Monografía | Monograph

Despliegue de un bus de Servicios Empresariales para la Universidad Agraria de La Habana

Deployment of a bus of Business Services for the Agrarian University of Havana

Ivett Sosa Franco[1], Dra. C Neili Machado García[2]

1-Ingeniera Informática, Profesora Departamento de Informática, Facultad de Ciencias Técnicas

2-Doctora en Ciencias, Directora de Informatización

Universidad Agraria de la Habana: Fructuoso Rodríguez Pérez, Cuba, Carretera Tapaste, Km 23 ½ Autopista Nacional. San José de las Lajas, Mayabeque, Cuba.

*Autor para correspondencia: ivett@unah.edu.cu, neili@unah.edu.cu

Resumen

El uso de servicios web en los sistemas actuales se ha vuelto de vital importancia en los softwares por las posibilidades que ofrece. Sistemas accesibles por otros sistemas, exportación de servicios, modularidad, entre otras son varias de las ventajas que brinda. Cuba actualmente está apostando por tecnologías con arquitecturas orientadas a servicios, basadas en estándares abiertos. La Universidad Agraria de La Habana "Fructuoso Rodríguez Pérez" cuenta con varios sistemas web creados con la tecnología de servicios web, pero no contaba con un sistema que los administrara. Asimismo, los estudiantes realizaban proyectos informáticos (desarrollo de software) como parte de las asignaturas curriculares, que en ocasiones incluía el consumo y creación de servicios web. Esto traía consigo el problema del consumo de servicios punto a punto, que a su vez devenía en un aumento exponencial en la complejidad de los sistemas, dificultad de mantenimiento, así como, puntos únicos de falla. Los servicios web (API REST en su totalidad) se encuentran desarrollados fundamentalmente en Java (utilizando Spring Boot) aunque algunos utilizan Python (utilizando Django) o Nodejs (utilizando Express). Los Buses de Servicios Empresariales (ESB) son sistemas desarrollados con el fin de administrar y centralizar los servicios web, eliminando así los llamados sistemas espaguetis. El presente trabajo tuvo como objetivo desplegar un ESB para la comunicación entre aplicaciones con arquitecturas orientadas a servicios. Durante el proceso de investigación fue utilizado el método teórico analítico, que está asociado al análisis de los datos. Se realizó un análisis de la problemática en búsqueda de una solución eficiente, lo que trajo consigo la revisión de las herramientas de tipo ESB del mercado.

Palabras clave: Bus de Servicios Empresariales, servicio web.

Summary

The use of web services in current systems has become vitally important in software due to the possibilities it offers. Systems accessible by other systems, export of services, modularity, among others, are several of the advantages it offers. Cuba is currently betting on technologies with service-oriented architectures, based on open standards. The Agrarian University of Havana "Fructuoso Rodríguez Pérez" has several web systems created with web services technology, but it did not have a system to manage them. Likewise, the students carried out computer projects (software development) as part of the curricular subjects, which sometimes included the consumption and creation of web services. This brought with it the problem of the consumption of point-to-point services, which in turn resulted in an exponential increase in the complexity of the systems, difficulty of maintenance, as well as single points of failure.Web services (REST API in its entirety) are developed mainly in Java (using Spring Boot) although some use Python (using Django) or Nodejs (using Express).Business Service Buses (ESB) are systems developed in order to manage and centralize web services, thus eliminating the so-called spaghetti systems.The present work aimed to deploy an ESB for communication between applications with service-oriented architectures. During the research process, the analytical theoretical method was used, which is associated with data analysis. An analysis of the problem was carried out in search of an efficient solution, which led to the revision of the ESB-type tools on the market.

Keywords: Business Service Bus, web service.

Recibido: 12 de noviembre de 2020
Aprobado: 27 de noviembre de 2020

Introducción

La evolución de las tecnologías de la información incorporó para el mundo corporativo de servicios los Sistemas de Información Compuestos (SIC) que logran integrar fácil, rápida y eficazmente diversas bases de datos incorporando el concepto de *brokers* (intermediario entre operaciones) y módulos de servicios de información (Almazán *et al*., 2017).

