo se halle en bases de datos públicas o privadas de un determinado organismo según establece la Ley 18.331 Artículo 14.

Seleccione el organismo sobre el cual desea realizar su solicitud:

Figura 37 - Solicitudes de acceso a datos de un ciudadano

En la misma encontramos una lista con todas las solicitudes creadas por el usuario y el estado de cada una.

Por otra parte en la *web* de administración de cada organismo se puede acceder al listado de solicitudes desde donde es posible ver el estado de cada una y finalizarlas. Cada vez que un administrador finaliza una solicitud de acceso a datos, el sistema notifica al ciudadano vía email que puede ir a retirar sus datos en el organismo correspondiente. En la Figura 38 se muestran las solicitudes creadas en la Dirección General de Aduanas.

Solicitudes de acceso a datos						
Solicitudes Pendientes						
Id	Fecha de creación	Fecha límite	Documento	Nombre	Estado	
4	24-02-2015	01-03-2015	CI 4482709	Morales, Dahiana	PENDING	<button>Finalizar</button>
Historial						
Id	Fecha de creación	Fecha de finalización	Documento	Nombre		
2	24-02-2015	24-02-2015	CI 4728398	Echevarria, Andres		

Figura 38 - Solicitudes creadas a Dirección General de Aduanas

6.4 Dashboard de administración y monitoreo

En este caso de estudio analizaremos el *dashboard* de administración, el mismo está compuesto por una serie de gráficas y tablas las cuales muestran:

Cantidad de mensajes por hora

Esta gráfica muestra la cantidad de mensajes recibidos en las últimas 24 horas.

Permite monitorear la fluctuación de mensajes, en particular es posible detectar horas pico en las que aumenta el tráfico entre organismos. En la Figura 39 se puede observar que la cantidad máxima de mensajes registrados por el sistema ocurrió a las 9hs con 113 mensajes.

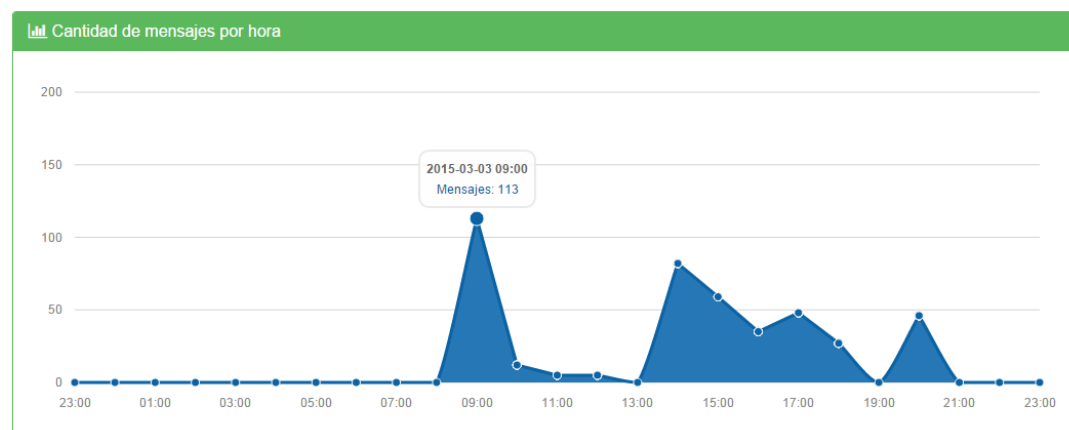


Figura 39 - Gráfica 1

Resultados

Esta gráfica muestra el porcentaje de mensajes que han sido: Autorizados o No Autorizados.

En la Figura 40 se puede observar que el 48% fueron Autorizados, el 51% No autorizados y el último 1% fueron estructuralmente inválidos.

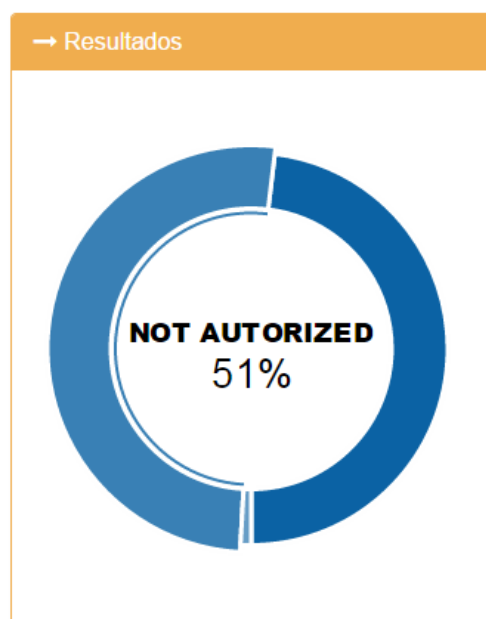


Figura 40 - Gráfica 2

Últimos mensajes

Esta gráfica muestra el desglose de los últimos diez mensajes recibidos, mostrando fecha, organismo origen, organismo destino y el resultado de la solicitud de la decisión de autorización. En la Figura 41 se muestra un ejemplo.

