TESINA DE LICENCIATURA

Programa de apoyo para alumnos con experiencia profesional

TÍTULO: Desarrollo de un Middleware para la Interacción con un Sistema Legacy
AUTORES: Cristian Godoy
DIRECTOR ACADÉMICO: Andrés Rodriguez
DIRECTOR PROFESIONAL: Javier Petruccelli
CARRERA: Licenciatura en Sistemas

Resumen

Con el paso del tiempo los sistemas de software se vuelven cada vez más obsoletos. A estos sistemas comúnmente se los denomina Sistemas Legacy. Una estrategia para modernizar estos sistemas es el wrapping funcional, a través del cual se envuelve la funcionalidad legacy con una capa de software mucho más moderna.
Por otro lado, la tecnología de web services se ha consolidado como un estándar a la hora de integrar diferentes aplicaciones de software a través de Internet.
En este trabajo se analizan estrategias para modernizar sistemas legacy y se presenta un caso de estudio en el cual se adapta un sistema legacy utilizando wrapping funcional y servicios web.

Palabras Clave

Legacy Systems – Wrapping Funcional – Web Services – Middleware – SOAP

Conclusiones

Combinar wrapping funcional y web services puede resultar una estrategia muy útil para integrar un sistema legacy con otras aplicaciones web, y evitar así el costo de un reemplazo o reimplementación del sistema completo.

Trabajos Realizados

- *Se estudiaron los sistemas legacy y las principales técnicas que pueden emplease para modernizar estos sistemas.*
- *Se describieron conceptos sobre web services junto con los estándares y protocolos que más se utilizan en su implementación.*
- *Se presentó un caso real de modernización de un sistema legacy y la solución adoptada aplicando una estrategia de wrapping funcional y servicios web.*

Trabajos Futuros

- *Evaluar otras soluciones para invocar web services SOAP desde otros motores de base de datos diferentes a SQL Server.*
- *Estudiar soluciones que permitan adaptar o migrar servicios basados en SOAP hacia otras tecnologías más recientes como REST.*

Fecha de la presentación: Diciembre 2022

Desarrollo de un Middleware para la Interacción con un Sistema Legacy

Cristian Godoy

Tabla de contenidos

Capítulo 1. Introducción

1.1 Objetivo

Los Sistemas Legacy son sistemas de software que han quedado antiguos dentro de una organización. Se trata de sistemas que han sido desarrollados hace varios años utilizando tecnologías y herramientas que se consideran obsoletas en la actualidad (Sneed, 2000). Estos sistemas constituyen un activo muy valioso para cualquier organización, dado que en ellos se implementan las funciones que dan soporte a los procesos de negocio y por lo general no pueden ser reemplazados o actualizados de manera sencilla.

En este contexto, y dado que los sistemas Legacy necesitan continuar operando dentro de las compañías, se hace necesario contar con una serie de estrategias que ayuden a gestionar el cambio y la evolución en este tipo de sistemas.

Por otro lado, el surgimiento de la tecnología de Web Services (*Web Services Architecture*, 2004) hace algunos años ha permitido a las compañías vincular sus propios sistemas con otros sistemas de terceros a través de Internet. De esta manera, una aplicación puede exponer ciertas funcionalidades como servicios para que otras aplicaciones los consuman a través de la Web, dando lugar así al desarrollo de aplicaciones más integradas y heterogéneas.

El objetivo de este trabajo es realizar un análisis de las soluciones disponibles para la modernización de un sistema Legacy, haciendo hincapié en un caso de estudio en el cual se lleva a cabo la transformación de un sistema legacy para que pueda interoperar con otros sistemas a través de Web Services.

1.2 Motivación

La Dirección General de Sistemas Informáticos de Administración Financiera (DGSIAF) es el área perteneciente a la Subsecretaría de Presupuesto de la Secretaría de Hacienda de la Nación. La función de este organismo es promover el desarrollo y mantenimiento de los sistemas que dan soporte a la gestión presupuestaria, financiera y contable del sector público nacional.

Entre los sistemas desarrollados por la DGSIAF se encuentra el Sistema de Unidades Ejecutoras de Préstamos Externos (UEPEX) que se utiliza para administrar los fondos provistos por financiamiento externo para programas o proyectos específicos del país. Se trata de un sistema que fue desarrollado hace varios años utilizando tecnologías y herramientas que actualmente han quedado en desuso.

Frente a esta situación y ante la necesidad del sistema UEPEX de poder interactuar y compartir información con otros sistemas de la administración pública nacional, se hace necesario llevar a cabo un proceso de transformación de este sistema sin necesidad de abordar un proceso de reingeniería completo.

1.3 Organización del Trabajo

Este trabajo se encuentra organizado de la siguiente manera:

En el **Capítulo 2** se introduce el concepto de sistemas legacy y se describen las distintas actividades del ciclo de evolución de un sistema de software. El capítulo se centra en la actividad de modernización de sistemas legacy y las principales técnicas para implementar esta actividad.

En el **Capítulo 3** se define el concepto de web services, sus características principales y las tecnologías más utilizadas para su implementación.

En el **Capítulo 4** se presenta un caso real de modernización de un sistema legacy en el que se aplica la técnica de wrapping funcional y la tecnología de web services. El capítulo describe la implementación realizada y los resultados de la solución adoptada.

Finalmente, en el **Capítulo 5** se presentan las conclusiones del trabajo y se plantean algunas líneas de investigación para trabajos futuros.

Capítulo 2. Sistemas Legacy

En este capítulo se introduce el concepto de sistema legacy, sus principales características y los problemas asociados a esta clase de sistemas. Luego se describe el ciclo de evolución de un sistema de software y las actividades que se llevan a cabo a lo largo del mismo: mantenimiento, modernización y reemplazo. El capítulo se centra en la actividad de modernización para la cual se detallan las principales técnicas empleadas a la hora de modernizar sistemas legacy.

2.1 Conceptos sobre Sistemas Legacy

En la literatura se pueden encontrar varias definiciones sobre Sistemas Legacy:

1. “Legacy software components can be jobs, transactions, programs, modules or procedures within existing application systems which are more than five years old... Although the technology with which they have been implemented is out of fashion, the application systems themselves are performing critical business functions in an acceptable and reliable manner.” (Sneed, 2000).
2. (Yang & Hongji, 2001) proponen la siguiente definición: “Legacy software is referred to software systems which were developed some time ago, which have been undergone considerable modification and which are still indispensably used nowadays. Two main features usually characterizes a legacy software system: (1) huge volume of software code, and (2) deteriorated software documentation.”
3. En (Wu et al., 1997) se define a un Sistema Legacy como: “The widespread use of computer technology over several decades has resulted in some large, complex systems which have evolved to a state where they significantly resist further modification and evolution. Such systems are termed Legacy Systems”

4. (Visaggio, 2001) aporta la siguiente definición: “A legacy system is generally one of an organization’s assets with a high economic value. Even when it ages, it is both difficult and risky to replace it. Difficult, because the system is used by many people within the organization and its replacement would involve retraining all the users to understand the new system. Risky, because the construction or purchase of a new system may go over budget and disrupt planned schedules; moreover, the new system may lack some functions the users of the previous system were used to having.”
5. En (Fahmideh et al., 2017) se caracteriza a estos sistemas como: “Those that are mission critical, expensive to maintain, brittle and inflexible to changes, run on obsolete hardware, incomplete or outdated documentation, and difficult to extend and integrate with other systems.”

A partir de estas definiciones se pueden inferir las características más importantes vinculadas a un Sistema Legacy:

- Software implementado hace varios años
- Software de importancia para las compañías
- Software construido con tecnologías obsoletas
- Software con poca o nula documentación
- Software que no puede ser reemplazado de manera sencilla

Aunque lo más razonable sería poder reemplazar a estos sistemas antiguos, la realidad es que esto no siempre es posible por algunas limitaciones tales como:

- Grandes inversiones acumuladas en el sistema legacy a lo largo del tiempo.
- Recursos de TI no disponibles para la implementación de un nuevo sistema.
- Falta de interés por parte de los stakeholders para reemplazar el viejo y confiable sistema legacy.

Las organizaciones deben adoptar estrategias para poder gestionar adecuadamente el software legacy. Una mala administración o el descuido de estos sistemas puede ocasionar pérdidas financieras y afectar negativamente la imagen corporativa. Algunos ejemplos conocidos sobre las consecuencias de no gestionar adecuadamente el software legacy, son:

- **Banco de Escocia:** En junio de 2012 los sistemas del Royal Bank of Scotland dejaron de funcionar, impidiendo que miles de usuarios pudieran acceder a sus cuentas y causándole al banco pérdidas millonarias en concepto de daños y perjuicios. Los analistas concluyeron que el fallo fue provocado por la compleja y obsoleta infraestructura de los sistemas informáticos. El CEO Ross McEwan declaró que el banco había ignorado su tecnología durante décadas (Nesbit, 2020).
- **Gobierno Americano:** El avance de la pandemia de COVID-19 hizo que se incrementaran la cantidad de solicitudes de desempleo en los Estados Unidos. Como consecuencia de esto, una gran cantidad de personas debieron esperar durante varias semanas para que los sistemas informáticos pudieran procesar sus solicitudes. Estos retrasos fueron atribuidos a problemas en el software anticuado e incompatible utilizado en varios de los estados federales. La mayoría de estos sistemas datan de la década del 80 y algunos incluso son anteriores a esa época (Charette, 2020).
- **Agencia de Informes Crediticios Equifax:** En septiembre de 2017 la compañía Equifax, una de las agencias de informes de crédito al consumo más grandes del mundo, anunció que fue víctima de un ciberataque por el cual se expusieron datos confidenciales de 146 millones de personas. La complejidad de sus sistemas legacy contribuyó a que no se reparara una vulnerabilidad crítica en el Sistema Automatizado de Entrevistas al Consumidor, un portal personalizado de

la compañía que fue desarrollado durante la década del 70 (*The Equifax Data Breach*, 2018).

- **Delta Air Lines:** En agosto de 2016 una falla técnica en uno de los data centers de la compañía Delta Air Lines obligó a suspender unos 2300 vuelos, provocando caos en varios aeropuertos y afectando negativamente la imagen de la empresa. El problema se originó cuando los sistemas de backup no se activaron correctamente luego de un corte de energía. Una vez superado este incidente, el sistema de back-end demoró varias horas para poder conectarse correctamente con los sistemas de facturación y embarque. El CFO de la compañía Paul Jacobson sostuvo que además de una aerolínea eran una empresa de tecnología y que por lo tanto necesitaban mejorar sus sistemas (Patel, 2017).

2.2 Problemas Característicos de los Sistemas Legacy

Los sistemas Legacy son una herramienta fundamental para dar soporte a los procesos de negocio dentro de una organización. En general, estos sistemas están en producción desde hace varios años y los usuarios ya se encuentran familiarizados con el uso de esta tecnología. Sin embargo, este clase de sistemas suelen estar asociados con las siguientes problemáticas:

- **Altos costos de mantenimiento:** El mantenimiento de estos sistemas suele ser costoso en cuanto a tiempo y esfuerzo debido a la falta de documentación y a la ausencia de personal idóneo en tecnologías legacy.
- **Problemas de integración:** Algunos sistemas legacy no cuentan con interfaces bien definidas para poder integrarse correctamente con otros sistemas más modernos.

- **Riesgos de seguridad:** Estos sistemas fueron desarrollados hace varios años utilizando herramientas y tecnologías que se encuentran deprecadas o que ya no reciben soporte por parte de la comunidad, por lo tanto resultan más vulnerables desde el punto de vista de la seguridad.

A pesar de estos inconvenientes los sistemas legacy continúan siendo utilizados por numerosas organizaciones alrededor del mundo. Estos sistemas, en general, implementan mucha lógica de negocios que representa el esfuerzo de varios años de programación, mantenimiento y pruebas. Por lo tanto, las empresas no pueden simplemente descartarlos por ser considerados problemáticos.

2.3 Evolución de los Sistemas Legacy

La evolución de un sistema es un concepto amplio que abarca desde la corrección de un bug hasta la reimplementación completa de todo el sistema. Las actividades propias de la evolución se pueden dividir en tres categorías: *mantenimiento*, *modernización* y *reemplazo*. La Figura 2.1 ilustra cómo se aplican las diferentes actividades de la evolución en las distintas fases del ciclo de vida del software. La línea punteada representa las crecientes necesidades comerciales, mientras que la línea continua representa la funcionalidad proporcionada por el sistema. El mantenimiento continuo va dando soporte a las necesidades del negocio, pero a medida que el sistema se vuelve cada vez más obsoleto, el mantenimiento deja atrás las necesidades del negocio. Se requiere entonces una actividad de modernización que represente un mayor esfuerzo, tanto en tiempo como en funcionalidad, que la actividad propia de mantenimiento. Finalmente, cuando el sistema anterior ya no puede evolucionar, debe ser reemplazado (Comella-Dorda et al., 2000). En las siguientes secciones se describen brevemente cada una de estas actividades.

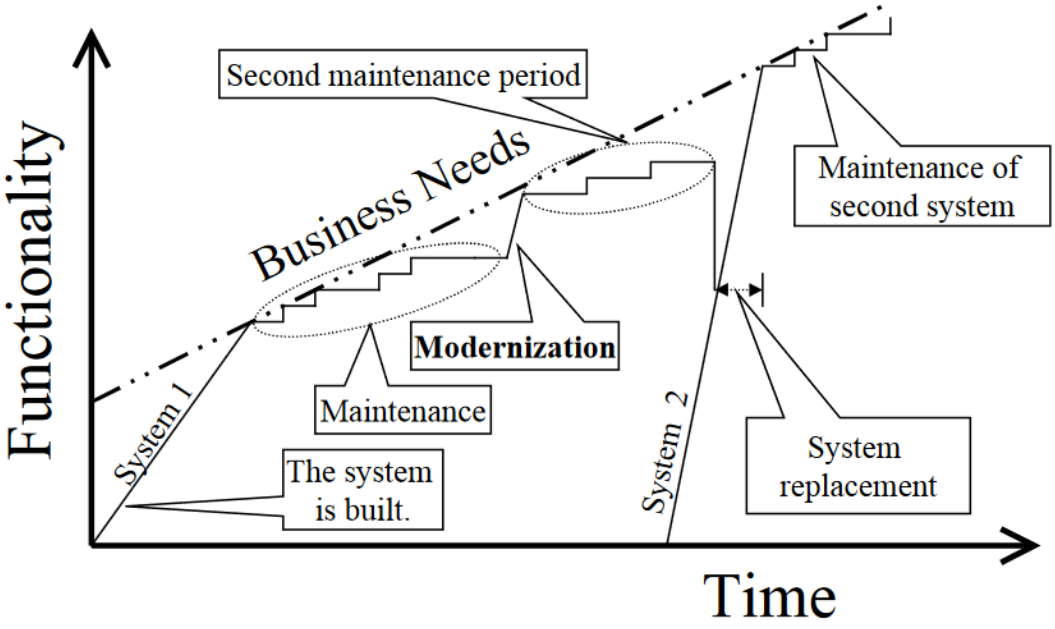


Figura 2.1. Ciclo de Vida de un Sistema de Software (Comella-Dorda et al., 2000)

2.3.1 Mantenimiento

El mantenimiento es un proceso continuo e incremental en el que se van introduciendo pequeñas modificaciones en el sistema. Estas modificaciones pueden ser el resultado de la corrección de bugs o el agregado de nuevas funcionalidades, siempre que esto no implique grandes cambios en la arquitectura del sistema. El mantenimiento es una actividad necesaria para que el sistema pueda evolucionar correctamente. Sin embargo, esta actividad tiene algunas limitaciones:

1. La adopción de nuevas tecnologías se encuentra limitada durante la etapa de mantenimiento. Por ejemplo, algunas mejoras como la implementación de una arquitectura distribuida o una interfaz GUI no se consideran parte del mantenimiento.

2. Los costos de mantenimiento se incrementan con el paso del tiempo debido a que el sistema se vuelve cada vez más obsoleto y resulta más dificultoso contratar expertos en tecnologías legacy.
3. La modificación de un sistema para adaptarlo a los nuevos requerimientos del negocio se vuelve cada vez más difícil porque los pequeños cambios tienen un mayor impacto en los sistemas legacy.

2.3.2 Reemplazo

El reemplazo consiste en desarrollar o comprar un nuevo sistema que sustituya al viejo y obsoleto sistema legacy. El reemplazo es apropiado para aquellos sistemas que no pueden adaptarse a las necesidades del negocio y para los cuales la modernización no resulta posible o no es rentable. Esta técnica posee algunos riesgos que deben ser evaluados antes de su implementación:

1. El reemplazo implica la construcción de un nuevo sistema desde cero, por lo cual se necesita una alta asignación de los recursos de TI al nuevo proyecto. Estos recursos pueden estar abocados a otras tareas y por lo tanto no estar familiarizados con las tecnologías necesarias para el desarrollo del nuevo sistema.
2. Los sistemas legacy han sido testados a lo largo del tiempo y encapsulan la experiencia de varios años de trabajo. El reemplazo, por su parte, requiere un testeo completo de todos los módulos y unidades implementadas, y aun así no hay garantías de que el nuevo sistema sea tan robusto o funcional como el anterior.