Los módulos de información circulan a través de una plataforma que los hace compatibles entre sí, automatizando los procesos y minimizando los tiempos de comunicación - ejecución, a esta herramienta se le conoce como Bus de Servicios Empresariales (ESB del inglés Enterprise Service Bus) que permite publicar los datos de información y ponerlos en circulación garantizando gobernabilidad, reutilización, optimización y continuidad en todas las interfaces posibles (Ahuja et al., 2011).

Empresas famosas y mundialmente conocidas como *AIRBUS, Oasics, tic:toc, CISCO, verizon, Unilever, Square* (MuleSoft, 2019), *Motorola, eBay, QANTAS, ZeOmega* (WSO2, 2019), entre otras utilizan

Introduction

The evolution of information technologies incorporated Composite Information Systems (CIS) for the corporate world of services, which manage to easily, quickly and efficiently integrate various databases incorporating the concept of brokers (intermediary between operations) and modules of services of information (Almazán et al., 2017).

The information modules circulate through a platform that makes them compatible with each other, automating processes and minimizing communication times - execution, this tool is known as Business Service Bus (ESB) that allows publish the information data and put it into circulation guaranteeing governance, reuse, optimization and continuity in all possible interfaces (Ahuja et al., 2011).

Famous and world-renowned companies such as AIRBUS, Oasics, tic: toc, CISCO, verizon, Unilever, Square (MuleSoft, 2019), Motorola, eBay, QANTAS, ZeOmega (WSO2, 2019), among others use ESB

implementaciones de ESB para asegurar, mejorar y agilizar la eficiencia de sus servicios.

Cuba actualmente está apostando por tecnologías con Arquitecturas Orientada a Servicios (SOA, del inglés Service Oriented Architecture), basadas en estándares abiertos. Para esto necesita soluciones de integración potentes como los ESB. A pesar de que las investigaciones en estos términos aún son escasas, centros universitarios de gran relevancia en el país como la UCI (Universidad de las Ciencias Informáticas) y la CUJAE (Instituto Superior Politécnico José Antonio Echeverría) investigan al respecto con artículos publicados y conocidos a nivel mundial como "Implementación de lenguajes de contrato electrónico en Oracle Service Bus" por la Revista Cubana de Ciencias Informáticas (RCCi) y "Capacidad de Orquestación de Servicios Web en las Herramientas MULE ESB y Oracle Service Bus" por la CUJAE.

Además, en el Complejo de Investigaciones de Tecnologías Integradas (CITI)[1] se está desarrollando un proyecto de investigación llamado Integración de Información basada en la Descripción Semántica de Fuentes de Datos Heterogéneas (SEMANTINFO), donde se utiliza la tecnología ESB para integrar aplicaciones utilizadas en el ámbito de la CUJAE. Específicamente, se integraron los sistemas SIGENU, Copérnico y Directorio CUJAE, mediante la composición de varios de los servicios web (Flores, 2015).

La Universidad Agraria de la Habana "Fructuoso Rodríguez Pérez" no queda excluida de la informatización ni del avance tecnológico. Actualmente cuenta con varios sistemas desplegados y en desarrollo utilizando arquitecturas orientadas a servicios. Sistemas como el SIGENU, implementado por la CUJAE hace varios años cuenta con una serie de servicios API REST que son consumidos por otras aplicaciones desarrolladas en la propia UNAH.

El aumento de sistemas con estas características trae consigo el problema del consumo de servicios punto a punto y esto a su vez causa un aumento exponencial de la complejidad en los sistemas, dificultad de mantenimiento, así como, puntos únicos de falla. La arquitectura resultante provoca estragos en la confiabilidad, pérdida de agilidad y falta de productividad.

implementations to secure, improve and streamline the efficiency of your services.

Cuba is currently betting on technologies with Service Oriented Architecture (SOA, Service Oriented Architecture), based on open standards. For this you need powerful integration solutions like ESBs. Despite the fact that research in these terms is still scarce, university centers of great relevance in the country such as the UCI (University of Computer Sciences) and the CUJAE (José Antonio Echeverría Higher Polytechnic Institute) investigate in this regard with published and well-known articles worldwide as "Implementation of electronic contract languages in Oracle Service Bus" by the Cuban Journal of Informatics Sciences (RCCi) and "Capacity for Orchestration of Web Services in MULE ESB and Oracle Service Bus Tools" by CUJAE.