→ Últimos mensajes				
Fecha	Org origen	Org Destino	Metodo	Resultado
03-03-2015 17:14	MSP	DNIC	ObtPersonaPorDoc	AUTHORIZED
03-03-2015 17:13	MSP	DNIC	ObtPersonaPorDoc	AUTHORIZED
03-03-2015 17:13	MSP	DNIC	ObtPersonaPorDoc	AUTHORIZED
03-03-2015 17:09	MSP	DNIC	ObtPersonaPorDoc	AUTHORIZED
03-03-2015 22:08	MSP	DNIC	ObtPersonaPorDoc	NOT AUTHORIZED
03-03-2015 22:08	MSP	DNIC	ObtPersonaPorDoc	NOT AUTHORIZED
03-03-2015 17:08	MSP	DNIC	ObtPersonaPorDoc	AUTHORIZED
03-03-2015 17:07	MSP	DNIC	ObtPersonaPorDoc	NOT AUTHORIZED
03-03-2015 17:00	MSP	DNIC	ObtPersonaPorDoc	NOT AUTHORIZED
03-03-2015 17:14	MSP	DNIC	ObtPersonaPorDoc	AUTHORIZED

Figura 41- Gráfica 3

Mensajes de los últimos siete días

En esta gráfica se muestra la cantidad de mensajes por día que han sido autorizados y no autorizados en Clouseau durante la última semana.

En la Figura 42 se puede observar por ejemplo que el 26 de febrero del año 2015 se registraron 183 mensajes autorizados y 225 no autorizados.

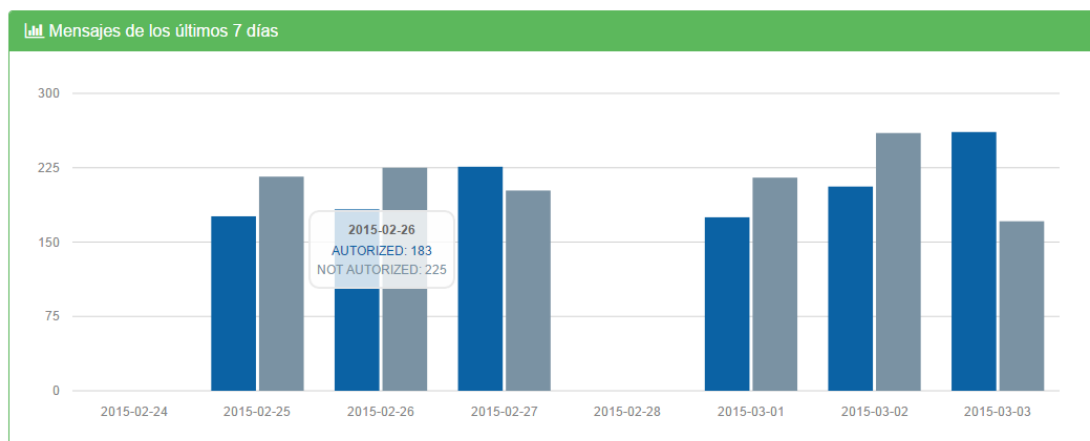


Figura 42 – Gráfica 4

7 Conclusiones y Trabajo a Futuro

En este capítulo se presentan las conclusiones del proyecto y las posibles mejoras que podrían ser realizadas como trabajo a futuro sobre la solución planteada.

7.1 Conclusiones

El objetivo general del proyecto consistió en proponer una solución que permita gestionar, monitorear y asegurar el cumplimiento de normativas asociadas a la protección de datos personales en el contexto de las plataformas de integración. En particular la solución fue planteada en el contexto de la Plataforma de Gobierno Electrónico y la ley de Protección de Datos Personales del Uruguay. Se identificaron como aportes del proyecto el análisis de la problemática planteada y las necesidades sobre la protección de datos personales. Además se realizó un estudio sobre el funcionamiento básico de la PGE.

Para cumplir con los objetivos fue necesario realizar un análisis de la Ley Nº 18.331 identificando los conceptos y requerimientos definidos en la misma. Como resultado de dicho análisis se obtuvieron una serie de requisitos funcionales que abarcaron los objetivos propuestos. Asimismo se determinó que el estándar más adecuado a incorporar en la solución teniendo en cuenta la realidad planteada, fuese XACML, ya que brinda mecanismos para controlar el acceso a recursos.

Se diseñó una solución basada en ESB la cual resuelve el problema de asegurar el cumplimiento de las normas establecidas y permite realizar un monitoreo del sistema.

Se llevó a cabo un prototipo el cual permitió validar la solución propuesta, su viabilidad y las herramientas utilizadas. En cuanto a las herramientas pudimos concluir que el estándar XACML es muy útil y flexible ya que se adecuó en gran medida a las necesidades planteadas. Por otra parte, creemos que el *framework* SwitchYard tiene una gran potencialidad y resuelve nativamente varios de los problemas planeados pero a nuestro criterio aún no es una herramienta muy madura. Al ser una herramienta relativamente nueva no existe mucho material disponible en la *web* y consideramos que la documentación presenta limitaciones. La falta de material sumado a que no contamos con soporte por parte del equipo de SwitchYard hizo que por momentos el desarrollo lleve más tiempo del estimado y fuera dificultoso.