2.3.4 Modernización

La modernización es la alternativa con más impacto en los sistemas legacy. Esta actividad implica cambios más extensos que en el mantenimiento, pero conservando una porción significativa del sistema existente. Estos cambios incluyen la reestructuración del sistema, mejoras funcionales o nuevos atributos de software. A continuación se describen las principales técnicas empleadas para la modernización de sistemas (Jha et al., 2015):

- **Screen Scraping:** Esta técnica se basa en la modernización de la interfaz de usuario (UI) de un sistema legacy para mejorar la visibilidad y usabilidad del mismo. Estos sistemas, por lo general, han sido implementados utilizando interfaces de solo texto que se ejecutan sobre una terminal de usuario. En contraste, una nueva interfaz puede implementarse de forma gráfica o con lenguaje HTML de manera de poder accederse a través de un navegador web.

  La técnica Screen Scraping puede aplicarse fácilmente dotando al viejo sistema con una nueva UI. La interacción entre la nueva interfaz y la interfaz legacy se lleva a cabo a través de un middleware específico (screen scraping tool). De esta manera, las acciones de los usuarios sobre la nueva interfaz se envían al middleware específico, el cual se encarga de mapear cada acción a la interfaz legacy correspondiente.
- **Wrapping Funcional:** A través de esta técnica se “envuelve” a un sistema legacy con una nueva capa de software, la cual permite ocultar la complejidad del viejo sistema y proveer una interfaz mucho más moderna. Los componentes envueltos con la nueva interfaz actúan como servidores, ejecutando las funciones requeridas por un cliente externo, el cual no necesita conocer en detalle la implementación legacy.

El wrapping permite conservar componentes legacy y al mismo tiempo integrarlos con otros componentes de software para poder cumplir con los nuevos requerimientos de la organización.

- **Wrapping de Datos:** El wrapping de datos permite el acceso a datos legacy utilizando interfaces o protocolos distintos a los del diseño original. Esta técnica mejora la conectividad y a la vez permite la integración de los datos legacy con infraestructuras mucho más modernas. (Comella-Dorda et al., 2000) proponen tres tipos de wrapping de datos:
    1. **Gateway de Base de Datos:** Utiliza un software específico encargado de llevar a cabo la traducción entre el protocolo legacy y algún protocolo de acceso a datos estándar (ODBC, JDBC, etc). Este tipo de wrapping resulta muy útil ya que en el desarrollo de nuevas aplicaciones la tendencia es la utilización de protocolos estándares.
    2. **Integración con XML:** Se emplea lenguaje XML (eXtensible Markup Language) para poder realizar la integración de datos entre sistemas. El lenguaje XML es un estándar de facto para el intercambio de datos a través de internet. Este tipo de wrapping utiliza un servidor XML que actúa como mediador entre la infraestructura legacy e Internet. Existen soluciones comerciales para facilitar la implementación de un servidor XML. Estas soluciones soportan una gran variedad de protocolos de comunicación lo que permite la integración con una gran variedad de sistemas legacy.
    3. **Replicación de Base de Datos:** consiste en la descentralización del almacenamiento masivo replicando la información en instancias locales de bases de datos más modernas. De esta manera, cada instancia local posee una copia de los datos que son de su interés, los cuales se

sincronizan periódicamente con el repositorio central. Con este enfoque, las nuevas aplicaciones pueden acceder directamente a los datos en su instancia local, evitando los costos de tener que acceder remotamente a un repositorio obsoleto.

- **Migración:** La migración consiste en mover una aplicación legacy a una nueva plataforma con el objetivo de facilitar el mantenimiento de estas aplicaciones y permitir que se adapten mejor a los nuevos requerimientos del negocio, conservando las funciones y los datos del sistema original.

  En la actualidad esta técnica suele emplearse para mover aplicaciones legacy hacia otras plataformas más flexibles como Cloud o SOA (Service Oriented Architecture). La migración a servicios permite la reutilización de componentes de software ya establecidos y a su vez la integración de estos componentes con nuevos servicios, incluida la orquestación de los mismos para hacer frente a las necesidades cambiantes del negocio.

  Esta técnica contribuye a una mejora en la interoperabilidad de los sistemas legacy ya que en las plataformas más modernas la tendencia es hacia la comunicación y el intercambio de datos entre sistemas.

- **Integración CGI:** Esta técnica propone la integración de un sistema legacy utilizando CGI (Common Gateway Interface). CGI es un estándar que se utiliza para la comunicación entre un servidor web y una aplicación externa. La estrategia consiste en emplear esta tecnología para proveer el acceso a un sistema legacy a través de la web.

  En un escenario típico de integración CGI, un servidor web ejecuta un programa para acceder a los datos o a las funciones del sistema legacy y posteriormente envía al cliente el resultado de la ejecución a través de páginas HTML.

Al igual que la técnica de screen scraping, la integración CGI provee una nueva interfaz para el sistema legacy (en este caso siempre mediante páginas HTML) pero a diferencia del screen scraping la nueva GUI no interactúa con la vieja GUI sino que se comunica directamente con el core del sistema legacy.

- **Traducción de Código:** Esta técnica consiste en tomar un código fuente legacy y reescribirlo en un lenguaje de programación más moderno. A través de la traducción de código se busca facilitar el mantenimiento del código legacy y al mismo tiempo incorporar un nuevo lenguaje de programación que se adapte mejor a las necesidades futuras del sistema. La implementación de esta técnica suele ser costosa en cuanto a tiempo pero existen en el mercado herramientas de traducción automática que ayudan a reducir estos tiempos.

  La traducción de código no realiza cambios significativos en la estructura del programa. Esto significa que por ejemplo un sistema legacy escrito en COBOL luego de ser traducido a código Java se verá como una aplicación COBOL escrita en Java.

Capítulo 3. Web Services

En este capítulo se introduce el concepto de Web Services junto las características principales de esta tecnología. Luego se presentan los dos estándares más utilizados para la implementación de web services en la actualidad: SOAP y REST.

3.1 Conceptos y Definiciones sobre Web Services

Web services es un concepto conocido desde hace más de dos décadas en el ámbito de la industria del software. Desde su origen han ido surgiendo distintas definiciones para este concepto a lo largo del tiempo. Actualmente, un Web Service puede definirse como una API[1] Web que se ejecuta en el mismo sistema donde se aloja y que puede accederse a través de una red, utilizando un protocolo y un formato preestablecido. Esta tecnología se basa en la utilización de estándares y protocolos abiertos para permitir el intercambio de datos entre distintos sistemas de software.

Por medio de Web Services distintas aplicaciones de software, desarrolladas en lenguajes de programación diferentes y ejecutadas en cualquier plataforma puedan interoperar entre sí intercambiando datos a través de una red de computadoras.

Para comprender mejor el concepto de Web Services se plantea el siguiente escenario de ejemplo: supongamos que se quiere desarrollar una aplicación para publicar y consultar noticias sobre la actualidad del país. Además de presentar las noticias al usuario, esta aplicación deberá mostrar información actualizada sobre el estado del clima. Una solución podría ser que la misma aplicación implemente funcionalidades para registrar mediciones de

[1] Una API o Interfaz de Programación de Aplicaciones es un conjunto de definiciones y protocolos que se utilizan para diseñar e integrar componentes de software. Esta interfaz define cómo se accede a la funcionalidad provista por un componente de software sin exponer cómo está implementada dicha funcionalidad.

temperaturas y así poder brindar esta información al usuario. Sin embargo, esta solución no parece una buena idea ya que le agrega complejidad a la aplicación que además de implementar la gestión de las noticias deberá implementar también la gestión de los datos meteorológicos. Una solución más simple puede ser que esta nueva aplicación invoque directamente al web service que provee el Servicio Meteorológico Nacional para obtener información actualizada sobre el estado del clima. De esta manera, nuestra aplicación se integra con la aplicación meteorológica independientemente de las tecnologías que se utilizaron para la implementación de la misma. La interoperabilidad se consigue por medio de la utilización de web services.

La figura 3.1 muestra los pasos necesarios para poder llevar a cabo la interacción con un Web Service:

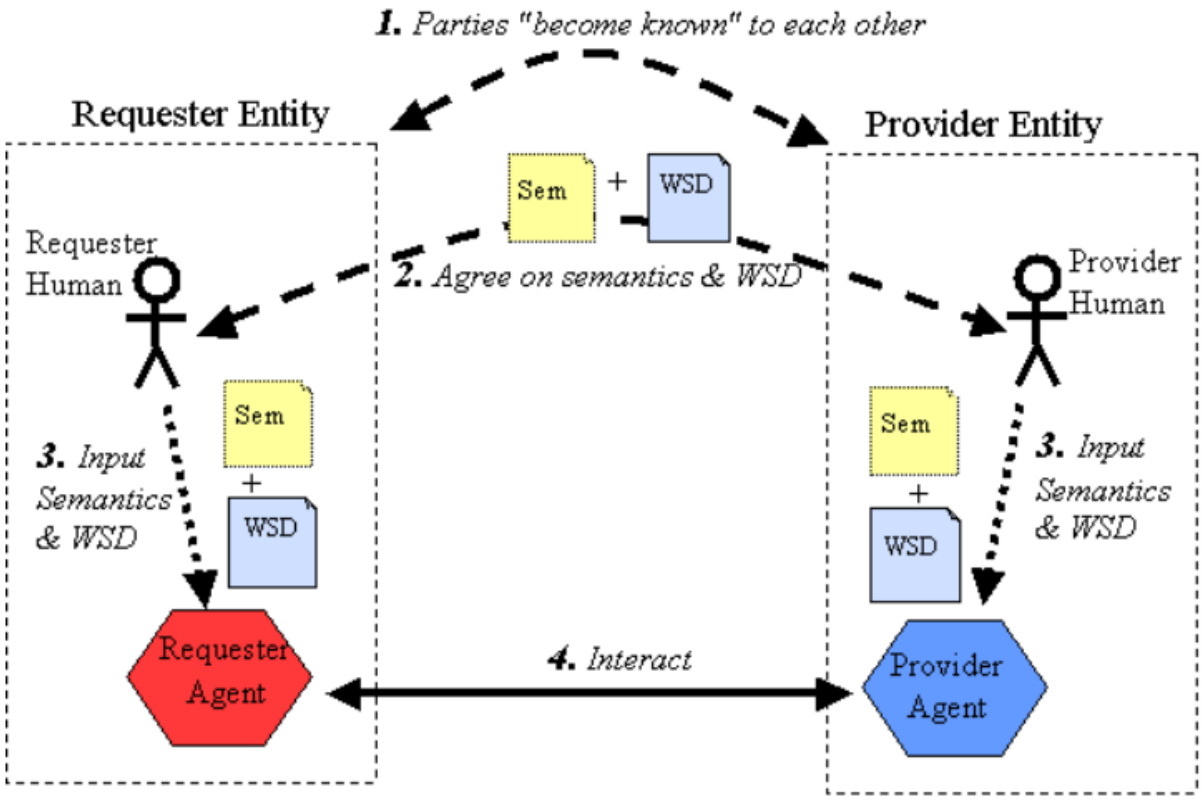


Figura 3.1. Pasos para la interacción con un Web Service (*Web Services Architecture*, 2004)

1. El solicitante y el proveedor deben conocerse entre sí o al menos haber registrado al proveedor en un catálogo de servicios para que pueda ser descubierto por el solicitante.
2. Solicitante y proveedor acuerdan la descripción del servicio (WSD) y la semántica que regirá la interacción entre las partes.
3. El solicitante y el proveedor utilizan el WSD y la semántica definida para implementar el servicio en los agentes correspondientes.
4. Los agentes intercambian mensajes para realizar alguna tarea en nombre del solicitante y el proveedor.

A continuación se describe el significado de los elementos principales de este diagrama:

- **Agente:** Es un componente de software implementado por el solicitante y el proveedor del servicio. Estos componentes interactúan entre sí enviando y recibiendo mensajes.
- **WSD y Semántica:** La descripción del servicio (WSD) contiene información sobre el formato de los mensajes, los tipos de datos, los protocolos, la ubicación del servicio y otros datos requeridos para poder llevar a cabo la interacción. La semántica se refiere al significado de los conceptos que maneja el servicio y su comportamiento.
- **Mensaje:** Representa la mínima unidad de intercambio entre agentes. Contiene dos partes: un encabezado con metadata (el tipo de operación, quien es el emisor, quien es el destinatario, etc) y un cuerpo con el contenido del mensaje.

3.2 Características de los Web Services

Los Web Services poseen una serie de características que los ha convertido en una de las tecnologías más utilizadas a la hora de comunicar e integrar diferentes aplicaciones de software. Entre estas características se incluyen:

- Utilizan estándares y protocolos abiertos como SOAP, XML y HTTP lo cual los hace independientes de la plataforma de ejecución.
- Pueden publicarse en Internet para ser accedidos por otras aplicaciones web.
- Su implementación no está ligada a un lenguaje de programación en particular. Una aplicación escrita en Java, por ejemplo, puede consumir un Web Service implementado en PHP.
- Ofrecen bajo acoplamiento entre el solicitante y el proveedor del servicio. Un desarrollador puede actualizar o modificar la lógica interna de un web service sin que se vean afectados los consumidores de ese servicio.
- Son utilizados para implementar aplicaciones distribuidas o sistemas basados en SOA[2].

3.3 Web Services SOAP

3.3.1 El Protocolo SOAP

SOAP (Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos, en diferentes procesos, pueden comunicarse entre sí intercambiando datos en

[2] SOA (Service Oriented Architecture) es un estilo de Arquitectura de Software que se basa en la definición de servicios reutilizables, con interfaces públicas bien definidas, donde los proveedores y consumidores de servicios interactúan en manera desacoplada para realizar los procesos de negocio.

lenguaje XML. Este estándar utiliza protocolos como HTTP o SMTP para el transporte de la información y lenguaje XML para la representación de la misma.

El modelo de comunicación de SOAP es muy similar al del protocolo HTTP. Un cliente realiza una petición (request) a un servidor, el cual procesa la petición y retorna una respuesta (response) con la información solicitada.

La figura 3.2 muestra los componentes principales de un web service basado en SOAP. Estos componentes son:

- **XML (eXtensible Markup Language):** Es un lenguaje de marcas basado en texto que se utiliza para representar información de forma estructurada. Es utilizado para describir servicios web y para codificar los mensajes que se intercambian entre el cliente y el servidor.
- **SOAP (Simple Object Access Protocol):** Protocolo de comunicación entre extremos que surge a partir de la necesidad de un formato de mensajes neutro, abierto y extensible. Los mensajes SOAP se representan en XML y se mapean en el protocolo de la capa de transporte (HTTP, SMTP, etc).
- **WSDL (Web Service Description Language):** Lenguaje común utilizado para describir servicios web. La descripción de los servicios se realiza de forma estándar mediante la utilización de documentos XML. Estos documentos agrupan toda la información relevante sobre el servicio: operaciones disponibles, estructura de los mensajes, parámetros de entrada y salida, tipos de datos, URL de acceso al servicio, etc.
- **UDDI (Universal Description, Discovery and Integration):** Es un estándar para registrar y descubrir servicios web. UDDI especifica cómo se publican y descubren los servicios, y cómo trabajan los repositorios donde se almacenan los descriptores de servicios web.