In addition, at the Integrated Technologies Research Complex (CITI) a research project called Integration of Information based on the Semantic Description of Heterogeneous Data Sources (SEMANTINFO) is being developed, where ESB technology is used to integrate applications used in the scope of the CUJAE. Specifically, the SIGENU, Copérnico and CUJAE Directory systems were integrated, through the composition of several of the web services (Flores, 2015).

The "Fructuoso Rodríguez Pérez" Agrarian University of Havana is not excluded from computerization or technological advancement. It currently has several systems deployed and under development using service-oriented architectures. Systems such as SIGENU, implemented by CUJAE several years ago, have a series of API REST services that are consumed by other applications developed in UNAH itself.

The increase in systems with these characteristics brings with it the problem of consumption of point-to-point services and this in turn causes an exponential increase in the complexity of the systems, difficulty of maintenance, as well as single points of failure. The resulting architecture wreaks havoc on reliability, loss of agility, and a lack of productivity.

[1] Grupo de investigación formado por el MININT, la CUJAE y otras instituciones (con sede en la CUJAE). Desarrolla tecnologías integradas de un amplio espectro de las ciencias técnicas, vinculando las necesidades de superación científica de especialistas con soluciones concretas y ágiles. Ejecuta proyectos por grupos de trabajo flexibles.

Desarrollo

Fundamentos Teóricos
Arquitectura Orientada a Servicios (SOA)

La Arquitectura Orientada a Servicios (del inglés Service Oriented Architecture, SOA) es una arquitectura que define que las aplicaciones brindan su funcionalidad empresarial en forma de servicios reutilizables. Los servicios son utilizados por otras aplicaciones que también podrían ser implementaciones de servicios. Con este enfoque, se implementan procesos comerciales complejos mediante la combinación de varios servicios. Esto se llama orquestación de servicios (Menge, 2007). La Figura 1 La típica Arquitectura Orientada a Serviciosmuestra el típico escenario de una aplicación con SOA. Los proveedores de servicios registran sus servicios en un servicio central de nombres. Una aplicación de consumidor puede usar este servicio de nombres para descubrir los servicios disponibles y recuperar información sobre cómo conectarse a un proveedor de servicios en particular. Luego, la aplicación del consumidor puede obtener una descripción del servicio que define cómo se puede utilizar el servicio y finalmente consultar el servicio.

Development

Theoretical fundament
Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is an architecture that defines that applications provide their business functionality in the form of reusable services. The services are used by other applications that could also be service implementations. With this approach, complex business processes are implemented by combining multiple services. This is called service orchestration (Menge, 2007). Figure 1 shows a typical application scenario with SOA. Service providers register their services with a central name service. A consumer application can use this naming service to discover available services and retrieve information on how to connect to a particular service provider. Then the consumer application can get a description of the service that defines how the service can be used and finally query the service.

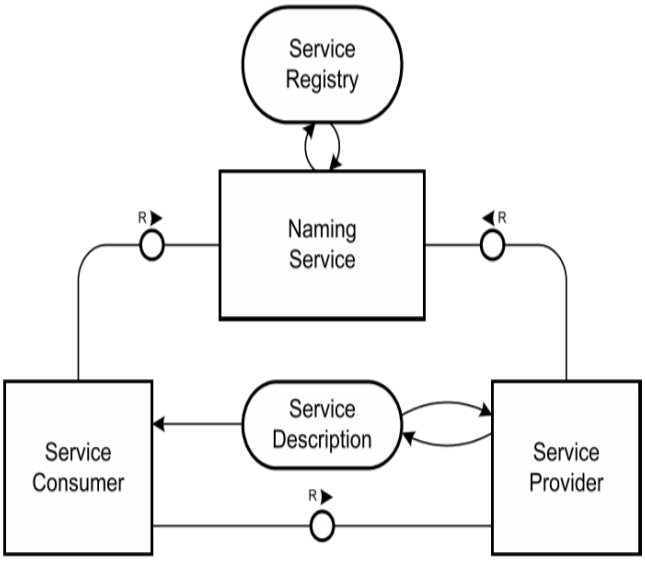

Figura 1 *La típica Arquitectura Orientada a Servicios*

Figure 1 *The typical Service Oriented Architecture*

SOA puede implementarse utilizando cualquier tecnología basada en servicios. Por lo general, se utilizan tecnologías de servicios web como SOAP o REST. SOA permite que las aplicaciones empresariales