Una vez finalizado el prototipo se realizaron una serie de pruebas y se verificó el correcto funcionamiento del mismo. A partir del análisis y las pruebas realizadas se identificaron una serie de posibles mejoras que se detallan en la siguiente sección.

7.2 Trabajo a Futuro

A continuación se mencionan algunas limitaciones encontradas en el prototipo realizado y posibles mejoras identificadas. Cabe destacar que las limitaciones encontradas no impiden que el prototipo sirva como validación de la solución propuesta.

- **Correspondencia *request-response***

Sería interesante contar con algún mecanismo que permita, al momento de evaluar un mensaje que es un *response* de un método, saber a qué *request* corresponde. Esto podría realizarse utilizando el atributo *RelatesTo* del cabezal *WS-Addressing*. Este atributo permite especificar en los mensajes de tipo *response* el ID del mensaje *request* con el cual está relacionado.

- **Identificar atributos requeridos de un *web service***

Actualmente, cuando se analiza un mensaje de tipo *request* en el sistema, no se identifica cuáles de sus elementos son requeridos por el *web service* invocado. En caso de intentar enviar datos requeridos sin el consentimiento del titular se intentará modificar el mensaje para transformarlo en un mensaje válido, por ejemplo, quitando los datos no permitidos del mensaje y se continuará con la invocación al *web service*.

Una mejora en la *performance* del sistema sería que al momento de procesar un mensaje *request*, si se identifica que no se tiene consentimientos para compartir un dato que es requerido por el *web service* invocado, se retorne un mensaje de error y no se realice la invocación al servicio. Esto es posible ya que al ser un dato requerido, la invocación del *web service* fallará inevitablemente.

- **Actions XACML**

Otra posible mejora sería contar con más acciones en los mensajes XACML. En el prototipo desarrollado solo se utiliza la acción “*Send*” sobre el recurso. Podrían agregarse acciones referentes por ejemplo al uso de los datos, o posibilidad de compartirlos, modificarlos entre otros y establecer consentimientos para cada tipo de acción.

- **Seguridad**

En cuanto a temas de seguridad, una posible mejora sería incorporar el protocolo *Hypertext Transfer Protocol Secure* [46](HTTPS) en el componente *web*.

- **Mejora en solicitud de acceso a datos**

Respecto a la funcionalidad que permite crear solicitudes de acceso a datos, una mejora sería que los organismos puedan agregar los datos requeridos por el ciudadano a la solicitud. Esto permitiría que el usuario pueda obtener los datos solicitados vía *web* sin necesidad de dirigirse personalmente al organismo. De esta manera el ciclo entero de la solicitud podría realizarse *online*.

- **Agregar funcionalidades en *backoffice* de administración**

Con el objetivo de mejorar y hacer más completo el *backoffice* de administración, identificamos como trabajo a futuro la implementación de bajas y modificaciones de entidades. Por ejemplo la eliminación de un organismo o de un *web service*. También se podrían agregar nuevas métricas al *dashboard*.

- **Cálculo de consentimientos a nivel de ciudadanos**

Si bien la solución contempla el cálculo de consentimientos a nivel de los administradores de cada organismo, sería interesante contar con la misma funcionalidad a nivel de ciudadanos. Es decir que los ciudadanos puedan, desde el portal *web*, indicar el trámite que desean realizar en un determinado organismo y a partir de dicha selección se calculen automáticamente los consentimientos faltantes para su realización. Dicho cálculo se realiza teniendo en cuenta los consentimientos necesarios para la realización del trámite y los ya brindados por el ciudadano. Luego de realizado el cálculo el ciudadano tiene la posibilidad de brindar los consentimientos necesarios y se registrarán en el sistema.

Cabe destacar que ésta funcionalidad podría agregarse fácilmente ya que utilizaría el mismo algoritmo que el cálculo de consentimientos ya implementado para los administradores. Para llevarla a cabo solo sería necesario desarrollar la interfaz *web* a nivel de ciudadanos.

- **Analizar XACML Privacy Policy Profile**

Investigar y analizar la viabilidad de la utilización de perfil *Privacy Policy Profile* introducido en la versión 3.0 de XACML para el manejo del concepto de finalidad.

- **Integración con motor de procesamiento de eventos complejos**

Otro aspecto para desarrollar a futuro y una posible línea de extensión de este proyecto podría ser la integración del mismo con un motor de procesamiento de eventos complejos (CEP) [47]. Esta integración podría permitir la generación de alertas cuando se detecten determinados eventos en tiempo real. Por ejemplo, podrían lanzarse alertas cuando se detecte un número determinado de mensajes bloqueados para un mismo organismo.

También podría usarse CEP para lanzar alertas que permitan cumplir con requerimientos temporales. En particular, la ley de protección de datos de Uruguay establece que todo ciudadano tiene derecho a obtener la información que sobre sí mismo se halle en cada organismo dentro de los cinco días hábiles de haber sido solicitada. La configuración de alarmas podría prevenir el vencimiento del plazo estipulado para la finalización de las solicitudes de acceso a datos del sistema.

La utilización de CEP en conjunto con el *dashboard* ya implementado formarían una herramienta muy poderosa para el monitoreo general del sistema, detección de incumplimientos por parte de los organismos y generación de métricas de negocio.