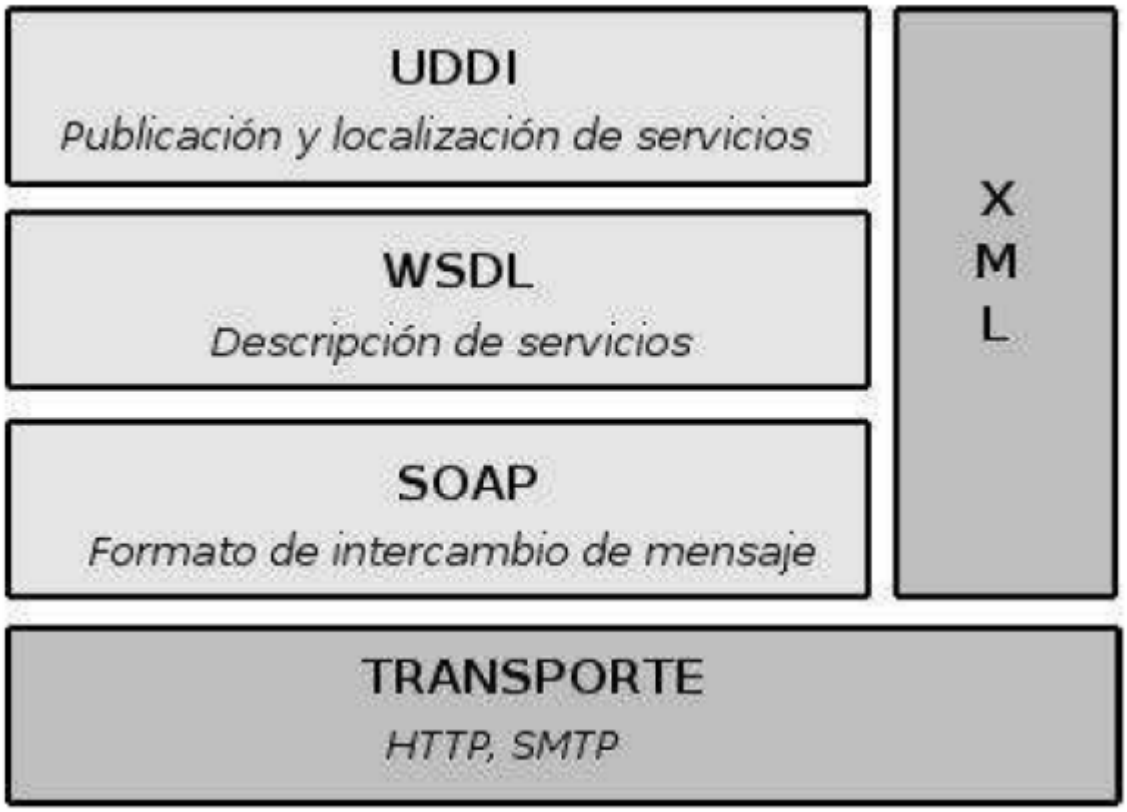


Figura 3.2. Componentes de un Web Service SOAP (Díaz Amado, 2014)

3.3.2 Estructura de Mensajes SOAP

Un mensaje SOAP se codifica como un documento XML. La figura 3.3 muestra la estructura de mensajes SOAP y sus elementos principales. Estos son:

- **Envelope (contenedor):** Es el elemento raíz del mensaje SOAP. Se utiliza para envolver la información a transmitir.
- **Head (cabecera):** Es un elemento opcional que contiene información específica de la aplicación (datos de seguridad, identificador de transacciones, etc).
- **Body (cuerpo):** Contiene la información a transportar. En este elemento se mapean las invocaciones y las respuestas en XML.

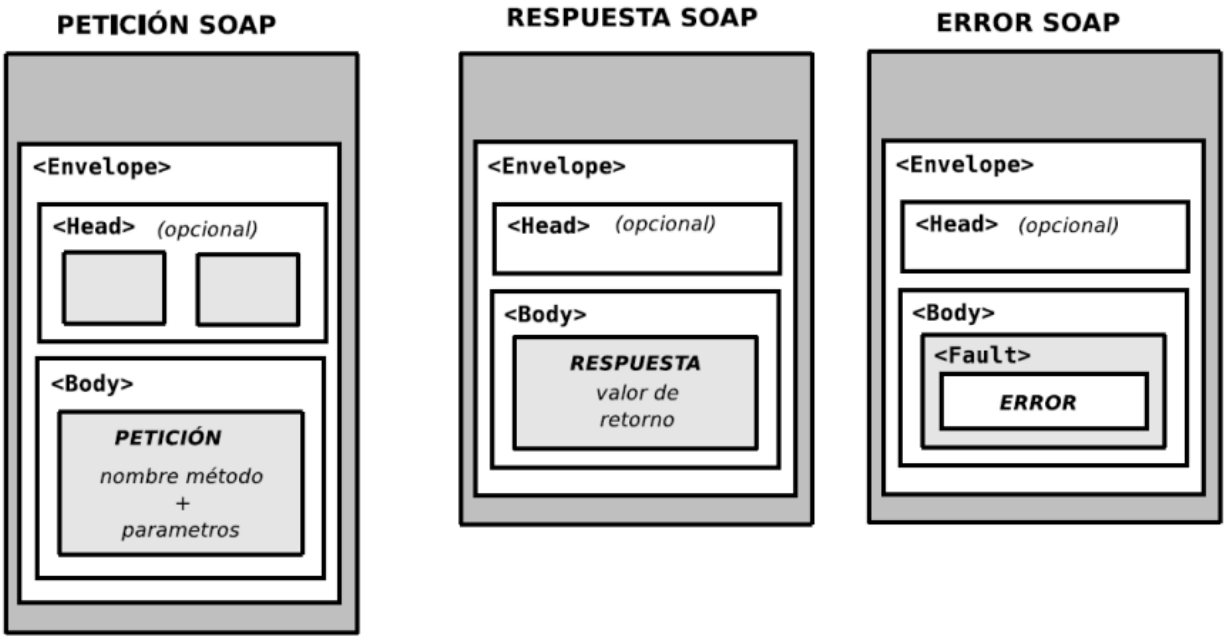


Figura 3.3. Estructura de Mensajes SOAP (Díaz Amado, 2014)

La especificación SOAP es independiente de los protocolos que se usan en la transmisión de los mensajes. SOAP solo define un contenedor de mensajes y cómo se encapsulan esos mensajes en el protocolo de la capa de transporte. Las implementaciones SOAP deben proveer los binding necesarios para el mapeo de los mensajes en los protocolos más comunes como HTTP o SMTP. La figura 3.4 muestra un escenario típico de intercambio de mensajes SOAP a través de HTTP.

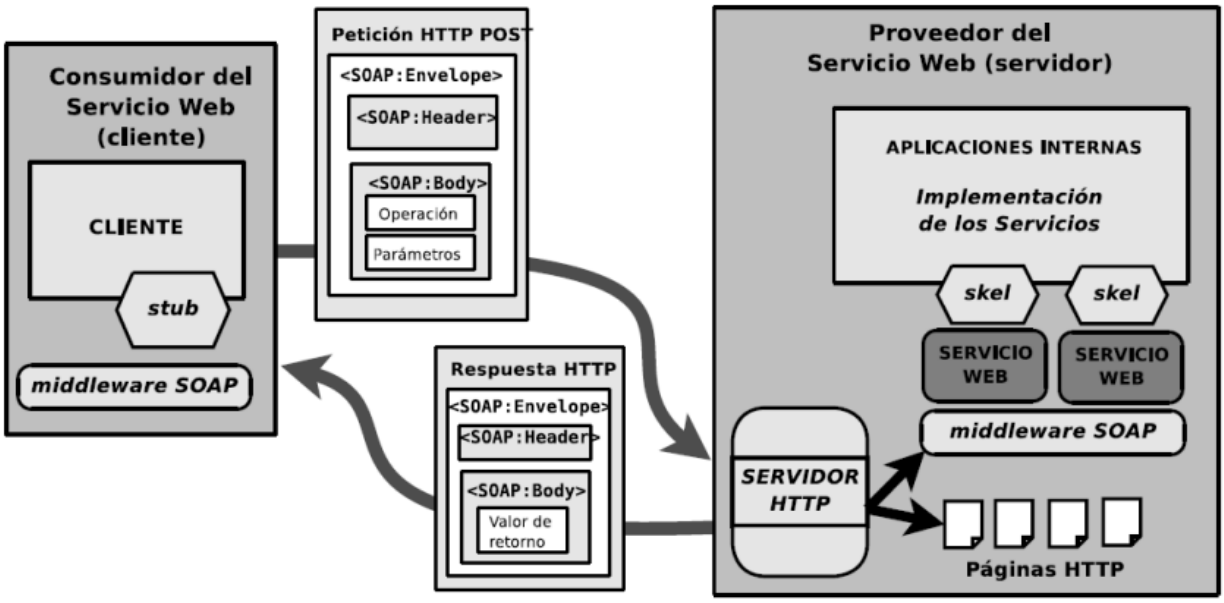


Figura 3.4. Intercambio de Mensajes SOAP a través del Protocolo HTTP (Díaz Amado, 2014)

En la figura 3.5 se presenta un ejemplo de mensaje SOAP en formato XML

```
<envelope>
    <header />
    <body>
        <empleados>
            <empleado>
                <nombre>Pablo Díaz</nombre>
                <edad>45</edad>
            </empleado>
            <empleado>
                <nombre>Carlos García</nombre>
                <edad>30</edad>
            </empleado>
            <empleado>
                <nombre>Florencia Perez</nombre>
                <edad>25</edad>
            </empleado>
        </empleados>
    </body>
</envelope>
```

Figura 3.5. Ejemplo de Mensaje SOAP

3.4 Web Services REST

3.4.1 La Especificación REST

REST (REpresentational State Transfer) es un estilo de arquitectura de software para sistemas distribuidos en la World Wide Web. REST plantea una serie de principios arquitectónicos para el desarrollo de aplicaciones en la web (entre las que se incluyen las APIs de Web Services).

En una implementación REST existen clientes, servidores y funcionalidades que se exponen a través de recursos identificados por URI[3]. Los clientes interactúan con los recursos

[3] Una URI (Uniform Resource Identifier) es una cadena de caracteres que identifica recursos de forma unívoca dentro de una red.

utilizando métodos que generan las solicitudes en los servidores, los cuales se encargan de procesar esas solicitudes y generar una respuesta adecuada.

REST no es un estándar en sí mismo, sólo plantea un conjunto de recomendaciones para un estilo de arquitectura específica. Aunque no se lo considere un estándar, esta especificación promueve la utilización de algunos estándares como HTTP, URI, XML, JSON, entre otros.

Las aplicaciones que siguen las restricciones del estilo REST se denominan aplicaciones RESTful. Se nombran a continuación las restricciones más relevantes:

- **Cliente-Servidor:** Utilización de un estilo arquitectónico cliente-servidor para separar responsabilidades. El servidor se encarga de controlar los datos mientras que el cliente se encarga de manejar las interacciones del usuario. Esta restricción mantiene al cliente y al servidor débilmente acoplados (el cliente no necesita conocer los detalles de implementación del servidor y el servidor se "despreocupa" de cómo son utilizados los datos que envía al cliente).
- **Sin estado:** Cada petición que recibe el servidor debería ser independiente y contener todo lo necesario para ser procesada.
- **Cacheable:** Debe admitir un sistema de almacenamiento en caché. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.
- **Interfaz uniforme:** Define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección o "URI".
- **Sistema de capas:** El servidor puede disponer de varias capas para su implementación. El uso de capas o servidores intermedios puede servir para

aumentar la escalabilidad (sistemas de balanceo de carga, cachés) o para implementar políticas de seguridad.

- **Código bajo demanda:** Permite que los servidores amplíen las funciones de un cliente transfiriendo código ejecutable.

REST propone la utilización de los métodos propios de HTTP para poder interactuar con los recursos. Estos métodos son:

- GET utilizado para recuperar un recurso.
- POST se utiliza la mayoría de las veces para crear un nuevo recurso. También puede utilizarse para el envío de datos a un recurso ya existente. En este segundo caso, no se crearía ningún recurso nuevo.
- PUT es útil para crear o editar un recurso. En el cuerpo de la petición irá la representación completa del recurso. En caso de existir, se reemplaza, de lo contrario se crea el nuevo recurso.
- PATCH realiza actualizaciones parciales. En el cuerpo de la petición se incluirán los cambios a realizar en el recurso. Puede ser más eficiente en el uso de la red que PUT ya que no envía el recurso completo.
- DELETE utilizado para eliminar un recurso.

La especificación REST simplifica la implementación de las APIs de Web Services ya que no se requieren descriptores de servicios ni capas de procesamiento adicional como en el caso del protocolo SOAP. Esto se traduce en implementaciones mucho más flexibles y ligeras.

3.4.2 Estructura de Mensajes REST

Los servicios REST permiten el intercambio de datos a través de diferentes formatos de mensajes, entre los que se incluyen el HTML, XML, texto sin formato o JSON. El formato más

utilizado es la Notación de Objetos JavaScript (JSON) que es un formato de texto para estructurar e intercambiar datos de forma rápida, sencilla y liviana. Este formato es soportado por una gran variedad de lenguajes de programación y se lo considera una alternativa mucho más ligera en comparación al modelo propuesto por XML.

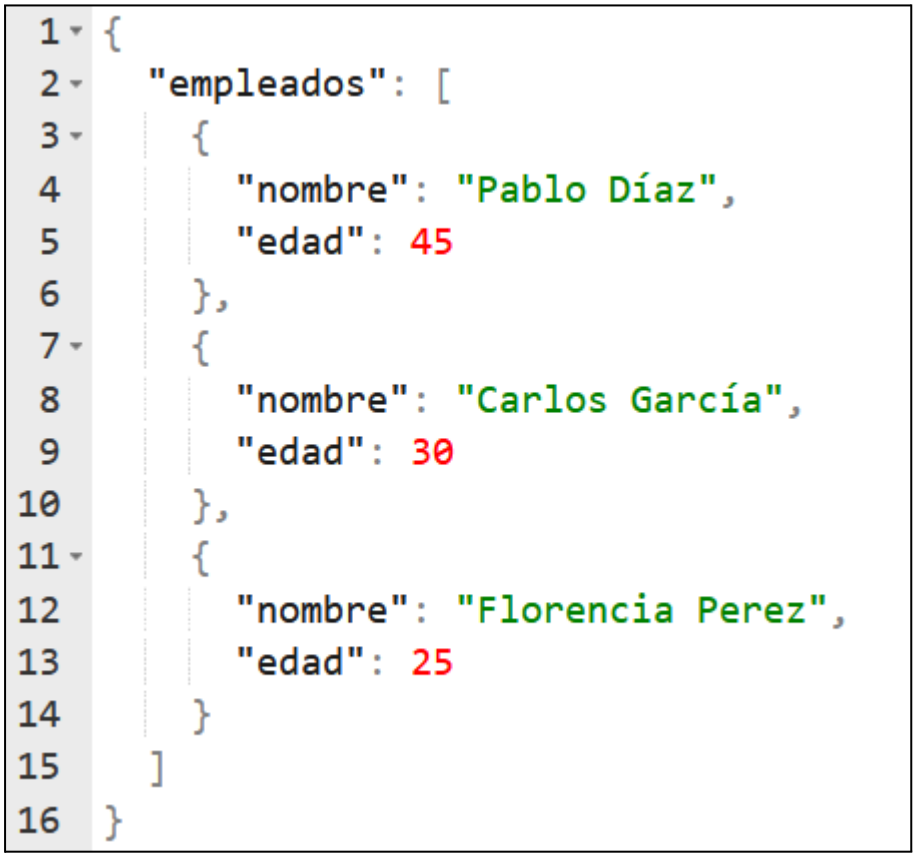
Con JSON los datos se estructuran como colecciones de pares clave/valor o como listas de valores ordenados, donde:

- **Clave:** corresponde al identificador del contenido y siempre es un string delimitado por comillas.
- **Valor:** representa el contenido correspondiente y puede contener los siguientes tipos de datos: string, array, object, number, boolean o null.

La representación de los datos debe seguir la siguiente notación específica:

- **Llaves { }** sirven para delimitar los objetos y son obligatorias para iniciar y terminar el contenido.
- **Corchetes [ ]** se usan para indicar un array.
- **Dos puntos :** sirven para separar la clave y su valor correspondiente.
- **Coma ,** se usa para indicar la separación entre los elementos.

La figura 3.6 muestra un ejemplo de mensaje en formato JSON.



```json
{
  "empleados": [
    {
      "nombre": "Pablo Díaz",
      "edad": 45
    },
    {
      "nombre": "Carlos García",
      "edad": 30
    },
    {
      "nombre": "Florencia Perez",
      "edad": 25
    }
  ]
}
```

Figura 3.6. Ejemplo de Mensaje JSON

Capítulo 4. Caso de Estudio: Implementación de un Middleware para la Interacción con un Sistema Legacy

En este capítulo se presenta un caso real de modernización de un sistema legacy aplicando la técnica de wrapping funcional. El capítulo comienza con la descripción de la aplicación legacy, sus características generales y los componentes principales de su arquitectura. Luego se plantea la necesidad de modernización del sistema y la solución adoptada mediante la implementación de un middleware o wrapper funcional para poder dotar al sistema con la nueva funcionalidad requerida. Al final del capítulo se presentan los resultados obtenidos con esta solución.

4.1 Contexto del Sistema Legacy

La Dirección General de Sistemas Informáticos de Administración Financiera (DGSIAF) es un área perteneciente a la Subsecretaría de Presupuesto de la Secretaría de Hacienda de la Nación. La función de este organismo es promover el desarrollo y mantenimiento de los sistemas que dan soporte a la gestión presupuestaria, financiera y contable del sector público nacional. Entre los sistemas desarrollados por la DGSIAF se encuentra el Sistema de Unidades Ejecutoras de Préstamos Externos (UEPEX) que se utiliza para administrar los fondos provistos por Organismos Internacionales de Crédito, para la ejecución de programas o proyectos específicos dentro del país. El principal objetivo de este sistema es asistir a los usuarios en la formulación y ejecución de los presupuestos que se otorgan a cada Unidad Ejecutora (UE) de Préstamos Externos.