SOA can be implemented using any service-based technology. Generally, web services technologies such as SOAP or REST are used. SOA enables complex business applications to be comprised of

complejas se compongan de estos servicios, incluso cuando los proveedores de esos servicios son aplicaciones alojadas en plataformas de sistemas operativos dispares, escritas en diferentes lenguajes de programación o basadas en modelos de datos separados. Esta composición flexible vincula los sistemas empresariales en toda la empresa y extienden los servicios empresariales a los clientes y socios comerciales. La adopción de SOA en aplicaciones críticas para los negocios se está produciendo solo de forma incremental. Refactorizar, envolver o reemplazar aplicaciones heredadas con nuevos equivalentes que cumplan con los estándares será un proceso lento. Eso implica que una infraestructura de integración no puede basarse únicamente en servicios (Menge, 2007).

SOA surge con la necesidad de construir Servicios de Información Compuestos (SIC), que permitan la integración de bases de datos heterogéneas en un sistema útil e inteligente orientado a la optimización de la comunicación integral de los negocios. SOA proporciona una estructura de información eficiente y versátil capaz de consultar en diversas fuentes de datos, lo que le otorga la capacidad de evolucionar, adaptarse a una particular demanda de servicio de una corporación (Maréchaux, 2006).

SOA le garantiza a las organizaciones estar muy bien actualizadas, ser eficaces y eficientes en cuanto a la prestación de servicios digitales, incorporándolo de una manera óptima en el competitivo mundo de Negocios Inteligentes[2].

Servicios Web

Un Servicio Web es una tecnología empresarial autónoma y sin estado a la que se puede acceder a través de una interfaz estandarizada y neutral en la implementación, utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones (Newcomer and Lomow, 2005).

Los servicios web permiten la interoperación de sistemas distribuidos heterogéneos con independencia de las plataformas hardware y software empleadas. Puede pensarse en ellos como en una arquitectura, conceptual y tecnológica, que hace posible que distintos servicios se describan, publiquen, descubran y utilicen a través de sistemas distribuidos, empleando la

these services, even when the providers of those services are applications hosted on disparate operating system platforms, written in different programming languages, or based on separate data models. This flexible composition links business systems across the enterprise and extends business services to customers and business partners. The adoption of SOA in business-critical applications is occurring only incrementally. Refactoring, wrapping, or replacing legacy applications with new, standards-compliant equivalents will be a slow process. This implies that an integration infrastructure cannot be based solely on services (Menge, 2007).

SOA arises with the need to build Composite Information Services (CIS), which allow the integration of heterogeneous databases in a useful and intelligent system aimed at optimizing the integral communication of businesses. SOA provides an efficient and versatile information structure capable of consulting various data sources, which gives it the ability to evolve, adapt to a particular service demand of a corporation (Maréchaux, 2006).

SOA guarantees organizations to be very well updated, effective and efficient in terms of the provision of digital services, incorporating it in an optimal way in the competitive world of Smart Business.

Web services

A Web Service is an autonomous and stateless business technology that can be accessed through a standardized and implementation-neutral interface, uses a set of protocols and standards that serve to exchange data between applications (Newcomer and Lomow, 2005) .

Web services allow the interoperation of heterogeneous distributed systems regardless of the hardware and software platforms used. They can be thought of as an architecture, conceptual and technological, which makes it possible for different services to be described, published, discovered and

[2] Es una herramienta gerencial cuya función es facilitar a las administraciones el cumplimiento de la misión de sus organizaciones, mediante el análisis de la información relativa al negocio y al entorno. Desde el punto de vista del manejo de información, compila, reúne y analiza datos e información, cuyo resultado disemina en la organización. Con ello permite obtener, de modo sistemático y organizado, información relevante sobre el ambiente externo y las condiciones internas de la organización para la toma de decisiones y la orientación estratégica.

infraestructura proporcionada por Internet (Newcomer and Lomow, 2005).

A nivel técnico, los servicios pueden implementarse de varias formas. En este sentido, se pueden distinguir dos tipos de servicios Web: los denominados servicios Web "grandes" ("big" Web Services), más conocidos como servicios Web SOAP, y los servicios Web REST (Alicante, 2014).