8 Referencias

- [1] "Integración de Aplicaciones encapsuladas para el desarrollo de Sistemas de Información Cooperativos", [En línea]. Available: <http://www.fing.edu.uy/~jabin/papers/TesisMaestria.pdf>. [Último acceso: 08 03 2014].
- [2] "Propuestas de Proyectos de Grado 2014", [En línea]. Available: <http://www.fing.edu.uy/inco/grupos/lins/pgs-2014.pdf>. [Último acceso: 08 03 2015].
- [3] "Parlamento", [En línea]. Available: <http://www.parlamento.gub.uy/leyes/AccesoTextoLey.asp?Ley=18331&Anchor=>. [Último acceso: 14 02 2015].
- [4] "AGESIC", [En línea]. Available: <http://www.agesic.gub.uy/>. [Último acceso: 14 02 2015].
- [5] "SOA", [En línea]. Available: <http://www.oracle.com/technetwork/articles/javase/soa-142870.html>. [Último acceso: 18 02 2015].
- [6] "Web Services Description Requirements", [En línea]. Available: <http://www.w3.org/TR/ws-desc-reqs/#definitions>. [Último acceso: 14 02 2015].
- [7] "SOAP Specifications", [En línea]. Available: <http://www.w3.org/TR/soap/>. [Último acceso: 16 02 2015].
- [8] "WSDL Specifications", [En línea]. Available: <http://www.w3.org/TR/wsdl>. [Último acceso: 14 02 2015].
- [9] M. Papazoglou, Web Services: Principles and Technology, Prentice Hall, 2007.
- [10] "WSDL", [En línea]. Available: http://help.sap.com/saphelp_erp60_sp/helpdata/en/0d/44c277ad3644da90006c786a568c14/content.htm. [Último acceso: 05 03 2015].
- [11] "UDDI Version 3.0.2", [En línea]. Available: <http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm>. [Último acceso: 16 Febrero 2015].
- [12] "Web Services Addressing Working Group", [En línea]. Available: <http://www.w3.org/2002/ws/addr/>. [Último acceso: 14 02 2015].
- [13] AGESIC, "Descripción y Guías de uso de la Plataforma de Gobierno Electrónico del Estado Uruguayo", [En línea]. Available: http://www.agesic.gub.uy/innovaportal/file/1295/1/descripcion_y_guias_de_uso_de_la_plataforma_de_gobierno_electronico_del_estado_uruguayo.pdf. [Último acceso: 18 02 2015].
- [14] D. Chappell, Enterprise Service Bus. O'Reilly., 2004.
- [15] "XSL Transformations (XSLT)", [En línea]. Available: <http://www.w3.org/TR/xslt>. [Último acceso: 14 02 2015].

- [16] "BizTalk ESB Toolkit 2.1", [En línea]. Available: <https://msdn.microsoft.com/en-us/biztalk/dd876606.aspx>. [Último acceso: 14 02 2015].
- [17] L. Gonzales, "Plataforma ESB Adaptativa para Sistemas", 2011.
- [18] D. C. S. L. W. D. a. O. O. J. Balasubramanian, "Evaluating the Performance of Middleware Load Balancing Strategies," in EDOC, pp. 135-146, 2004.
- [19] D. Chappell, Enterprise Service Bus: Theory in Practice. O'Reilly Media, 2004.
- [20] R. Kissel, Glossary of key information security terms, NIST Interag. Rep. NIST IR, vol. 7298, p. 3, 2013.
- [21] P. C. Ramarao Kanneganti, SOA Security. Manning., 2008.
- [22] R. Shirey, "IETF - RFC 2828 - Internet Security Glossary, IETF - RFC 2828 - Internet Security Glossary", [En línea]. Available: <http://www.ietf.org/rfc/rfc2828.txt>. [Último acceso: 14 Febrero 2015].
- [23] D. Gollmann, Computer Security, 2nd ed. Wiley, 2006.
- [24] U. D. o. Defense, "Trusted Computer System Evaluation Criteria ["Orange Book"]", [En línea]. Available: <http://csrc.nist.gov/publications/history/dod85.pdf>. [Último acceso: 14 Febrero 2015].
- [25] "WS-Security 1.1", [En línea]. Available: <https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>. [Último acceso: 14 02 2015].
- [26] "WS-Trust 1.3", [En línea]. Available: <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>. [Último acceso: 14 02 2015].
- [27] "WS-POLICY Specification", [En línea]. Available: <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html>. [Último acceso: 16 02 2015].
- [28] "XACML-Specification", [En línea]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml. [Último acceso: 16 02 2015].
- [29] S. RUBINO- HALLMAN, E-government in Latin America and The Caribbean. Reinventing governance in the Information Age. P.12..
- [30] "datospersonales", [En línea]. Available: <http://www.datospersonales.gub.uy/e-learning/>. [Último acceso: 22 02 2015].
- [31] "AGESIC - CertificadosService", [En línea]. Available: http://www.agesic.gub.uy/innovaportal/v/3392/1/agesic/dinapyme_consulta_de_certificados.html. [Último acceso: 30 03 2015].
- [32] "Catálogo de Servicios - AGESIC", [En línea]. Available: http://www.agesic.gub.uy/innovaportal/v/1602/1/agesic/catalogo_de_servicios.html?menuderecho=5. [Último acceso: 14 03 2015].
- [33] AGESIC, "Modelo persona AGESIC", [En línea]. Available: http://www.agesic.gub.uy/innovaportal/v/2533/1/agesic/metadatos_modelo_de_referencia_persona.html. [Último acceso: 01 03 2015].