La figura 4.1 muestra el ecosistema de aplicaciones de administración financiera entre las que se incluye el sistema UEPEX.

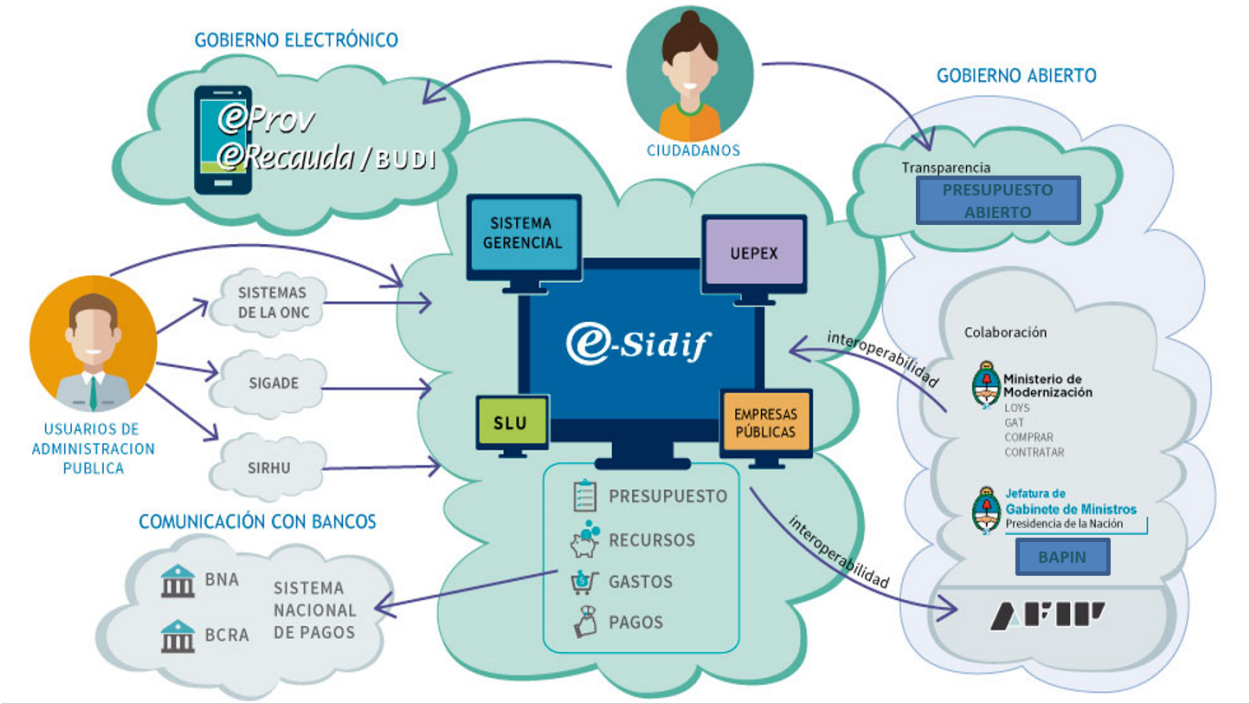


Figura 4.1. Sistema UEPEX dentro del ecosistema de Administración Financiera

4.2 Características y Arquitectura del Sistema Legacy

UEPEX es un sistema desktop desarrollado e implementado a principios de la década del 90. Para su desarrollo se utilizaron las siguientes tecnologías:

- **Visual Basic:** Fue el lenguaje de programación seleccionado para implementar la interfaz gráfica de usuario.
- **SQL Server:** Fue el motor de base de datos elegido para manejar la persistencia de datos y la implementación de Stored Procedures.
- **Windows Server:** Servidor utilizado para alojar la aplicación.

- **Citrix:** Herramienta utilizada para proveer la virtualización de la aplicación y permitir el acceso desde cualquier PC de escritorio.

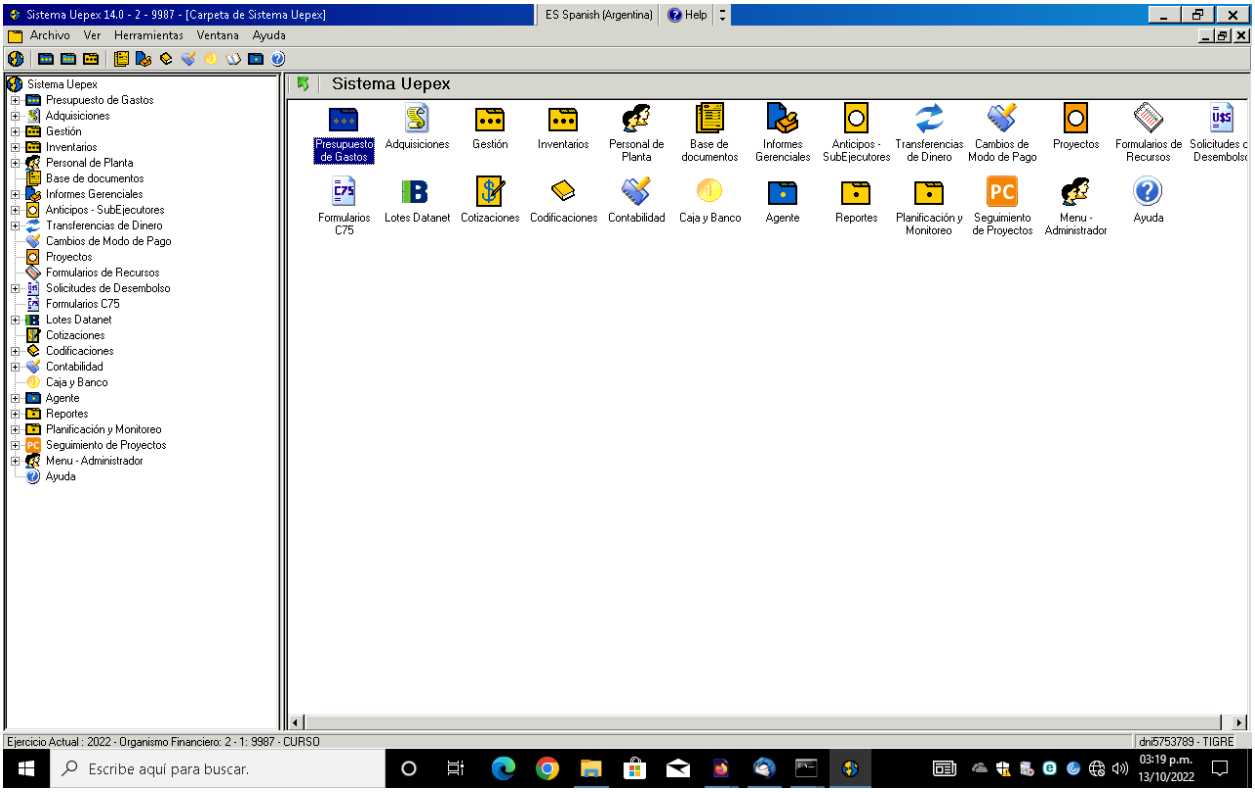


Figura 4.2. Pantalla Principal del Sistema UEPEX

La arquitectura del sistema (figura 4.3) se implementó siguiendo un modelo cliente/servidor de 2 capas:

- **Cliente Uepex:** Implementa la GUI para poder interactuar con el sistema. Se comunica con las bases de datos *uepex local* y *uepex accesos* para delegar en estas las peticiones que realizan los usuarios.
- **Base de Datos Uepex Local:** Almacena información sobre los préstamos externos que administra una Unidad Ejecutora. Contiene *stored procedures* que implementan la lógica de negocios necesaria para atender las peticiones que se

reciben desde el cliente. Se comunica con la base de datos *uepex accesos* para acceder a información centralizada.

- **Base de Datos Uepex Accesos:** Contiene datos y *stored procedures* que necesitan mantenerse de forma centralizada para que puedan accederse desde los clientes o desde las bases de datos locales.

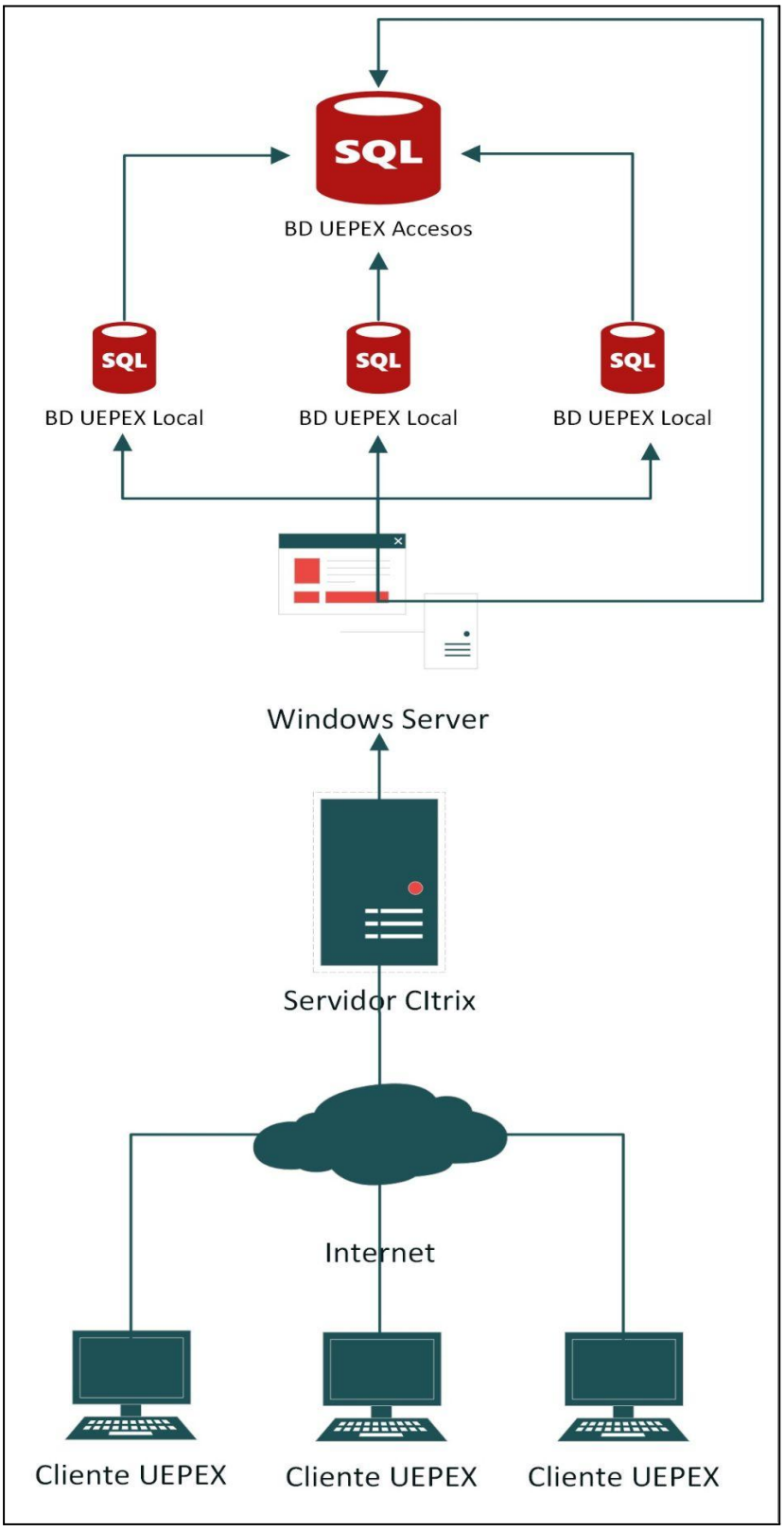


Figura 4.3. Arquitectura Sistema UEPEX

Actualmente se encuentran implementadas un total de 255 Unidades Ejecutoras de Préstamos Externos. El sistema registra un total de 1069 altas de usuarios, de los cuales acceden concurrentemente en horas pico unos 219 de ese total.

Desde hace varios años el sistema se encuentra en fase de mantenimiento. Como parte de ese mantenimiento se realizan correcciones de bugs y algunos ajustes menores en la funcionalidad del sistema. Estos ajustes, por lo general, son requeridos por la Mesa de Ayuda de la DGSIAF que mantiene contacto directo con los usuarios de la aplicación, y por lo tanto, va observando las problemáticas en el funcionamiento del sistema.

Por las características del sistema UEPEX, y de acuerdo con lo expuesto en el capítulo 2, se puede inferir que dicho sistema se encuadra dentro de la clasificación de sistema legacy.

4.3 Motivación del Problema

En el año 2017 surgió la necesidad de vincular el sistema UEPEX con el módulo de Locaciones y Servicios (LOyS), uno de los módulos que conforman el Sistema de Gestión Documental Electrónica (GDE). La figura 4.4 muestra los módulos principales que forman parte del sistema GDE. Este sistema se encuentra bajo la órbita de la Subsecretaría de Innovación Administrativa perteneciente a la Jefatura de Gabinete de Ministros de la Nación.

El módulo LOyS se utiliza para la tramitación de los contratos de las personas físicas o jurídicas que prestan servicios bajo los regímenes de Locación de Obras, Locación de Servicios, Asistencias Técnicas y PNUD[4] dentro del ámbito de la Administración Pública Nacional.

[4] El Programa de las Naciones Unidas para el Desarrollo (PNUD) es un programa que impulsa la ONU para mejorar la calidad de vida de las naciones. A través de este programa se financian proyectos que requieren la contratación de personal para poder implementar la ejecución de estos proyectos.

La necesidad de vinculación surgió debido a que las contrataciones de tipo PNUD necesitaban financiarse a través de programas de préstamos externos. UEPEX es el sistema que contiene los datos presupuestarios y financieros que son requeridos para la ejecución de estos programas; LOyS, por su parte, contiene los datos necesarios para la tramitación y ejecución de los contratos.

Para cumplir con este requerimiento, ambos sistemas necesitaban intercambiar datos para que pueda llevarse a cabo la ejecución y el pago de los contratos a los prestadores.

Este nuevo requerimiento representó un gran desafío para el equipo UEPEX, ya que dicho sistema se encuentra implementado íntegramente con stored procedures, mientras que el sistema LOyS es un sistema Web que fue concebido para interactuar con otros sistemas a través de Internet.

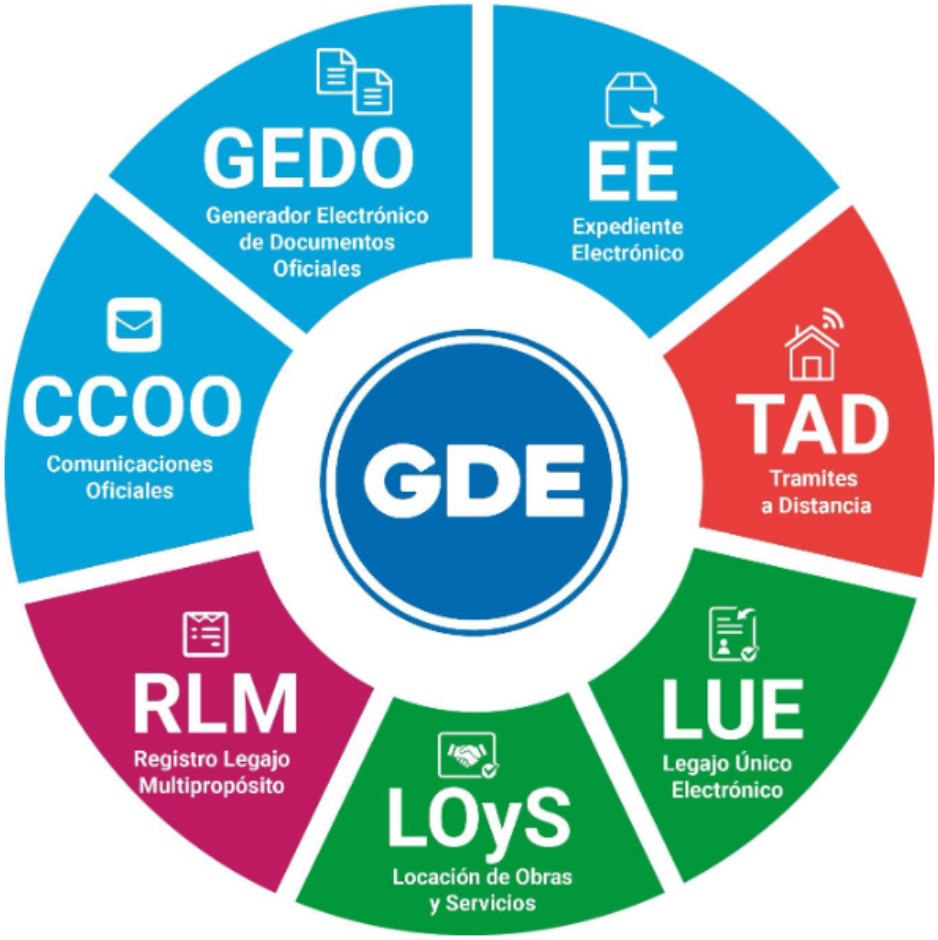


Figura 4.4. Módulos Principales del Sistema GDE

4.4 Planteo de la Vinculación Funcional UEPEX - LOyS

Debido a las grandes diferencias entre los sistemas se definió que cada uno mantenga la gestión de sus propios comprobantes. Cada sistema ejecuta sus propias gestiones y en determinados puntos del flujo de ejecución se esperan los datos o eventos del otro sistema que permiten continuar con la gestión. La figura 4.5 muestra la vinculación funcional planteada entre ambos sistemas.