Servicios Web SOAP

Los servicios Web SOAP, o servicios Web "big", utilizan mensajes XML para intercomunicarse que siguen el estándar SOAP (del inglés Simple Object Access Protocol), un lenguaje XML que define la arquitectura y formato de los mensajes. Dichos sistemas normalmente contienen una descripción legible por la máquina de la descripción de las operaciones ofrecidas por el servicio, escrita en WSDL (Web Services Description Language), que es un lenguaje basado en XML para definir las interfaces sintácticamente. SOAP proporciona un mecanismo simple y liviano para intercambiar información estructurada y mecanografiada entre pares en un entorno descentralizado y distribuido utilizando XML (Alicante, 2014).

Servicios Web REST

Los servicios Web REST (del inglés Representational State Transfer Web Services) son adecuados para escenarios de integración básicos ad-hoc[3]. Dichos servicios Web se suelen integrar mejor con HTTP (Protocolo de transferencia de hipertexto) que los servicios basado en SOAP, ya que no requieren mensajes XML o definiciones del servicio en forma de fichero WSDL. Estos utilizan estándares muy conocidos como HTTP, SML, URI, MIME, y tienen una infraestructura "ligera" que permite que los servicios se construyan utilizando herramientas de forma mínima. Gracias a ello, el desarrollo de servicios REST es barato y tiene muy pocas "barreras" para su adopción (Alicante, 2014).

Bus de Servicios Empresariales (ESB)

Las organizaciones necesitan soluciones de integración potentes, pero requieren que se basen en estándares abiertos y que admitan SOA. La descentralización de los servicios web en una organización trae consigo problemas de integración punto a punto de los sistemas

used through distributed systems, using the infrastructure provided by the Internet (Newcomer and Lomow, 2005).

At a technical level, services can be implemented in various ways. In this sense, two types of Web services can be distinguished: the so-called "big" Web services, better known as SOAP Web services, and REST Web services (Alicante, 2014).

SOAP Web Services

SOAP Web services, or "big" Web services, use XML messages to intercommunicate that follow the SOAP standard (Simple Object Access Protocol), an XML language that defines the architecture and format of the messages. Such systems usually contain a machine-readable description of the description of the operations offered by the service, written in WSDL (Web Services Description Language), which is an XML-based language for defining interfaces syntactically. SOAP provides a simple and lightweight mechanism to exchange structured and typed information among peers in a decentralized and distributed environment using XML (Alicante, 2014).

REST Web Services

REST (Representational State Transfer Web Services) Web services are suitable for basic ad-hoc integration scenarios. Such Web services are generally better integrated with HTTP (Hypertext Transfer Protocol) than SOAP-based services, as they do not require XML messages or service definitions in the form of a WSDL file. They use well-known standards like HTTP, SML, URI, MIME, and have a "lightweight" infrastructure that allows services to be built using tools in a minimal way. Thanks to this, the development of REST services is cheap and has very few "barriers" for its adoption (Alicante, 2014).

Business Service Bus (ESB)

Organizations need powerful integration solutions, but they require that they be based on open standards and support SOA. The decentralization of web services in an organization brings with it problems of point-to-point integration of systems (systems connected with other systems one by one), as well as exploitable security gaps. A system that orders them

[3] Locución latina que significa literalmente “para esto”. Generalmente se refiere a una solución específicamente laborada para un problema o fin preciso y, por tanto, no generalizable ni utilizable para otros propósitos.

(sistemas conectados con otros sistemas uno a uno), así como brechas explotables de seguridad. Un sistema que los ordene e impida que varios sistemas concurrentes en la misma organización se interconecten unos a otros formando lo conocido como integración espagueti fortalece en gran medida el desarrollo de la misma. Además de que centralizándolos se puede manejar de mejor forma la seguridad. Exactamente esto llevó a la idea de un Bus de Servicios Empresariales (del inglés Enterprise Service Bus, ESB).