- [34] "Spring", [En línea]. Available: <http://projects.spring.io/spring-framework/>. [Último acceso: 23 02 2015].
- [35] "Hibernate", [En línea]. Available: <http://hibernate.org/>. [Último acceso: 23 02 2015].
- [36] "MySQL", [En línea]. Available: <http://www.mysql.com/>. [Último acceso: 23 02 2015].
- [37] "JSF", [En línea]. Available: <http://www.java-serverfaces.org/>. [Último acceso: 23 02 2015].
- [38] "Bootstrap", [En línea]. Available: <http://getbootstrap.com/>. [Último acceso: 23 02 2015].
- [39] "Maven", [En línea]. Available: <http://maven.apache.org/>. [Último acceso: 23 02 2015].
- [40] "APACHE CAMEL", [En línea]. Available: <http://camel.apache.org/>. [Último acceso: 23 02 2015].
- [41] J. Anstey, Camel in Action.
- [42] "XPath specification", [En línea]. Available: <http://www.w3.org/TR/xpath/>. [Último acceso: 23 02 2015].
- [43] "Morris.js", [En línea]. Available: <http://morrisjs.github.io/morris.js/>. [Último acceso: 23 02 2015].
- [44] "JBoss XACML", [En línea]. Available: <http://docs.jboss.org/jbosssecurity/docs/jbossexacml/pdf/jbossexacml.pdf>. [Último acceso: 02 04 2015].
- [45] "Routing Slip Pattern", [En línea]. Available: <http://camel.apache.org/routing-slip.html>. [Último acceso: 22 02 2015].
- [46] "RFC", [En línea]. Available: <http://tools.ietf.org/html/rfc2818>. [Último acceso: 21 02 2015].
- [47] "Complex Events", [En línea]. Available: <http://www.complexevents.com/2013/09/17/understanding-real-time-intelligence/#8>. [Último acceso: 15 03 2015].
- [48] "Servicio Básico de Información - DNIC", [En línea]. Available: http://agesic.gub.uy/innovaportal/v/1799/1/agesic/servicio_basico_de_informacion.html. [Último acceso: 15 03 2015].
- [49] M. P. a. W. Heuvel, "Service oriented architectures: approaches".
- [50] "MDC", [En línea]. Available: <http://www.eaipatterns.com/CanonicalDataModel.html>. [Último acceso: 23 03 2015].
- [51] "SwitchYard", [En línea]. Available: <http://switchyard.jboss.org/>. [Último acceso: 24 03 2015].

Apéndice 1. Mensajes XACML

A continuación en la Figura 43 y la Figura 44 se encuentran ejemplos de *request* y *response* XACML generados por Clouseau.

```
<Response>
<Result ResourceID="http://www.onlinestore.com/sensitive/paymentinfo.html">

    <Decision>Deny</Decision>

    <Status>
        <StatusCode value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>

    <Obligations>

    <Obligation ObligationId="LogSuccessfulRead" FulfillOn="Permit">
    <AttributeAssignment  AttributeId="actions"  DataType="http://clouseauSchema#ListActions">
        <ListActions>
            <Action>
                <type>Email</type>
            </Action>
        </ListActions>
    </AttributeAssignment>

    <AttributeAssignment  AttributeId="params"  DataType="http://clouseauSchema#ListParams">
        <ListParams>
            <param>Documento</param>
            <param>Sexo</param>
        </ListParams>
    </AttributeAssignment>

    <AttributeAssignment  AttributeId="transformation"  DataType="http://www.w3.org/2001/XMLSchema#string">
        TransformerFilter
    </AttributeAssignment>
    </Obligation>

    </Obligations>
</Result>
</Response>
```

Figura 43 - Ejemplo de response XACML de Clouseau

```

< Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 http://docs.oasis-
open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
ReturnPolicyIdList="false" >

  <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute IncludeInResult="false" AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        DGI
      </AttributeValue>
    </Attribute>
  </Attributes>

  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <Attribute IncludeInResult="false"
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
        mensaje
      </AttributeValue>
    </Attribute>
  </Attributes>

  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute IncludeInResult="false" AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Send
      </AttributeValue>
    </Attribute>
  </Attributes>

  <Attribute Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">

    <Attribute IncludeInResult="false" AttributeId="purpose">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">T200</AttributeValue>
    </Attribute>

    <Attribute IncludeInResult="false" AttributeId="receiverOrganization">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">BPS</AttributeValue>
    </Attribute>

    <Attribute IncludeInResult="false" AttributeId="documentNumber">
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">4728398</AttributeValue>
    </Attribute>

    <Attribute IncludeInResult="false" AttributeId="documentType">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CI</AttributeValue>
    </Attribute>

    <Attribute IncludeInResult="false" AttributeId="method">
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">wsServicioDeInformacionBasico</Attribu
        teValue>
    </Attribute>

    <Attribute IncludeInResult="false" AttributeId="requestType">
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">RESPONSE</AttributeValue>
    </Attribute>
  </Attributes>
</Request>