En UEPEX el circuito comienza con la carga de la **Solicitud de Contratación**. Luego, el usuario procede con la habilitación de la solicitud y en ese momento se valida contra el módulo EE del sistema GDE que el expediente electrónico ingresado sea un dato válido en ese sistema. Una vez que la solicitud se encuentra habilitada se cargan los datos del **Contrato** y se le asocia la solicitud correspondiente. Posteriormente se procede con la habilitación del contrato en donde se le solicitan a LOyS los datos cargados en ese sistema para validar que coincidan con los datos del contrato UEPEX. Si la habilitación tuvo éxito se envían a LOyS los datos del contrato con la información de las partidas presupuestarias que financiarán el gasto. A partir de ese momento UEPEX queda a la espera de que el sistema LOyS genere automáticamente las **Facturas** que saldarán el contrato. El usuario UEPEX genera la **Liquidación** para cada factura de acuerdo al plan de pagos indicado. Como resultado de la liquidación se genera el comprobante de **Pago**. Cuando el usuario ejecuta la habilitación del pago, se notifica a LOyS la información del mismo.

En LOyS el circuito comienza con la generación del **Expediente Electrónico** en el módulo EE del sistema GDE. Luego, desde el módulo LOyS, se procede con la carga de datos para la **Solicitud de Locación**. Una vez completada la solicitud se espera la recepción del contrato UEPEX para verificar que los datos ingresados en ese sistema coincidan con los datos indicados en la solicitud. Si los datos son correctos, se procede con la **Firma del Contrato**. Cuando se completan todas las firmas el sistema habilita la ejecución del contrato a través de la **carga de una o más Facturas** que saldarán el monto total del contrato. La carga de la factura en LOyS dispara la generación automática de la factura correspondiente en UEPEX. Durante la carga de la factura, el usuario también tiene la posibilidad de dar de baja el contrato vigente, para lo cual el sistema LOyS se comunica con UEPEX para consultar si puede llevarse a cabo dicha baja. El flujo continúa hasta el punto de **confirmación de la factura**. Si no se confirma la factura, ésta se anula en LOyS y se dispara la anulación automática de la factura en UEPEX. Si la factura se confirma, LOyS queda a la **espera de que UEPEX envíe los datos del**

Pago. Al recibir los datos del pago, LOyS verifica si el contrato quedó saldado completamente, en cuyo caso, procederá con la liberación del expediente asociado. Si por el contrario, el contrato no ha sido saldado, el sistema esperará la carga de la próxima factura.

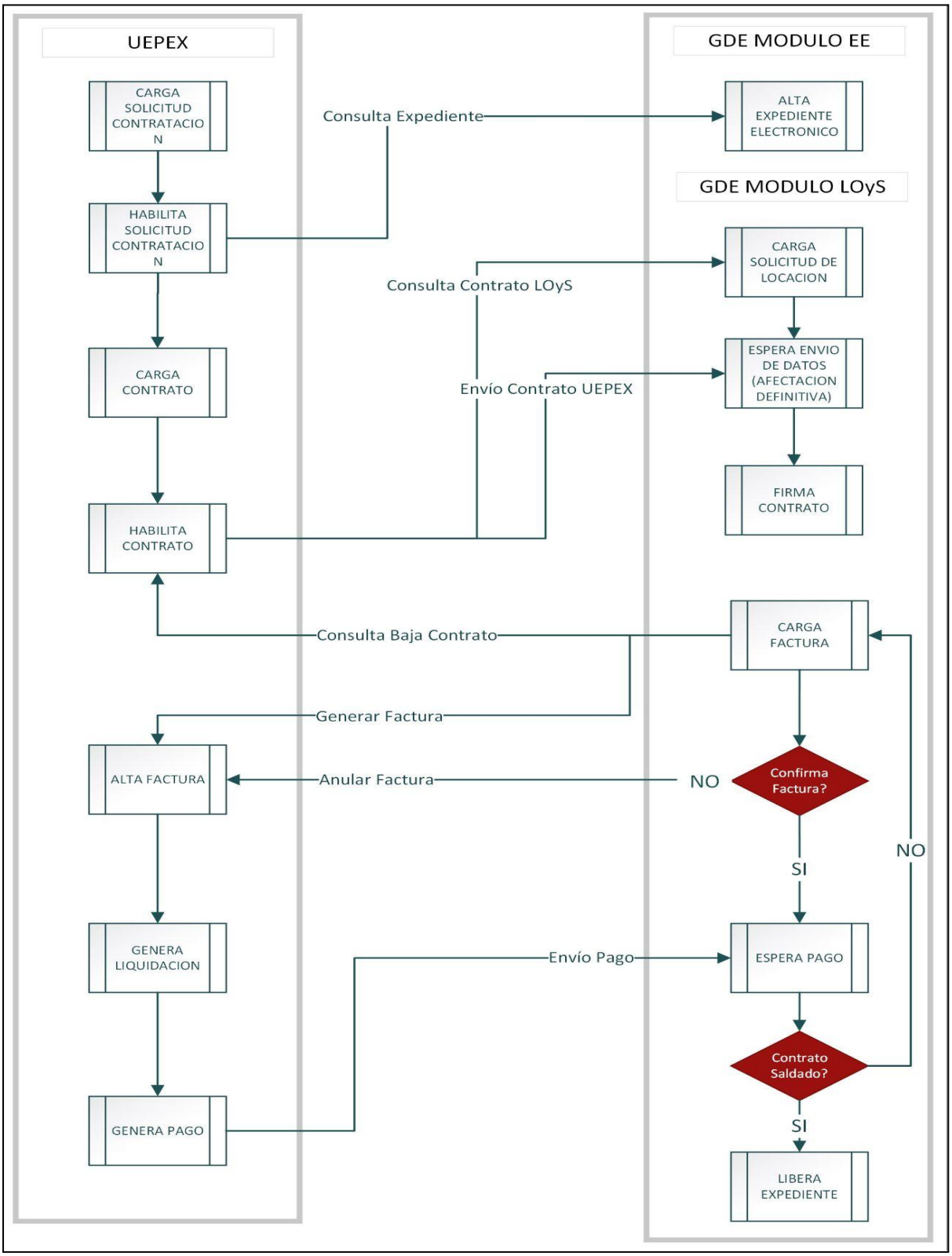


Figura 4.5. Vinculación Funcional UEPEX - LOyS

4.5 IPEX: Un Middleware para la Vinculación Funcional UEPEX - LOyS

Para resolver la comunicación entre ambos sistemas se optó por la utilización de Web Services SOAP. LOyS ya utilizaba esta tecnología para interactuar con otros sistemas de la Administración Pública Nacional. UEPEX, en cambio, no tenía resuelta la comunicación con otras aplicaciones ya que hasta ese momento no había tenido la necesidad de interactuar con otros sistemas.

Por la arquitectura y las tecnologías utilizadas en el desarrollo de UEPEX resultaba difícil la posibilidad de publicar servicios web dentro de la misma aplicación. Para resolver esta problemática se decidió poner en práctica un proceso de modernización del sistema, de manera de poder dotarlo con la nueva funcionalidad. Esta funcionalidad fue implementada en una nueva capa de software denominada IPEX (Interfaz uePEX) que actúa como un middleware entre el aplicativo UEPEX e Internet.

4.6 Arquitectura del Middleware

La figura 4.6 muestra la arquitectura planteada para resolver la vinculación funcional UEPEX-LOyS. Los principales componentes de esta arquitectura son:

- **IPEX:** Es una aplicación Web que expone una capa de servicios para operaciones UEPEX desde Internet. Esta aplicación también actúa como cliente invocando servicios LOyS en nombre de UEPEX. Para los servicios expuestos, la aplicación recibe las peticiones desde Internet y ejecuta las operaciones correspondientes en las bases de datos UEPEX. Para los servicios clientes, en

cambio, recibe las peticiones desde UEPEX y retransmite estas peticiones al sistema LOyS. La aplicación también interactúa con el sistema DAUT para poder autenticar y autorizar las peticiones que se realizan desde LOyS. Si bien esta aplicación surgió para resolver la integración UEPEX-LOyS, la misma podría utilizarse para integrar UEPEX con otros sistemas externos.

- **UEPEX:** Interactúa con la capa IPEX para poder vincularse con los servicios de LOyS. Para ello, invoca a los servicios en IPEX desde las bases de datos UEPEX, y a su vez, ejecuta los stored procedures que se invocan desde IPEX.
- **DAUT:** Es una aplicación Web que se utiliza para autenticar y autorizar los accesos a los sistemas de la DGSIAF. IPEX delega en esta aplicación la autenticación y autorización de los requests que provienen desde Internet.

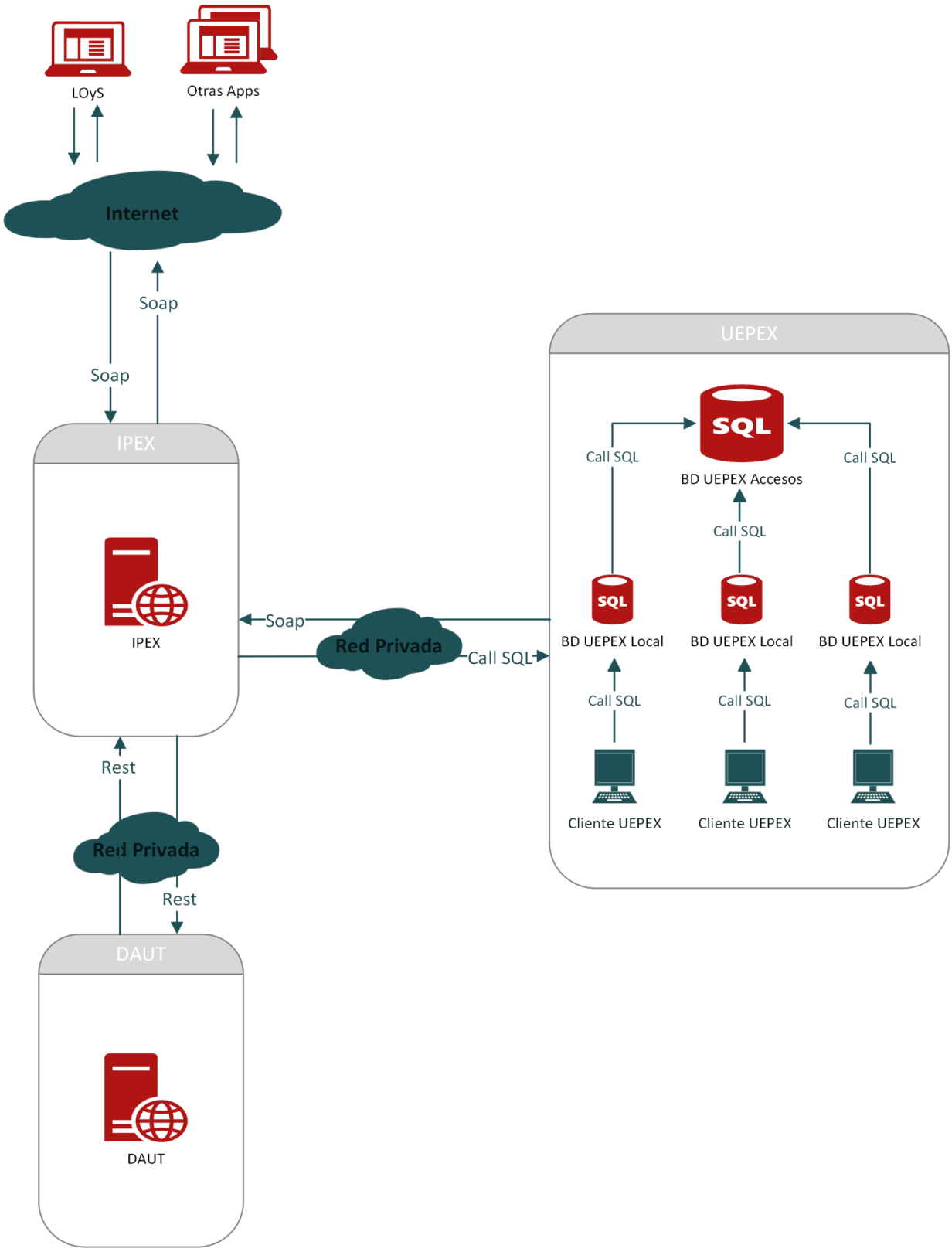


Figura 4.6. Arquitectura del Middleware

4.7 Funcionamiento del Middleware

Para que UEPEX pueda invocar Web Services externos se implementó un stored procedure en la base de datos Uepex Accesos denominado *Soap Invoker*. Este procedimiento contiene la lógica necesaria para realizar las invocaciones SOAP desde la base de datos SQL. De esta manera, cuando una uepex local necesita invocar a un servicio externo delega dicha invocación en la base de datos Uepex Accesos. El Soap Invoker interactúa con IPEX realizando invocaciones a los servicios privados implementados en dicho sistema. Estos servicios sólo están disponibles para ser accedidos localmente desde UEPEX. Cuando un servicio privado recibe un request desde UEPEX lo retransmite al sistema externo correspondiente. Del mismo modo, cuando recibe un response desde un sistema externo lo retransmite al Soap Invoker. La figura 4.7 muestra la interacción que se lleva a cabo al invocar un web service externo desde UEPEX.

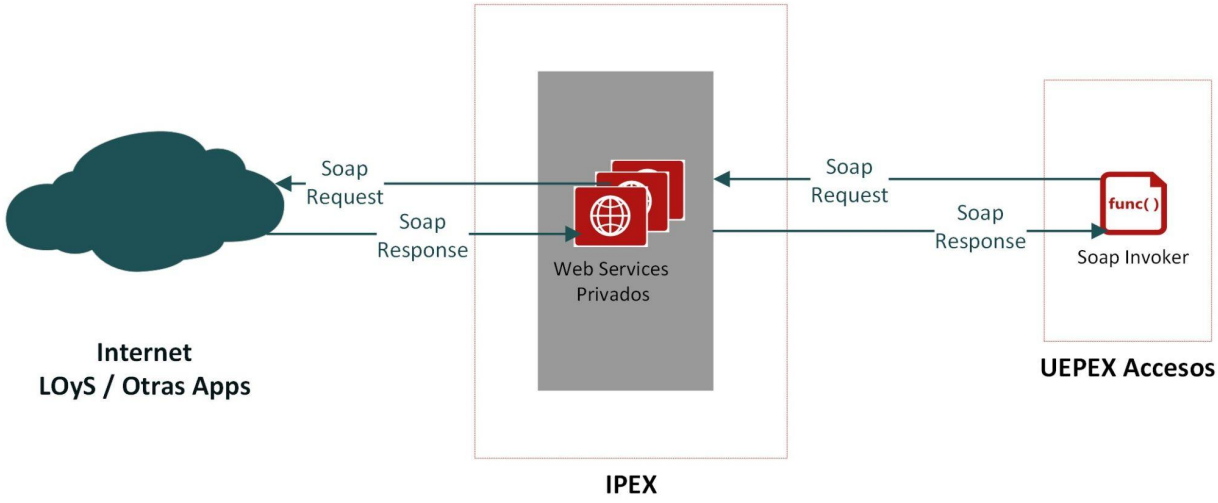


Figura 4.7. UEPEX Invoca un Web Service Externo

Por otro lado, para que UEPEX pueda publicar Web Services a otros sistemas se implementaron en IPEX otro tipo de servicios denominados públicos. Estos servicios, a diferencia de los privados, están disponibles para ser accedidos a través de Internet. De esta manera, cuando un sistema externo necesita invocar a un servicio de UEPEX realiza una invocación a uno de los servicios públicos implementados en IPEX. Estos servicios, al recibir los requests desde Internet delegan la ejecución en los stored procedures de la base de datos Uepex Accesos y posteriormente envían el response correspondiente al sistema externo que realizó la invocación. La figura 4.8 muestra la interacción que se lleva a cabo al invocar un web service publicado por UEPEX.