Un ESB es fundamentalmente una arquitectura. Es un conjunto de reglas y principios para integrar numerosas aplicaciones en una infraestructura similar a un bus[4]. Los productos ESB permiten a los usuarios construir este tipo de arquitectura, pero varían en la forma en que lo hacen y las capacidades que ofrecen. El concepto central de la arquitectura ESB es que se integran diferentes aplicaciones al colocar un bus de comunicación entre ellas y luego permite que cada aplicación se comunique con el bus. Esto desacopla los sistemas entre sí, lo que les permite comunicarse sin dependencia o conocimiento de otros sistemas en el bus. El concepto de ESB nació de la necesidad de alejarse de la integración punto a punto, que se vuelve frágil y difícil de manejar con el tiempo. La integración punto a punto da como resultado que el código de integración personalizado se distribuya entre las aplicaciones sin una forma central de monitorear o solucionar problemas. Esto a menudo se conoce como "código de espagueti" y no se escala porque crea dependencias estrechas entre las aplicaciones (Maréchaux, 2006).

Los ESB generalmente usan contenedores de servicios distribuidos en un entorno en red. Estos contenedores alojan servicios de integración como enrutadores, transformadores, adaptadores de aplicaciones y les proporcionan una amplia gama de servicios de comunicación. Las aplicaciones se conectan al bus mediante adaptadores de aplicaciones (Menge, 2007).

El ESB debe coordinar las interacciones de los diversos recursos y proporcionar soporte transaccional. Un objetivo general es proporcionar mensajería e integración sin escribir código. Por lo tanto, se proporcionan componentes genéricos que se pueden configurar para realizar un escenario deseado. Un ESB es una columna vertebral ideal para implementar SOA porque proporciona un mecanismo para interconectar todos los servicios necesarios en la solución empresarial compuesta sin comprometer la seguridad, la fiabilidad, el rendimiento y la escalabilidad (Menge, 2007).

and prevents several concurrent systems in the same organization from interconnecting with each other forming what is known as spaghetti integration greatly strengthens its development. In addition to centralizing them, security can be managed in a better way. Exactly this led to the idea of an Enterprise Service Bus (ESB).

An ESB is fundamentally an architecture. It is a set of rules and principles for integrating numerous applications in a bus-like infrastructure. ESB products allow users to build this type of architecture, but they vary in the way they do it and the capabilities they offer. The core concept of the ESB architecture is that different applications are integrated by placing a communication bus between them and then allowing each application to communicate with the bus. This decouples the systems from each other, allowing them to communicate without dependency or knowledge of other systems on the bus. The ESB concept was born out of the need to move away from point-to-point integration, which becomes fragile and difficult to manage over time. Peer-to-peer integration results in custom integration code being distributed across applications without a central way to monitor or troubleshoot. This is often referred to as "spaghetti code" and does not scale because it creates tight dependencies between applications (Maréchaux, 2006).

ESBs generally use distributed service containers in a networked environment. These containers host integration services such as routers, transformers, application adapters, and provide them with a wide range of communication services. Applications are connected to the bus through application adapters (Menge, 2007).

The ESB must coordinate the interactions of the various resources and provide transactional support. An overall goal is to provide messaging and integration without writing code. Therefore, generic components are provided that can be configured to perform a desired scenario. An ESB is an ideal backbone for implementing SOA because it provides a mechanism to interconnect all the necessary services in the composite business solution without compromising security, reliability, performance, and scalability (Menge, 2007).

[4] Modelo de arquitectura de software donde un canal de comunicación compartido facilita las conexiones y la comunicación entre los módulos de software.

¿Por qué usar un ESB?

El aumento de la agilidad organizacional al reducir el tiempo de comercialización de nuevas iniciativas es una de las razones más comunes por las que las empresas implementan un ESB. Una arquitectura ESB facilita esto al proporcionar un sistema simple, bien definido, "conectable" que escala realmente bien. Además, un ESB proporciona una manera de aprovechar sus sistemas existentes y exponerlos a nuevas aplicaciones utilizando sus capacidades de comunicación y transformación (MuleSoft, 2019).

Un ESB proporciona facilidades a las organizaciones. Permite tener reunidos los servicios web de sus aplicaciones en un solo lugar, independientemente de tener varios servidores físicos dedicados. Desarrolladores de Front-end[5] no necesitan diferenciar los servidores físicos donde fueron implementados los servicios para sus aplicaciones (todos estarían aparentemente en el mismo servidor). En casos de migrado de servicios entre servidores si desde un inicio fueron utilizados los enlaces proporcionados por el ESB pueden migrarse sin problema alguno, evitando así la necesidad de realizar ajustes en las aplicaciones (solo sería necesario modificar las configuraciones de los servicios en el ESB).