```

Figura 44 – Ejemplo de request XACML de Clouseau

Apéndice 2. Transformaciones XSLT

En el prototipo se incluyen dos archivos XSL a modo de ejemplo. En la Figura 45 y Figura 46 se muestra el contenido de los mismos.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="utf-8" indent="no" />
  <xsl:param name="params" />
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
  <xsl:template match="//*[contains($params,local-name())]">
    <xsl:copy>
      <xsl:apply-templates select="@*|*|comment()|processing-instruction()" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Figura 45- TransformerFilter.xsl

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="utf-8" indent="no" />
  <xsl:template match="@*|node()">
    <Error>El contenido de este mensaje ha sido bloqueado.</Error>
  </xsl:template>
</xsl:stylesheet>
```

Figura 46 – TransformerBlocker.xsl

El archivo “TransformerFilter.xsl” recibe una lista de *Strings* como parámetros y retorna una copia del documento XML recibido, eliminando los valores en las etiquetas cuyos nombres se encuentren en la lista recibida. Esto permite filtrar del mensaje datos no autorizados.

En el archivo “TransformerBlocker.xsl” se retorna un nuevo documento XML cuyo contenido es “<Error>El contenido de este mensaje ha sido bloqueado.</Error>”

Apéndice 3. Servicio Básico de Información

En este apéndice se detalla el *web service* utilizado en el primer caso de estudio. Como ya se mencionó se trata del Servicio Básico de Información provisto por DNIC.

El servicio consta de dos métodos. El primer método, devuelve los datos filiatorios asociados a un número de cédula. El segundo, permite conocer la versión del WS que se está utilizando.

“La DNIC brinda en estos servicios la información tal cual está disponible en su base de datos. Esto significa que los errores que pudieran haber están reflejados también en estos servicios. Es de notar que la base de datos de la DNIC fue cargada en forma masiva al comienzo de la informatización, y que si bien los datos son depurados todo el tiempo pueden existir registros que no se encuentren en regla (por ej. personas que desde entonces no han hecho trámites en la DNIC y por lo tanto para los cuales no se pudo comprobar la veracidad de la información disponible).” [48]

A continuación se presenta el wsdl del *web service*:

```

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://wsDNIC/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://wsDNIC/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Servicio Web de Información
  Básica</wsdl:documentation>
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://wsDNIC/">
      <s:element name="ObtPersonaPorDoc">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="paramObtPersonaPorDoc"
type="tns:objParamObtPersonaPorDoc" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="objParamObtPersonaPorDoc">
        <s:complexContent mixed="false">
          <s:extension base="tns:objParamAutenticacion">
            <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="Nrodocumento" type="s:string" />
              <s:element minOccurs="0" maxOccurs="1" name="TipoDocumento" type="s:string" />
            </s:sequence>
          </s:extension>
        </s:complexContent>
      </s:complexType>
      <s:complexType name="objParamAutenticacion">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Organizacion" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="PasswordEntidad" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="ObtPersonaPorDocResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="ObtPersonaPorDocResult"
type="tns:obtPersonaPorDocResult" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="obtPersonaPorDocResult">
        <s:complexContent mixed="false">
          <s:extension base="tns:objResult">
            <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="ObjPersona" type="tns:objPersona" />
            </s:sequence>
          </s:extension>
        </s:complexContent>
      </s:complexType>
      <s:complexType name="objResult">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Errores" type="tns:ArrayOfMensaje" />
          <s:element minOccurs="0" maxOccurs="1" name="Warnings" type="tns:ArrayOfMensaje" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="ArrayOfMensaje">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="Mensaje" nillable="true"
type="tns:Mensaje" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="Mensaje">

```

```

<s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="CodMensaje" type="s:int" />
  <s:element minOccurs="0" maxOccurs="1" name="Descripcion" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="DatoExtra" type="s:string" />
</s:sequence>
</s:complexType>
<s:complexType name="objPersona">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="CodTipoDocumento" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="NroDocumento" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Nombre1" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Nombre2" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Apellido1" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Apellido2" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="ApellidoAdoptivo1" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="ApellidoAdoptivo2" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="Sexo" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="FechaNacimiento" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="CodNacionalidad" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="NombreEnCedula" type="s:string" />
  </s:sequence>
</s:complexType>
<s:element name="ProductDesc">
  <s:complexType />
</s:element>
<s:element name="ProductDescResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="ProductDescResult" type="tns:obtProductInfo" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="obtProductInfo">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="modalidad" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="version" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="descripcion" type="s:string" />
  </s:sequence>
</s:complexType>
</s:schema>
</wsdl:types>
<wsdl:message name="ObtPersonaPorDocSoapIn">
  <wsdl:part name="parameters" element="tns:ObtPersonaPorDoc" />
</wsdl:message>
<wsdl:message name="ObtPersonaPorDocSoapOut">
  <wsdl:part name="parameters" element="tns:ObtPersonaPorDocResponse" />
</wsdl:message>
<wsdl:message name="ProductDescSoapIn">
  <wsdl:part name="parameters" element="tns:ProductDesc" />
</wsdl:message>
<wsdl:message name="ProductDescSoapOut">
  <wsdl:part name="parameters" element="tns:ProductDescResponse" />
</wsdl:message>
<wsdl:portType name="wsServicioDeInformacionBasicoSoap">
  <wsdl:operation name="ObtPersonaPorDoc">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Método de obtención de datos de
una persona por número de cédula.</wsdl:documentation>
    <wsdl:input message="tns:ObtPersonaPorDocSoapIn" />
    <wsdl:output message="tns:ObtPersonaPorDocSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="ProductDesc">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Método para identificación del
presente servicio web, versión y modalidad de ejecución</wsdl:documentation>
  </wsdl:operation>

```