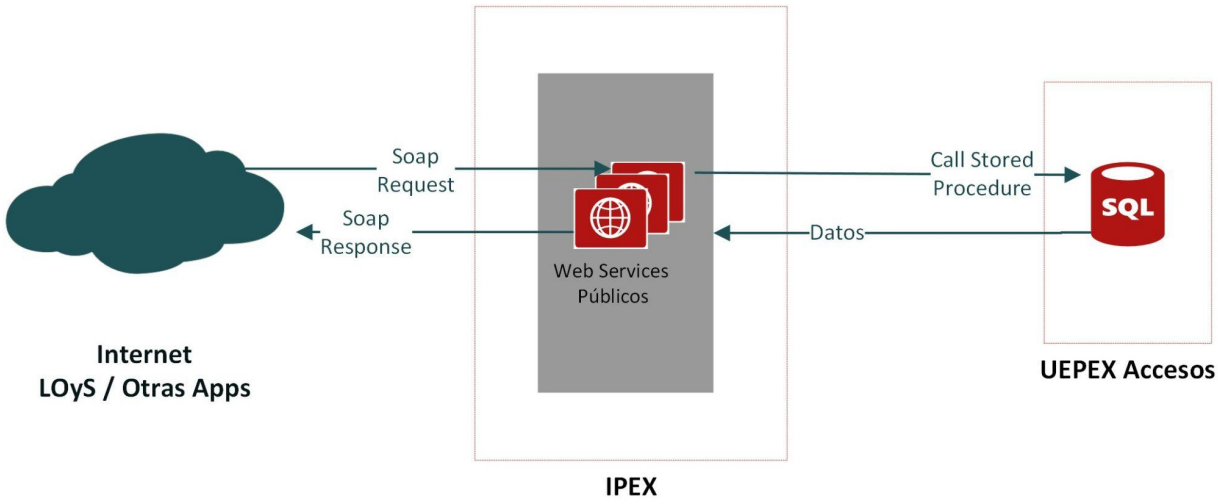


Figura 4.8. Sistema Externo Invoca un Web Service de UEPEX

4.8 Stored Procedure SoapInvoker

El stored procedure soapInvoker es el procedimiento que se utiliza para poder enviar peticiones SOAP desde la base de datos UEPEX hacia la aplicación IPEX. La figura 4.9 muestra el código SQL con la implementación realizada para este procedimiento.

Este stored procedure recibe como parámetros de entrada la URL del servidor al que se quiere acceder, el request XML que se debe transmitir, los datos para la autenticación (username y password) y el tiempo máximo de espera para recibir una respuesta desde el servidor. Como parámetros de salida el procedimiento retorna uno de los siguientes datos: el response XML, si la operación fue correcta, o un mensaje de error, si se produjo un fallo en la operación.

Para poder realizar las peticiones HTTP el procedimiento utiliza el objeto **MSXML2.ServerXMLHTTP** que es un objeto que provee el motor de base de datos de SQL Server. Este objeto contiene métodos y propiedades para poder establecer una conexión HTTP con otro servidor. El soapInvoker interactúa con este objeto utilizando los stored procedures que el motor de base de datos provee para poder realizar operaciones con los objetos del sistema. Para esta implementación se utilizaron cinco de estos procedimientos:

- **sp_OACreate:** Utilizado para crear una instancia del objeto.
- **sp_OAMethod:** Utilizado para invocar un método del objeto creado.
- **sp_OAGetProperty:** Utilizado para acceder a una propiedad del objeto creado.
- **sp_OAGetErrorInfo:** Utilizado para obtener datos de error sobre el objeto creado.
- **sp_OADestroy:** Utilizado para eliminar la instancia del objeto creado.

```
USE [master] GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

/* Procedure [dbo].[sp_SoapInvoker] Permite invocar web services SOAP. Hace
un request al host indicado por @uri y retorna el resultado en el xml
```

```
@responsexml. En caso de producirse un error en la conexión o durante la
comunicación, o un fault, se retorna el mismo en @errorMsg.
Fecha: 09/09/2017 */
CREATE PROCEDURE [dbo].[sp_SoapInvoker]
    @uri nvarchar(200) = '',
    @requestBody xml = '',
    @userName nvarchar(100),
    @password nvarchar(100),
    @responsexml xml output,
    @errorMsg nvarchar(max) output,
    @receiveTimeout int = 60000
AS
SET NOCOUNT ON

DECLARE @methodName varchar(50)
DECLARE @objectID int
DECLARE @hResult int
DECLARE @exceptionMsg varchar(200)
DECLARE @faultExists bit
DECLARE @faultcode nvarchar(100)
DECLARE @faultstring nvarchar(max)
DECLARE @httpStatus int
DECLARE @t table (ID int, strxml xml)

SET @errorMsg = ''
SET @responsexml = 'FAILED'
SET @methodName = 'POST'

-- Se crea el objeto MSXML2.ServerXMLHTPP para encapsular el acceso
-- HTTP con el host remoto. En @objectID se guarda el ID del objeto creado
EXEC @hResult = sp_OACreate 'MSXML2.ServerXMLHTTP', @objectID OUT
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'create failed'
    GOTO destroy_and_raise
END

-- Se ejecuta el método setTimeouts() en el objeto creado para setear
-- los timeouts de la operación
EXEC @hResult = sp_OAMethod @objectID, 'setTimeouts', null, 60000, 60000,
30000, @receiveTimeout
IF @hResult <> 0
BEGIN
```

```
    SET @exceptionMsg = 'setTimeouts failed'
    GOTO destroy_and_raise
END
-- Se ejecuta el método open() en el objeto HTTP para proporcionar los
-- datos de inicialización del request
EXEC @hResult = sp_OAMethod @objectID, 'open', null, @methodName, @uri,
'false', @userName, @password
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'open failed'
    GOTO destroy_and_raise
END

-- Se ejecuta el método setRequestHeader() en el objeto HTTP para
-- indicar el Content-Type en el header del mensaje
EXEC @hResult = sp_OAMethod @objectID, 'setRequestHeader', null,
'Content-Type', 'text/xml;charset=UTF-8'
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'setRequestHeader failed'
    GOTO destroy_and_raise
END

-- Se ejecuta el método setRequestHeader() en el objeto HTTP para
-- indicar el Content-Lenght en el header del mensaje
declare @len int
SET @len = DATALENGTH(@requestBody)
EXEC @hResult = sp_OAMethod @objectID, 'setRequestHeader', null,
'Content-Length', @len
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'setRequestHeader failed'
    GOTO destroy_and_raise
END

-- Se ejecuta el método send() en el objeto HTTP para enviar el request
-- al host indicado
EXEC @hResult = sp_OAMethod @objectID, 'send', null, @requestBody
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'send failed'
    GOTO destroy_and_raise
END
```

```
-- Se obtiene la property que contiene el status HTTP de la operación
EXEC @hResult = sp_OAGetProperty @objectID, 'status', @httpStatus OUT
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'status failed'
    GOTO destroy_and_raise
END

-- Se obtiene la property que contiene el response de la operación
Insert into @t (strxml) exec @hResult = sp_OAGetProperty @objectID,
'responseXML.xml'
IF @hResult <> 0
BEGIN
    SET @exceptionMsg = 'responseXML failed'
    GOTO destroy_and_raise
END

-- Se construye el errorMsg si el HTTP status retornó error
SELECT @responsexml = strxml from @t
if (@httpStatus < 200) or (@httpStatus > 299)
BEGIN
    ;WITH XMLNAMESPACES ( 'http://schemas.xmlsoap.org/soap/envelope/' as
    soap)
    SELECT @faultExists =
    @responsexml.exist('(/soap:Envelope/soap:Body/soap:Fault)')

    if @faultExists = 1
    BEGIN
        ;WITH XMLNAMESPACES ( 'http://schemas.xmlsoap.org/soap/envelope/'
        as soap)
        SELECT @faultcode = @responsexml.value('(/soap:Envelope/soap:Body
        /soap:Fault/faultcode)[1]', 'nvarchar(100)')

        ;WITH XMLNAMESPACES ( 'http://schemas.xmlsoap.org/soap/envelope/'
        as soap)
        SELECT @faultstring = @responsexml.value('(/soap:Envelope/soap:Body
        /soap:Fault/faultstring)[1]', 'nvarchar(max)')

        SET @errorMsg = @faultcode + ' - ' + @faultstring
    END
    ELSE
        SET @errorMsg = 'HTTP Status: ' + CAST(@httpStatus as varchar)
```

```
END

-- Se elimina el objeto HTTP creado
EXEC sp_OADestroy @objectID
GOTO end_label

destroy_and_raise:
    DECLARE @sourceErr varchar(255), @descErr varchar(255)
    EXEC sp_OAGetErrorInfo @objectID, @sourceErr OUT, @descErr OUT
    EXEC sp_OADestroy @objectID
    SET @errorMsg = @exceptionMsg + '-' + @sourceErr + '-' + @descErr

end_label:
    SET NOCOUNT OFF
    SET QUOTED_IDENTIFIER ON
GO
```

Figura 4.9. Stored Procedure SoapInvoker en UEPEX ACCESOS

4.9 Invocando un Web Service desde UEPEX

Debido a que las invocaciones a web services externos fueron resueltas todas de la misma manera, solo se presentará una de las implementaciones realizadas. Esta implementación muestra cómo se resolvió la invocación al servicio de **Consulta de Expediente Electrónico** en el sistema GDE.

4.9.1 Web Service Consulta de Expediente Electrónico

El WebService de Consulta de Expediente Electrónico es un servicio provisto por el sistema GDE para que otros sistemas externos puedan consultar información sobre los expedientes electrónicos que se tramitan en dicho sistema.

Este servicio recibe como entrada un código de expediente electrónico y retorna como salida los principales datos para ese expediente.

Un Expediente Electrónico se identifica dentro de GDE mediante un código compuesto por los siguientes campos : **[Tipo de Actuación] – [Año de Actuación] – [Número de Actuación] – [Secuencia de tres espacios vacíos] – [Ecosistema] – [Repartición].** A continuación se detallan cada uno de estos campos:

- **Tipo de Actuación:** Identifica el tipo de documento al que pertenece la actuación. Para expedientes electrónicos se utiliza el código EX.
- **Año de Actuación:** Indica el año en el que se crea la actuación.
- **Número de Actuación:** Corresponde a un número de 8 dígitos que el sistema genera de forma automática y secuencial.
- **Secuencia de tres espacios vacíos:** es un atributo que tiene significado funcional en la tramitación de actuaciones papel. Para las actuaciones electrónicas la secuencia siempre es nula.
- **Ecosistema:** Corresponde al ámbito donde se tramita la actuación. Para la instancia del GDE instalada en la Administración Pública Nacional se utiliza siempre el código APN.
- **Repartición:** Identifica el código de la repartición. Este valor está compuesto por las iniciales de las palabras que componen el nombre del área donde se inicia la actuación, seguido del símbolo numeral y a continuación las iniciales del organismo de dependencia.

Por ejemplo, la identificación **EX-2022-08395999- -APN-DGPFE#MS** referencia un expediente electrónico del año 2022, bajo el número de actuación 08395999, en la instancia GDE de la Administración Pública Nacional (APN) y gestionado por la Dirección General de Programas con Financiamiento Externo (DGPFE) del Ministerio de Salud de la Nación (MS).

A continuación se presenta un ejemplo de request y response para el servicio de Consulta de Expediente Electrónico junto con la descripción de los parámetros de entrada y salida utilizados por este servicio.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ar="http://ar.gob.gcaba.ee.services.external/">
   <soapenv:Header />
   <soapenv:Body>
      <ar:consultaEE>
         <codigoEE>EX-2022-08395999- -APN-DGPFE#MS</codigoEE>
      </ar:consultaEE>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 4.10. Ejemplo de Request SOAP para el Web Service Consulta de Expediente Electrónico

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:consultaEEResponse
       xmlns:ns2="http://ar.gob.gcaba.ee.services.external/">
         <return>
            <codigoEE>EX-2022-08395999- -APN-DGPFE#MS</codigoEE>
            <codigotrata>GENE00329</codigotrata>
            <descripcionTrata>Contrato PNUD</descripcionTrata>
            <estado>Tramitacion</estado>
            <listDocumentosOficiales>DOCPE-2022-17794407-APN-DTD#JGM
            </listDocumentosOficiales>
            <listDocumentosOficiales>PV-2022-26480376-APN-DGPFE#MS
            </listDocumentosOficiales>
            <listDocumentosOficiales>IF-2022-17183604-APN-DGPFE#MS
            </listDocumentosOficiales>
            <listDocumentosOficiales>ACTO-2022-47163663-APN-JGM
            </listDocumentosOficiales>
            <sistemaOrigen>LOyS</sistemaOrigen>
         </return>
      </ns2:consultaEEResponse>
   </soap:Body>
</soap:Envelope>
```

Figura 4.11. Ejemplo de Response SOAP para el Web Service Consulta de Expediente Electrónico

Parámetro de Entrada	Tipo de Dato	Descripción	Ejemplo
codigoEE	String	Código del expediente a consultar	EX-2022-08395999--APN-DGPFE#MS

Tabla 4.1. Parámetros de Entrada para el Web Service Consulta de Expediente Electrónico

Parámetro de Salida	Tipo de Dato	Descripción	Ejemplo
codigoEE	String	Código del expediente consultado	EX-2022-08395999--APN-DGPFE#MS
codigoTrata	String	Código de trámite del expediente	GENE00329
descripcionTrata	String	Descripción del código de trámite que sirve para facilitar la identificación del tipo de trámite	Contrato PNUD
estado	String	Estado en el que se encuentra el expediente	Tramitación
ListDocumentos Oficiales	String	Lista que contiene los documentos oficiales asociados al expediente.	ACTO-2022-47163663-APN-JGM
sistemaOrigen	String	Código del sistema creador	LOyS

Tabla 4.2. Parámetros de Salida para el Web Service Consulta de Expediente Electrónico

4.9.2 Implementado la Invocación al Web Service

4.9.2.1 Creación de un Stored Procedure en la base UEPEX ACCESOS

El primer paso para la solución consistió en la implementación de un stored procedure en la base de datos UEPEX ACCESOS denominado **spConsultaEE**. La figura 4.12 muestra la implementación realizada para este procedimiento. Este stored procedure contiene la lógica necesaria para poder iniciar la consulta de un expediente electrónico dentro de UEPEX. La funcionalidad fue implementada en la base de datos UEPEX ACCESOS ya que necesitaba poder invocarse desde cualquiera de las uepex locales que realizan gestiones PNUD.

El procedimiento spConsultaEE recibe como parámetros de entrada la identificación del expediente electrónico que se quiere consultar y el usuario que realiza la consulta. Luego construye el request correspondiente con los datos recibidos y delega en el stored procedure **soapInvoker** la invocación al web service externo. Al finalizar la ejecución retorna como parámetros de salida un valor booleano indicando el resultado de la consulta y el mensaje de error en caso de que se reciba una respuesta errónea .