Una infraestructura basada en ESB proporciona además otro nivel de seguridad. En una organización que proporciona servicios web a terceros, implementar ESB hace que solo tenga un servidor de servicios web de cara a la extranet empresarial, que si, además, se le configura la debida autenticación hace más seguro el sistema de ataques externos.

Características típicas de ESB

Un ESB es una infraestructura de integración distribuida basada en mensajes y estándares abiertos que proporciona servicios de invocación, enrutamiento, mediación, adaptadores, seguridad, administración, orquestación de procesos, procesamiento de eventos complejos y herramientas de integración para facilitar las interacciones de aplicaciones y servicios distribuidos dispares de manera segura y confiable (Vollmer and Gilpin, 2006).

Invocación

La invocación es la capacidad de un ESB para enviar solicitudes y recibir respuestas de servicios de integración y recursos integrados. Esto significa que un ESB tiene que soportar los estándares para la comunicación de servicios web incluyendo SOAP, el Lenguaje de Descripción de Servicios Web (WSDL),

Why use an ESB?

Increasing organizational agility by reducing time-to-market for new initiatives is one of the most common reasons companies implement an ESB. An ESB architecture facilitates this by providing a simple, well-defined, "pluggable" system that scales really well. Additionally, an ESB provides a way to leverage your existing systems and expose them to new applications using its communication and transformation capabilities (MuleSoft, 2019).

An ESB provides facilities to organizations. It allows you to have the web services of your applications gathered in one place, regardless of having several dedicated physical servers. Front-end developers do not need to differentiate between the physical servers where the services for their applications were implemented (they would all be apparently on the same server). In cases of migrating services between servers, if the links provided by the ESB were used from the beginning, they can be migrated without any problem, thus avoiding the need to make adjustments to the applications (it would only be necessary to modify the settings of the services in the ESB) .

An ESB-based infrastructure also provides another level of security. In an organization that provides web services to third parties, implementing ESB means that it only has one web services server facing the corporate extranet, which if, in addition, the proper authentication is configured, makes the system more secure from external attacks.

Typical ESB Features

An ESB is an open standards and messages-based distributed integration infrastructure that provides invocation, routing, mediation, adapter, security, administration, process orchestration, complex event processing, and integration tools to facilitate application and service interactions. Distributed safely and reliably (Vollmer and Gilpin, 2006).

Invocation

Invocation is the ability of an ESB to send requests and receive responses from integration services and integrated resources. This means that an ESB has to support standards for web services communication including SOAP, the Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI) and the WS- *

Descripción Universal, Descubrimiento e Integración (UDDI) y la familia WS- * de estándares. Además, la API de Java Message Service (JMS) y la arquitectura de conector J2EE (JCA), deben implementarse para la integración con los sistemas MOM y los servidores de aplicaciones. Por supuesto, un ESB también debe ser capaz de manejar los protocolos subyacentes como Protocolo de control de transmisión (TCP), Protocolos de Datagramas de Usuario (UDP), Protocolo de Transferencia de Hipertexto (HTTP) o Capa de Sockets Seguros (SSL) (Menge, 2007).

Enrutamiento

El enrutamiento es la capacidad de decidir el destino de un mensaje durante su transporte. Los servicios de enrutamiento son una característica esencial de un ESB porque permiten desacoplar la fuente de un mensaje desde el destino final. Para habilitar el enrutamiento y otras funciones de comunicación, se debe hacer referencia a puntos finales de mensajería dispares. El estándar común para el direccionamiento es utilizar identificadores uniformes de recursos (URI). Además, WS-Addressing (Box *et al*., 2004) puede implementarse en soluciones ESB para describir puntos finales de servicios web de una manera neutral en el transporte. La decisión a qué destino se envía un mensaje se puede tomar en función de varias condiciones que conducen a diferentes tipos de enrutadores (Chappell, 2004).

Una forma especial de un enrutador basado en contenido es un filtro de mensajes. Reenvía un mensaje solo si el contenido coincide con un criterio determinado. De lo contrario, el mensaje será eliminado (Gilpin and Vollmer, 2005).