```

    <wsdl:input message="tns:ProductDescSoapIn" />
    <wsdl:output message="tns:ProductDescSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="wsServicioDeInformacionBasicoSoap"
type="tns:wsServicioDeInformacionBasicoSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="ObtPersonaPorDoc">
    <soap:operation soapAction="http://wsDNIC/ObtPersonaPorDoc" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="ProductDesc">
    <soap:operation soapAction="http://wsDNIC/ProductDesc" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="wsServicioDeInformacionBasico">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Servicio Web de Información
  Básica</wsdl:documentation>
  <wsdl:port name="wsServicioDeInformacionBasicoSoap"
binding="tns:wsServicioDeInformacionBasicoSoap">
    <soap:address location="https://servicios.pge.red.uy:10018//wsServicioDeInformacionBasico" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Apéndice 4. Plataforma de Gobierno Electrónico

En este apéndice se describe la Plataforma de Middleware y el Sistema de Seguridad de la PGE.

1 Plataforma de Middleware

La plataforma de middleware provee mecanismos para facilitar el desarrollo, despliegue e integración de servicios y aplicaciones [13].

A continuación, se muestra un ejemplo de funcionamiento de la misma extraído de "AGESIC, Descripción y Guías de uso de la Plataforma de Gobierno Electrónico del Estado Uruguayo" [13], y luego se describirán sus principales componentes.

Ejemplo de Uso de la Plataforma de Middleware

El ejemplo consta de un servicio "Cédula de Identidad" (CI), que permite, a partir de un número de cédula, obtener información pública de una persona. El servicio es provisto por los organismos A y B.

La invocación de los servicios de la PGE se realiza mediante el envío de mensajes. Por lo que, para consumir el servicio de CI, los funcionarios y ciudadanos deben enviar, a través de la aplicación en uso, un mensaje a la PGE con los datos de su solicitud.

Una vez que el mensaje llega a la PGE, y pasa los controles de seguridad necesarios, es reenviado a la Plataforma de Middleware que se encarga de realizar las siguientes acciones:

1. Se realizan validaciones de integridad como la verificación de nulos, estructuras de datos incompletos o errores en tipos de datos. En caso de encontrar errores, el mensaje es rechazado y se notifica al cliente los motivos.
2. Se realizan validaciones para determinar si el mensaje satisface las políticas de seguridad definidas por la PGE. En casos de encontrar errores, el mensaje es rechazado y se notifica al cliente los motivos.
3. Se elige el mejor destino del mensaje.
4. Una vez seleccionado el destino del mensaje en caso de ser necesario se aplican transformaciones al mensaje.
5. Finalmente se envía el mensaje al servicio destino.

La Figura 47 presenta de forma gráfica el ejemplo descrito anteriormente.

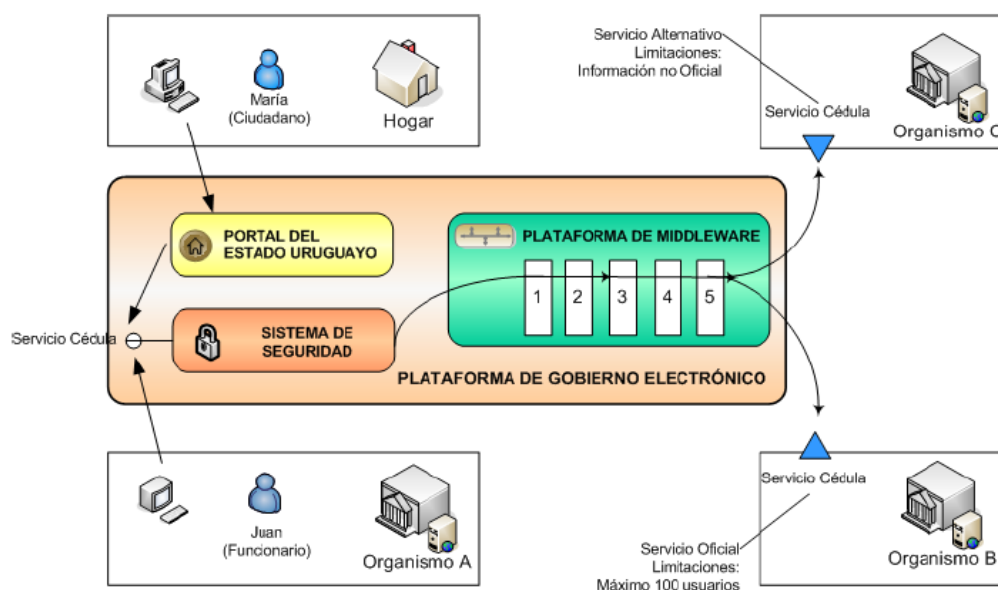


Figura 47 - Ejemplo de Funcionamiento de la Plataforma de Middleware [13]

Componentes de la Plataforma de Middleware

La Figura 48 presenta una visión general de la Plataforma de Middleware.