```
USE [UEPEX_ACCESOS]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

/*
-- Consulta un expediente electrónico en GDE usando un ws de IPEX.
-- Retorna en @existeEE si el expediente existe o no.
-- Si el expediente no existe, o cualquier otro error se informa en
@errorMsg.
*/
ALTER PROCEDURE [dbo].[spConsultaEE]
      @ee nvarchar(100),
      @existeEE bit output,
      @baseUsuario varchar(128),
      @errorMsg nvarchar(200) output
AS
   SET NOCOUNT ON
   DECLARE @consultaEERequest xml;
```

```
DECLARE @uri nvarchar(200);
DECLARE @responseText xml;

SET @existeEE = 0
SET @errorMsg = ''

-- Se inicializa el request con los datos del expediente y usuario
-- recibidos por parámetro
SET @consultaEERequest = '
   <soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ar="http://ar.gob.gcaba.ee.services.external/">
      <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
         <wsa:Action>ar.gob.mecon.dgsiaf.ipex.ws.expediente.consultaEE
         </wsa:Action>
         <wsa:MessageID>uuid:1</wsa:MessageID>
         <wsuf:usuarioFisico login="' + @baseUsuario + '"
         xmlns:wsuf="http://ws-web.mecon.ar/ipex/ws/security/"/>
      </soapenv:Header>
      <soapenv:Body>
         <ar:consultaEE>
         <codigoEE>' + @ee + '</codigoEE>
         </ar:consultaEE>
      </soapenv:Body>
   </soapenv:Envelope>'

-- Se obtiene la URL del endPoint privado en IPEX a la cual se enviará
-- el request generado
select @uri = v_value from IPEX_PROPERTIES where c_key =
'webService.private.ipex.url'

-- Se invoca al stored procedure SoapInvoker para que envíe el request a
-- la URL indicada
exec sp_SoapInvoker @uri, @consultaEERequest, '', '', @responseText out,
@errorMsg out

-- Si no hubo errores se procesa la respuesta OK
IF (@errorMsg = '')
BEGIN
   ;WITH XMLNAMESPACES (
   'http://schemas.xmlsoap.org/soap/envelope/' as soap,
   'http://ar.gob.gcaba.ee.services.external/' as ns2)
   select @existeEE = @responseText.exist('(/soap:Envelope/soap:Body
```

```
    /ns2:consultaEEResponse/return/codigoEE[.=sql:variable("@ee")])')
END
```

Figura 4.12. Implementación del Stored Procedure para la Consulta de Expediente Electrónico en UEPEX ACCESOS

4.9.2.2 Implementación de un Endpoint Privado en IPEX

El siguiente paso para la solución fue el desarrollo de un Endpoint en la aplicación IPEX para que reciba los requests que se transmiten a través del soapInvoker y los reenvie al web service del GDE. La figura 4.13 muestra la implementación de esta funcionalidad en la clase **ExpedienteElectronicoPrivateEnpoint**. Este Endpoint está configurado para ser expuesto únicamente dentro de la red interna de la DGSIAF.

La annotation **@Endpoint** se utiliza para marcar a esta clase como un componente adecuado para el manejo de servicios web con Spring-WS.

Este servicio debe realizar invocaciones al Web Service de Consulta de Expediente Electrónico para reenviar los pedidos que recibe desde el sistema UEPEX. Para ello, interactúa con la clase **ExpedienteElectronicoClient** la cual se encuentra inyectada en la variable **client** con **@Autowired**.

El método **consultaEE()** implementa la funcionalidad requerida. Aquí la annotation **@SkipAuthentication** se utiliza para no autenticar los mensajes contra el sistema DAUT, dado que estos siempre provienen desde la aplicación local UEPEX y por lo tanto no requieren autenticación. La annotation **@Action** mapea la ejecución del método con todos los requests que contengan el valor del Action definido por la annotation. Las annotations **@RequestPayload** y **@RespoonsePayload** se utilizan para delegar en Spring-WS la conversión de datos entre XML y objetos Java.

```java
package ar.gob.mecon.dgsiaf.ipex.ws.expediente;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;
import org.springframework.ws.soap.addressing.server.annotation.Action;
import ar.gob.mecon.dgsiaf.ipex.ws.expediente.jaxb.ConsultaEE;
import ar.gob.mecon.dgsiaf.ipex.ws.expediente.jaxb.ConsultaEEResponse;
import ar.gob.mecon.dgsiaf.ipex.ws.interceptor.SkipAuthentication;

@Endpoint
public class ExpedienteElectronicoPrivateEndpoint {

  protected transient Logger LOGGER = LoggerFactory.getLogger(getClass());

  public static final String ACTION =
  "ar.gob.mecon.dgsiaf.ipex.ws.expediente.consultaEE";

  @Autowired
  private ExpedienteElectronicoClient client;

  public ExpedienteElectronicoPrivateEndpoint() {

  }

  @SkipAuthentication
  @Action(ACTION)
  @ResponsePayload
  public ConsultaEEResponse consultaEE(@RequestPayload ConsultaEE request)
{
      return this.getClient().consultaEE(request);
  }

  public ExpedienteElectronicoClient getClient() {
      return client;
  }

  public void setClient(ExpedienteElectronicoClient client) {
      this.client = client;
  }
```

```
}
```

Figura 4.13. Implementación del Endpoint Privado para la Consulta de Expediente Electrónico en IPEX

4.9.2.3 Implementación de un Cliente de Web Service en IPEX

El último paso para resolver la invocación fue implementar una clase que actúe como cliente del web service externo y ejecute la invocación desde IPEX. La figura 4.14 muestra la implementación de la clase **ExpedienteElectronicoClient** que es la clase encargada de implementar dicha funcionalidad. Aquí el método **consultaEE()** recibe por parámetro un objeto request y el framework Spring se encarga de realizar la invocación a través del **WebServiceTemplate**.

La URL del servicio se encuentra almacenada en una property del sistema y su valor se inyecta en la variable **expedienteWsUrl** cuando se construye la aplicación

La superclase **GdeWsClient** implementa el seteo del keystore y la password que se necesitan para autenticar los mensajes SOAP contra el servidor GDE. Dicha implementación se deja fuera de este trabajo ya que no es el objetivo mostrar la solución desde el punto de vista de la seguridad.

```
package ar.gob.mecon.dgsiaf.ipex.ws.expediente;

import javax.validation.constraints.NotBlank;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import ar.gob.mecon.dgsiaf.ipex.ws.expediente.jaxb.ConsultaEE;
import ar.gob.mecon.dgsiaf.ipex.ws.expediente.jaxb.ConsultaEEResponse;
import ar.gob.mecon.dgsiaf.ipex.ws.gde.GdeWsClient;

public class ExpedienteElectronicoClient extends GdeWsClient {
```

```
    private static final Logger LOGGER =
    LoggerFactory.getLogger(ExpedienteElectronicoClient.class);

    @NotBlank
    @Value("${webService.gde.consultaee.url:}")
    private String expedienteWsUrl;

    public ConsultaEEResponse consultaEE(ConsultaEE request) {
        ConsultaEEResponse response = null;
        LOGGER.info("Consultando expediente: " + request.getCodigoEE());
        response = (ConsultaEEResponse) this.getWebServiceTemplate().
        marshalSendAndReceive(getExpedienteWsUrl(), request);

        return response;
    }

    public String getExpedienteWsUrl() {
        return expedienteWsUrl;
    }

    public void setExpedienteWsUrl(String expedienteWsUrl) {
        this.expedienteWsUrl = expedienteWsUrl;
    }
}
```

Figura 4.14. Implementación del Cliente de Web Service para la Consulta de Expediente Electrónico en IPEX

Las conversiones de datos entre XML y los objetos java se realizan automáticamente mediante un objeto Marshaller. Para ello, se definieron dos beans de Spring en la clase **IpexWsClientConfiguration:** un bean **Jaxb2Marshaller** para convertir los mensajes desde/hacía XML y un bean **ExpedienteElectronicoClient** al cual se le asigna el bean de Marshaller. En la figura 4.15 se presenta la implementación de esta configuración.

```
package ar.gob.mecon.dgsiaf.ipex.context;
```

```java
import javax.validation.constraints.NotBlank;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;
import ar.gob.mecon.dgsiaf.ipex.ws.contrato.ContratoWSClient;
import ar.gob.mecon.dgsiaf.ipex.ws.expediente.ExpedienteElectronicoClient;
import ar.gob.mecon.dgsiaf.ipex.ws.pg.PagoWSClient;

@Configuration
public class IpexWsClientConfiguration {

    @Bean
    public Jaxb2Marshaller marshaller() throws Exception {
       Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
       marshaller.setContextPaths(
           "ar.gob.mecon.dgsiaf.ipex.ws.expediente.jaxb"
            // Otros contextPaths);
       return marshaller;
    }

    @Bean
    public ExpedienteElectronicoClient expedienteElectronicoClient
     (Jaxb2Marshaller marshaller) {
       ExpedienteElectronicoClient client = new
                                        ExpedienteElectronicoClient();
       client.setMarshaller(marshaller);
       client.setUnmarshaller(marshaller);

       return client;
    }

  /**
   * Otras configuraciones correspondientes a otros WS Clients
   */
}
```

Figura 4.15. Configuración del Cliente de Web Service para la Consulta de Expediente Electrónico en IPEX

4.10 Publicando un Web Service en UEPEX

Al igual que en la invocación a servicios externos, solo se presentará una de las implementaciones realizadas para la publicación de Web Services en UEPEX. Los demás servicios publicados siguen la misma lógica de solución. La implementación descripta en esta sección muestra la solución para publicar el Web Service de **Consulta de Contrato Anulado**.

4.10.1 Web Service Consulta de Contrato Anulado

El Web Service de Consulta de Contrato Anulado es un servicio que el sistema UEPEX provee para que LOyS pueda consultar si un contrato específico ha sido anulado en dicho sistema.

Este Web Service recibe como parámetros de entrada un identificador de expediente electrónico y los datos del beneficiario del contrato. Como parámetros de salida el servicio retorna un único valor booleano indicando el resultado de la consulta.

Las figuras 4.16 y 4.17 muestran un ejemplo de request y response para este servicio. Seguidamente, las tablas 4.3 y 4.4 describen cada uno de los parámetros de entrada y salida.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sch="http://dgsiaf.mecon.gob.ar/ipex/schemas">
   <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
      oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand=
      "1">
         <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004
         /01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id=
         "UsernameToken-1">
            <wsse:Username>XXXXX</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/
            oasis-200401-wss-username-token-profile-1.0#Password
            Text">XXXXX</wsse:Password>
```

```
         </wsse:UsernameToken>
      </wsse:Security>
      <wsa:To soapenv:mustUnderstand="1">http://localhost:8080/ipex/ws/
      </wsa:To>
      <wsa:Action>ar.gob.mecon.dgsiaf.ipex.ws.ConsultaAnulacionDe
      Contrato</wsa:Action>
   </soapenv:Header>
   <soapenv:Body>
      <sch:ConsultarAnulacionDeContratoIpexRequest>
         <sch:expediente>EX-2022-08395999-   -APN-DGPFE#MS</sch:expediente>
         <sch:beneficiario>
            <sch:extranjero>false</sch:extranjero>
            <sch:tipoIdentificacion>CUI</sch:tipoIdentificacion>
                 <sch:codigoIdentificacion>20999999991
                 </sch:codigoIdentificacion>
         </sch:beneficiario>
      </sch:ConsultarAnulacionDeContratoIpexRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 4.16. Ejemplo de Request SOAP para el Web Service Consulta de Contrato Anulado

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sch="http://dgsiaf.mecon.gob.ar/ipex/schemas">
   <soapenv:Header />
   <soapenv:Body>
      <sch:ConsultarAnulacionDeContratoIpexResponse>
         <sch:anulable>true</sch:anulable>
      </sch:ConsultarAnulacionDeContratoIpexResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 4.17. Ejemplo de Response SOAP para el Web Service Consulta de Contrato Anulado

Parámetro de Entrada	Tipo de Dato	Descripción	Ejemplo

expediente	String	Código del expediente a consultar	EX-2022-08395999--APN-DGPFE#MS
extranjero	Boolean	Indica si el beneficiario del contrato es local o extranjero	False
tipoIdentificacion	Enumerativo	Identifica el tipo de beneficiario en el sistema. Valores posibles: - CUI (Código Único de Identificación Tributaria) - PAS (Pasaporte)	CUI
codigoIdentificacion	String	Número de CUIT o pasaporte del beneficiario	20999999991

Tabla 4.3. Parámetros de Entrada para el Web Service Consulta de Contrato Anulado

Parámetro de Salida	Tipo de Dato	Descripción	Ejemplo
anulable	Boolean	Indica si el contrato se encuentra anulado en el sistema	True

Tabla 4.4. Parámetros de Salida para el Web Service Consulta de Contrato Anulado

4.10.2 Implementando la Publicación del Web Service

4.10.2.1 Implementación de un Endpoint Público en IPEX

Como primer paso para la solución se implementó una clase endpoint en la aplicación IPEX denominada **ConsultarAnulacionDeContratoEndpoint**. En la figura 4.18 se presenta el código java para esta implementación. Este endpoint es el encargado de recibir los requests externos, disparar el procesamiento de la consulta y retornar el response correspondiente al

sistema LOyS. La implementación de esta clase se encuentra anotada con **@Endpoint** para indicar que es una clase que expone web services.

El método **handleConsultarAnulacionContrato()** delega la implementación de la consulta en un servicio interno que se encuentra inyectado en la variable **contratoService**. Si este servicio retorna un mensaje de error, se genera una excepción para informar al sistema externo el error recibido. Si por el contrario, no se recibe ningún mensaje de error, se retorna como respuesta un valor booleano indicando que el contrato se encuentra anulado. La implementación del método incluye además las siguientes annotations: **@Action** para indicar el action del mensaje SOAP que disparará la ejecución del método, **@RequestPayload** para convertir la entrada XML en el tipo de entrada y **@ResponsePayload** para convertir el tipo de salida a XML.

```
package ar.gob.mecon.dgsiaf.ipex.ws.consultarAnulacionDeContrato;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;
import org.springframework.ws.soap.addressing.server.annotation.Action;
import ar.gob.mecon.dgsiaf.ipex.contrato.service.ContratoService;
import ar.gob.mecon.dgsiaf.ipex.ws.config.xsd.jaxb.ConsultarAnulacionDe
ContratoIpexRequest;
import ar.gob.mecon.dgsiaf.ipex.ws.config.xsd.jaxb.ConsultarAnulacionDe
ContratoIpexResponse;

@Endpoint
public class ConsultarAnulacionDeContratoEndpoint {

  private final Logger LOGGER = LoggerFactory.getLogger(this.getClass());

  @Autowired
  private ContratoService contratoService;
```

```
  public static final String ACTION =
  "ar.gob.mecon.dgsiaf.ipex.ws.ConsultaAnulacionDeContrato";

  @Action(ACTION)
  @ResponsePayload
  public ConsultarAnulacionDeContratoIpexResponse
   handleConsultarAnulacionContrato(
    @RequestPayload ConsultarAnulacionDeContratoIpexRequest
    consultarAnulacionRequest) throws Exception {
      String mensaje = contratoService.consultarAnulacionDeContrato
        (consultarAnulacionRequest.getExpediente(),
         consultarAnulacionRequest.getBeneficiario());
      Boolean isAnulable = mensaje.isEmpty();
      ConsultarAnulacionDeContratoIpexResponse response = new
        ConsultarAnulacionDeContratoIpexResponse();
      response.setAnulable(isAnulable);
      if (!isAnulable) {
      throw new SoapFaultIpexException(mensaje);
      }
      return response;
  }

}
```

Figura 4.18. Implementación del Endpoint Público para la Consulta de Contrato Anulado en IPEX

Luego de la implementación del endpoint se agregó la configuración requerida para poder exponer el servicio al sistema LOyS. Dicha configuración se encuentra implementada en la clase **IpexWsConfiguration** la cual se presenta en la figura 4.19. En la definición de esta clase se incluyen dos annotations: **@EnableWs** utilizada para habilitar las características necesarias para el manejo de Web Services dentro de la aplicación y **@Configuration** utilizada para indicarle a Spring que la clase contiene beans de configuración. La configuración además extiende de la clase **WsConfigurerAdapter** que es una clase que Spring provee para facilitar la configuración de Web Services.

Para el servicio de Consulta de Contrato Anulado se definieron tres beans: **ServlerRegistrationBean** para registrar el servlet encargado de atender y despachar las

peticiones SOAP en las URLs indicadas, **SimpleWsdl11Definition** para publicar el WSDL del servicio en internet y **SimpleXsdSchema** para exponer el esquema de tipos XSD.

```
package ar.gob.mecon.dgsiaf.ipex.context;

import org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.config.annotation.WsConfigurerAdapter;
import org.springframework.ws.transport.http.MessageDispatcherServlet;
import org.springframework.ws.wsdl.wsdl11.SimpleWsdl11Definition;
import org.springframework.xml.xsd.SimpleXsdSchema;


@EnableWs
@Configuration
public class IpexWsConfiguration extends WsConfigurerAdapter {

    @Bean
    public ServletRegistrationBean
     messageDispatcherServlet(ApplicationContext applicationContext) {
       MessageDispatcherServlet servlet = new MessageDispatcherServlet();
       servlet.setApplicationContext(applicationContext);
       ServletRegistrationBean srb = new ServletRegistrationBean(servlet,
        "/ws/*", "/ws-private/*");

        return srb;
    }

    @Bean
    public SimpleWsdl11Definition consultarAnulacionDeContratoWsdl() {
       return new SimpleWsdl11Definition(new ClassPathResource(
        "consultarAnulacionDeContrato.wsdl"));
    }

    @Bean
    public SimpleXsdSchema ipexTypesXsd() {
       return new SimpleXsdSchema(new ClassPathResource("ipexTypes.xsd"));
    }
```

```
    /**
     * Otros beans de configuración
     */

}
```

Figura 4.19. Configuración del Web Service para la Consulta de Contrato Anulado en IPEX

4.10.2.2 Implementación de un Servicio en IPEX

El siguiente paso consistió en la implementación del servicio para la consulta de contrato anulado. Para ello se definieron dos clases: una interface **ContratoService** donde se incluye la definición del método **consultarAnulacionDeContrato()** y una clase **ContratoServiceImpl** donde se incluye la implementación para ese método. Las figuras 4.20 y 4.21 muestran las implementaciones para estas dos clases.

La clase ContratoServiceImpl se encuentra anotada con **@Service** para indicar que se trata de un componente de la capa de servicios. La implementación del método consultarAnulacionDeContrato() delega la consulta en un objeto que contiene la lógica de acceso a datos y que se inyecta a través de **@Autowired** en la variable **storeProcedureRepository**. Cuando finaliza la ejecución de este método se retorna un mensaje con la respuesta que se recibe desde el repositorio.

```
package ar.gob.mecon.dgsiaf.ipex.contrato.service;

import ar.gob.mecon.dgsiaf.ipex.ws.config.xsd.jaxb.Beneficiario;
import ar.gob.mecon.dgsiaf.ipex.ws.pnud.jaxb.ConsultarContrato;
import
ar.gob.mecon.dgsiaf.ipex.ws.pnud.jaxb.ConsultarContratoResponseWrapper;

public interface ContratoService {
```

```
  String consultarAnulacionDeContrato(String expediente, Beneficiario
   beneficiario);

  /**
   * Otros métodos de la interfaz
   */

}
```

Figura 4.20. Implementación de la Interfaz para el Servicio de Consulta de Contrato Anulado en IPEX

```
package ar.gob.mecon.dgsiaf.ipex.contrato.service.impl;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import ar.gob.mecon.dgsiaf.ipex.contrato.service.ContratoService;
import ar.gob.mecon.dgsiaf.ipex.repository.IpexStoreProcedureRepository;
import ar.gob.mecon.dgsiaf.ipex.ws.config.xsd.jaxb.Beneficiario;
import ar.gob.mecon.dgsiaf.ipex.ws.contrato.ContratoWSClient;
import ar.gob.mecon.dgsiaf.ipex.ws.pnud.jaxb.ConsultarContrato;
import ar.gob.mecon.dgsiaf.ipex.ws.pnud.jaxb.ConsultarContratoRequest;
import
ar.gob.mecon.dgsiaf.ipex.ws.pnud.jaxb.ConsultarContratoResponseWrapper;

@Service
public class ContratoServiceImpl implements ContratoService {

  private static final Logger LOGGER =
   LoggerFactory.getLogger(ContratoServiceImpl.class);

  @Autowired
  @Qualifier("ipexStoreProcedureRepository")
  private IpexStoreProcedureRepository storeProcedureRepository;

  @Override
  public String consultarAnulacionDeContrato(String expediente,
Beneficiario
```

```
   beneficiario) {
      String mensaje = "";
      mensaje = storeProcedureRepository.consultaBajaContrato(expediente,
      beneficiario.getTipoIdentificacion().value(),
        beneficiario.getCodigoIdentificacion());

      return mensaje;
  }

  /**
   * Otros métodos del servicio
   */

}
```

Figura 4.21. Implementación del Servicio para la Consulta de Contrato Anulado en IPEX

4.10.2.3 Implementación de un Repositorio en IPEX

Una vez completada la implementación del servicio se trabajó en el desarrollo de la capa de acceso a datos. Para ello, al igual que en la implementación del servicio, se definieron dos clases: una interface **IpexStoreProcedureRepository** donde se indica la definición del método **consultaBajaContrato()** y una clase **IpexStoreProcedureRepositoryImpl** donde se realiza la implementación específica para ese método. Las figuras 4.22 y 4.23 exponen la implementación para estas dos clases.

La clase IpexStoreProcedureRepositoryImpl se encuentra anotada con **@Repository** para que el container de Spring la gestione como un componente de acceso a datos.

El método consultaBajaContrato() es el encargado de invocar al stored procedure que ejecuta la consulta en la base de datos UEPEX. Para invocar a este procedimiento se utiliza un **EntityManager** que es una clase que provee JPA para interactuar con una base de datos física, en este caso, la base de datos UEPEX ACCESOS. La annotation **@PersistentContext** inyecta una instancia de esta clase en la variable **entityManager**. A través del EntityManager

se setean los parámetros de entrada, se dispara la ejecución del procedimiento y se recupera el parámetro de salida con el mensaje de respuesta.

```
package ar.gob.mecon.dgsiaf.ipex.repository;

public interface IpexStoreProcedureRepository {

  String consultaBajaContrato(String expediente, String tipoIdentificacion,
   String codigoIdentificacion);

  /**
   * Otros métodos de la interfaz
   */

}
```

Figura 4.22. Implementación de la Interfaz para el Repository de Acceso a Datos en IPEX

```
package ar.gob.mecon.dgsiaf.ipex.repository.impl;

import javax.persistence.EntityManager;
import javax.persistence.ParameterMode;
import javax.persistence.PersistenceContext;
import javax.persistence.StoredProcedureQuery;
import org.springframework.stereotype.Repository;
import ar.gob.mecon.dgsiaf.ipex.repository.IpexStoreProcedureRepository;

@Repository("ipexStoreProcedureRepository")
public class IpexStoreProcedureRepositoryImpl implements
IpexStoreProcedureRepository{

  private static final String SP_CONSULTA_BAJA_CONTRATO_UEPEX =
   "spConsultaBajaContratoUepex";

  @PersistenceContext
  private EntityManager entityManager;

  public String consultaBajaContrato(String expediente, String
   tipoIdentificacion, String codigoIdentificacion) {
      String errorMsg;
```

```
        StoredProcedureQuery storedProcedure = entityManager
         .createStoredProcedureQuery(SP_CONSULTA_BAJA_CONTRATO_UEPEX);
        storedProcedure.registerStoredProcedureParameter("ee", String.class,
          ParameterMode.IN);
        storedProcedure.setParameter("ee", expediente);
        storedProcedure.registerStoredProcedureParameter(
         "tipoIdentificacion", String.class, ParameterMode.IN);
        storedProcedure.setParameter("tipoIdentificacion",
         tipoIdentificacion);
        storedProcedure.registerStoredProcedureParameter(
         "codigoIdentificacion", String.class, ParameterMode.IN);
        storedProcedure.setParameter("codigoIdentificacion",
         codigoIdentificacion);
        storedProcedure.registerStoredProcedureParameter("errorMsg",
         String.class, ParameterMode.OUT);
        storedProcedure.execute();
        errorMsg = (String)
         storedProcedure.getOutputParameterValue("errorMsg");

        return errorMsg;
    }
    /**
     * Otros métodos del Repositorio
     */

}
```

Figura 4.23. Implementación del Repository de Acceso a Datos en IPEX

4.10.2.4 Implementación de un Stored Procedure en la Base UEPEX ACCESOS

El siguiente paso fue el desarrollo de un stored procedure en la base de datos UEPEX ACCESOS para determinar si un contrato en particular se encuentra dado de baja. El procedimiento **spConsultaBajaContratoUepex** que se muestra en la figura 4.24 implementa dicha funcionalidad. Este procedimiento recibe como parámetros de entrada un expediente electrónico, un tipo y un identificador de beneficiario. Con estos parámetros, el procedimiento consulta la información centralizada en esta base de datos para obtener el ID del contrato vinculado al expediente y el nombre de la base UEPEX donde fue gestionado dicho contrato. Si

la consulta no arroja ningún resultado, el procedimiento retorna un mensaje de error informando que no existen datos para la información consultada. Por el contrario, si existe un ID de contrato y una base con información del mismo, el procedimiento delega la consulta en la base de datos correspondiente.

```
USE [UEPEX_ACCESOS]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
/*
-- Consulta si un contrato uepex está dado de baja o no.
-- Retorna @errorMsg vacío en caso de que pueda darse de baja, o el mensaje
-- de error que corresponda
*/
ALTER PROCEDURE [dbo].[spConsultaBajaContratoUepex]
      @ee varchar(100),
      @tipoIdentificacion varchar(5),
      @codigoIdentificacion varchar(20),
      @errorMsg varchar(200) output
AS
   declare @idContrato int
   declare @bdUepex varchar(50)
   declare @contratoDesafectado bit
   declare @strSql varchar(max)

   set @errorMsg = ''

-- Se consulta en la tabla centralizada si existe una base uepex que
-- contenga información del contrato para el expediente y beneficiario
-- recibidos por parámetro.
select @bdUepex = ugde.DBuepex, @idContrato = ugde.idContrato
from UEPEX_GDE ugde
where ugde.expediente = @ee and ugde.tipoIdentificacion =
@tipoIdentificacion and ugde.codigoIdentificacion = @codigoIdentificacion

-- Si la consulta no obtuvo resultados, se informa el error
IF (@@ROWCOUNT = 0)
BEGIN
    set @errorMsg = 'No existe contrato para el expediente: ' + @ee + ' y
```

```
beneficiario: ' + @tipoIdentificacion + '-' + @codigoIdentificacion + ' en
el sistema UEPEX'
    return
END

-- Si la consulta obtuvo resultados, se invoca al stored procedure que
-- consulta la baja del contrato en la uepex correspondiente
set @strSql = @bdUepex + '.dbo.ipex_ConsultaBajaContratoUepex '
exec @strSql @idContrato, @contratoDesafectado out

-- Si el contrato se encuentra dado de baja (desafectado) se informa
-- la respuesta al usuario
IF (@contratoDesafectado = 0)
    set @errorMsg = 'El contrato para el expediente: ' + @ee + ' y
beneficiario: ' + @tipoIdentificacion + '-' + @codigoIdentificacion + ' no
está desafectado en UEPEX'
```

Figura 4.24. Implementación del Stored Procedure para la Consulta de Contrato Anulado en UEPEX ACCESOS

4.10.2.5 Creación de un Stored Procedure en las UEPEX locales

Como último paso para la solución se implementó un stored procedure en cada base de datos UEPEX local. La implementación de este stored procedure se muestra en la figura 4.25. El procedimiento **ipex_ConsultaBajaContratoUepex** recibe como parámetro de entrada un ID de contrato y retorna como parámetro de salida un valor booleano indicando si dicho contrato se encuentra dado de baja.

```
USE [uepex_UE051]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO
```

```
ALTER PROCEDURE [dbo].[ipex_ConsultaBajaContratoUepex](
    @idContrato int,
    @contratoDesafectado bit out)
AS
BEGIN
   -- Consulta si el idContrato recibido por parámetro se encuentra
   -- desafectado
   SELECT *
   FROM   contratos con
   WHERE  con.idCabecera = @idContrato
          AND con.importeRestante = 0
          AND con.estado = 'Desafectado'

   IF (@@ROWCOUNT = 0)
      SET @contratoDesafectado = 0
   ELSE
      SET @contratoDesafectado = 1

END
```

Figura 4.25. Implementación del Stored Procedure para la Consulta de Contrato Anulado en una UEPEX local

4.11 Resultados Obtenidos

La modernización del sistema UEPEX mediante la inclusión de una capa de servicios web permitió llevar a cabo la adaptación de este sistema para poder conectarlo a Internet.

La solución adoptada resultó menos costosa en cuanto a tiempo y esfuerzo de implementación en comparación a otras alternativas como la migración a SOA o el reemplazo del sistema actual por uno más moderno. (Ver tabla 1 de comparación de estrategias de modernización en SeetharamaTantry et al., 2017).

Por otra parte, el hecho de haber aislado la funcionalidad requerida en una nueva aplicación web permitió desacoplar del sistema UEPEX las tareas de despliegue y

mantenimiento de esta nueva aplicación. Con esta solución, el sistema UEPEX se mantiene desplegado en la vieja infraestructura legacy mientras que la aplicación IPEX se ejecuta sobre un cluster de contenedores. Los principales beneficios de esta plataforma son el despliegue continuo, la escalabilidad del sistema, el balanceo de carga, la optimización de recursos y el monitoreo de la aplicación.

Por último, la implementación realizada permitió adaptar el sistema UEPEX en el corto plazo para alinearlo al nuevo requerimiento de interoperabilidad, pero no resolvió los problemas de infraestructura y mantenimiento de este sistema. Para resolver esos problemas será necesario aplicar una estrategia de reemplazo o modernización del sistema completo en el mediano o largo plazo.

Capítulo 5. Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones de esta tesina y se plantean algunas líneas de investigación para trabajos futuros.

5.1 Conclusiones

Las organizaciones necesitan gestionar sus sistemas legacy para poder adaptarlos a los nuevos requerimientos funcionales y tecnológicos. Para ello, existen diversas estrategias que abarcan desde la modernización de una interfaz gráfica hasta el reemplazo completo de todo un sistema.

Una estrategia bastante útil es el wrapping funcional, el cual permite “envolver” funcionalidades legacy con una capa funcional mucho más moderna. Esta estrategia permite exponer hacia otros clientes funcionalidades o datos del sistema legacy, y al mismo tiempo, retener la inversión realizada en estos sistemas y extender su vida útil.

La solución expuesta en este trabajo muestra que la combinación de wrapping funcional y la tecnología de web services, puede ser una estrategia apropiada para integrar un sistema legacy con otras aplicaciones web, evitando los costos de un reemplazo o reimplementación del sistema completo.

5.2 Trabajos Futuros

A continuación se dejan planteadas algunas líneas de investigación para futuros trabajos:

- La implementación del soapInvoker que se expuso en esta tesina funciona únicamente en una base de datos SQL Server. Se propone evaluar otras

soluciones para poder invocar servicios web desde otros motores de bases de datos distintos a SQL Server.

- REST propone una tecnología mucho más rápida y flexible en comparación al protocolo SOAP. Como consecuencia de esto algunos organismos de la Administración Pública Nacional están utilizando REST para implementar sus servicios web. Se plantea el estudio de soluciones que permitan adaptar o migrar los servicios basados en SOAP a la tecnología REST.

Referencias Bibliográficas

Bach, H. M. (2006). Integración de Sistemas Heredados Utilizando Web Services. *Universidad Ricardo Palma*.

Charette, R. (2020). *Inside the Hidden World of Legacy IT Systems*. IEEE Spectrum For The Technology Insider. https://spectrum.ieee.org/inside-hidden-world-legacy-it-systems

Comella-Dorda, S., Wallnau, K., Seacord, R. C., & Robert, J. (2000). A Survey of Legacy System Modernization Approaches.

Delgado, A., González, L., & Piedrabuena, F. (2006). *Desarrollo de aplicaciones con enfoque SOA*.

DGSIAF. Argentina.gob.ar. https://www.argentina.gob.ar/economia/sechacienda/dgsiaf

Díaz Amado, J. F. (2014). Distribución de un Entorno de Modelado Utilizando Servicios Web. *Universidad Politécnica de Cartagena*.

Diferencias entre REST y SOAP. Red Hat. https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest#rest

The Equifax Data Breach. (2018). https://republicans-oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf

Fahmideh, M., Daneshgar, F., Beydoun, G., & Rabhi, F. (2017). Challenges in migrating legacy software systems to the cloud — an empirical study. *Information Systems*.

Frutos, M., Ferreira Szpiniak, A., & Zorzán, F. A. *Integración entre aplicaciones internas y externas de la UNRC utilizando Web Services*.

GDE - Sistema de Gestión Documental Electrónica. Argentina.gob.ar. https://www.argentina.gob.ar/jefatura/innovacion-publica/innovacion-administrativa/gde-sistema-de-gestion-documental-electronica

Introducción a JSON (JavaScript Object Notation). JSON. https://www.json.org/json-es.html

Jha, S., Jha, M., O'Brien, L., & Wells, M. (2015). Integrating Legacy System into Big Data Solutions: Time to make the Change.

Nesbit, J. (2020). *Are Legacy Systems Holding Big Banks Back?* BeSmartee. https://www.besmartee.com/grav/are-legacy-systems-holding-big-banks-back

Patel, B. (2017). Legacy systems are problems for boardrooms not computer geeks. *Financial Times*. https://www.ft.com/content/5bf9de84-d665-11e6-944b-e7eb37a6aa8e

¿Qué es una API? (2017). Red Hat. https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

SeetharamaTantry, H., Murulidhar, N., & Chandrasekaran, K. (2017). Implications of Legacy Software System Modernization - A Survey in a Changed Scenario. *International Journal of Advanced Research in Computer Science*.

Sneed, H. (2000). Encapsulation of legacy software: A technique for reusing legacy software components. *Annals of Software Engineering*.

UEPEX. Argentina.gob.ar. https://www.argentina.gob.ar/economia/sechacienda/dgsiaf/uepex

Visaggio, G. (2001). Ageing of a data-intensive legacy system: symptoms and remedies. *Journal of Software Maintenance and Evolution: Research and Practice*.

Web Services Architecture. (2004). W3C. https://www.w3.org/TR/ws-arch/

Wu, B., Lawless, D., Bisbal, J., Grimson, J., Wade, V., O'Sullivan, D., & Richardson, R. (1997). Legacy Systems Migration - A Method and its Tool-kit Framework.

XML Soap. W3Schools. https://www.w3schools.com/xml/xml_soap.asp

Yang, L., & Hongji, Y. (2001). Simplicity: a key engineering concept for program understanding. *Proceedings 9th International Workshop on Program Comprehension*.