Figura 48 - Plataforma de Middleware [13]

Entornos de Ejecución

Si bien en general las aplicaciones y servicios de la PGE se alojan en los propios organismos, la Plataforma de Middleware provee entornos de ejecución para alojar aplicaciones y servicios en la propia PGE. Estos entornos se basan en tecnologías de Middleware tales como Servidores de Aplicaciones, entre otros [13].

Registro de Servicios

El Registro de Servicios de la PGE provee funcionalidades para que los organismos publiquen, describan, busquen y descubran servicios en la PGE. Además de los nombres, proveedores y categorías de los servicios, es posible acceder a la descripción de los mismos, especificada en WSDL que brinda los datos necesarios para que una aplicación cliente pueda invocarlos [13].

Productos Enterprise Service Bus

Los productos de ESB de la Plataforma de Middleware proveen mecanismos que pueden ser utilizados por los organismos para el consumo y provisión de servicios.

Algunos de los principales mecanismos que brindan estos productos en el marco de la PGE son: transparencia de la ubicación, Mecanismos de Mensajería Confiable, Transformación y Enriquecimiento de Mensajes, Ruteo Basado en Contenido y Monitoreo [13].

2 Sistema de Seguridad

El Sistema de Seguridad provee servicios de seguridad al resto de los componentes. Brinda mecanismos que permiten realizar el control de acceso a los servicios de la plataforma, de acuerdo a políticas definidas, para que el acceso a los mismos sea seguro. También se encarga del auditado de los eventos relacionados con la seguridad [13].

Como se puede observar en la Figura 49, el Sistema de Seguridad se divide en tres grandes componentes: Sistema de Auditoría, Sistema de Control de Acceso y Servicio Periféricos de Seguridad.

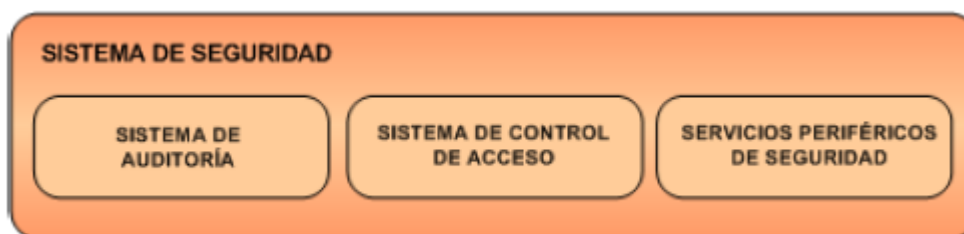


Figura 49 - Sistema de Seguridad [13]

- Sistema de Auditoría: provee las herramientas para recolectar información y realizar análisis y reportes de auditoría.
- Servicios Periféricos de Seguridad: Los integran una Autoridad Certificadora (CA) que emite y gestiona los certificados que se utilicen en la operativa de la PGE y un servicio de directorio que refleja la estructura organizacional y de roles de los organismos. Este directorio posee una estructura gestionada por la PGE, pero también replica automáticamente las estructuras de directorios de los organismos, brindando una visión unificada.
- Sistema de Control de Acceso: brinda los mecanismos para aplicar políticas de control de acceso sobre los servicios publicados y aplicaciones disponibles. El control de acceso se realiza siguiendo un esquema RBAC, utilizando el rol del usuario que quiere acceder al servicio o aplicación y las políticas de acceso definidas en la PGE [13].

Apéndice 5. Glosario

DAO	Data Access Objects.
ESB	Enterprise Service Bus.
MDC	Modelo Canónico de Datos.
OASIS	Organization for the Advancement of Structured Information Standards, organización encargada de orientar la adopción de estándares de comercio electrónico y servicios web.
PAP	Policy Administration Point, entidad del sistema que crea las políticas o conjuntos de políticas de acceso.
PDP	Policy Decision Point, entidad del sistema que evalúa las políticas aplicables y emite una decisión de autorización.
PEP	Policy Enforcement Point, entidad del sistema que ejecuta el control de acceso, realizando solicitudes de decisión de autorización y haciendo cumplir las decisiones tomadas.
PIP	Policy Information Point, entidad del sistema que actúa como fuente de atributos.
SAML	Security Assertion Markup Language
W3C	World Wide Web Consortium.
WSDL	Web Services Description Language, es una especificación de un lenguaje basado en XML para describir de forma estándar un <i>web service</i>
XACML	eXtensible Access Control Markup Language, es un estándar de OASIS basado en XML que define un lenguaje de políticas de acceso y un lenguaje <i>request/response</i> para la toma de decisiones de control de acceso.
XML	eXtensible Markup Language.
XSLT	Extensible Stylesheet Language Transformations, lenguaje especificado por W3C para realizar transformaciones sobre documentos XML.