Los diferentes enrutadores se pueden combinar para crear flujos de mensajes complejos. Todos los enrutadores pueden implementarse como los denominados enrutadores dinámicos. Eso significa que el enrutador puede reconfigurar sus reglas de enrutamiento basándose en mensajes de configuración especiales que pueden ser enviados por los destinos participantes (Ortiz, 2007).

Mediación

La mediación se refiere a todas las transformaciones o traducciones entre recursos dispares, incluido el protocolo de transporte, el formato del mensaje y el contenido del mensaje. Estas transformaciones son muy importantes para la integración porque las aplicaciones

family of standards. Additionally, the Java Message Service (JMS) API and the J2EE Connector Architecture (JCA) must be implemented for integration with MOM systems and application servers. Of course, an ESB must also be able to handle the underlying protocols like Transmission Control Protocol (TCP), User Datagram Protocols (UDP), Hypertext Transfer Protocol (HTTP) or Secure Sockets Layer (SSL) (Menge, 2007).

Routing

Routing is the ability to decide the destination of a message during its transport. Routing services are an essential feature of an ESB because they allow the source of a message to be decoupled from the final destination. To enable routing and other communication features, disparate messaging endpoints must be referenced. The common standard for addressing is to use uniform resource identifiers (URIs). Furthermore, WS-Addressing (Box et al., 2004) can be implemented in ESB solutions to describe web service endpoints in a transport-neutral way. The decision to which destination a message is sent can be made based on various conditions leading to different types of routers (Chappell, 2004).

A special form of a content-based router is a message filter. Forward a message only if the content matches certain criteria. Otherwise, the message will be deleted (Gilpin and Vollmer, 2005).

Different routers can be combined to create complex message flows. All routers can be implemented as so-called dynamic routers. That means that the router can reconfigure its routing rules based on special configuration messages that can be sent by participating destinations (Ortiz, 2007).

Mediation

Mediation refers to all transformations or translations between disparate resources, including the transport protocol, message format, and message content. These transformations are very important for integration because applications rarely match a common data format. Again, XSL and XPath are powerful tools for working with XML messages. Those standards allow an ESB to provide generic XML transformer components that are configured

[5] Consiste en la conversión de datos en una interfaz gráfica para que el usuario pueda ver e interactuar con la información de forma digital usando HTML, CSS y JavaScript. Parte del software que interactúa con el o los usuarios. En términos de diseño web se refiere a la parte del software dedicado a las propias páginas web

rara vez coinciden en un formato de datos común. Nuevamente, XSL y XPath son herramientas poderosas para trabajar con mensajes XML. Esos estándares permiten que un ESB proporcione componentes genéricos de transformador XML que se configuran a través de una hoja de estilo XSL. Los transformadores más complejos pueden invocar otros recursos que están conectados al bus, por ejemplo, una base de datos para aumentar información adicional a un mensaje. Las formas especiales de servicios de transformación son normalizadores, enriquecedores de contenido, filtros de contenido o envoltorios de sobres (Silver, 2004).

Adaptadores

Muchas soluciones ESB proporcionan una amplia gama de adaptadores de aplicaciones. Estos pueden ser adaptadores para paquetes de aplicaciones populares como Enterprise Resource Planning (ERP), Supply Chain Management (SCM) y Customer Relationship Management (CRM). Esos adaptadores se conectan a las interfaces de transacción nativas, API y estructuras de datos que exponen estas aplicaciones comerciales y presentan una interfaz estándar, lo que facilita la reutilización de la lógica y los datos comerciales. Por lo general, la mayoría de los adaptadores de un proveedor en particular funcionan de la misma manera, lo que minimiza las habilidades requeridas para usar cada sistema conectado. El uso de adaptadores prefabricados reduce el trabajo requerido para integrar aplicaciones en ambientes SOA (Chappell, 2004).

Seguridad

Una infraestructura de integración de clase empresarial tiene que proporcionar mensajes seguros. Un ESB pueda cifrar y descifrar el contenido de los mensajes, manejar la autenticación y el control de acceso para los puntos finales de mensajería y utilizar mecanismos de persistencia seguros (Gilpin and Vollmer, 2005).

Administración

Un ESB debe proporcionar servicios de auditoría y registro para monitorear la infraestructura y el escenario de integración y posiblemente también para controlar la ejecución del proceso. Debe haber un mecanismo central para la configuración y administración. Además, se pueden incluir herramientas para la medición de uso (Gilpin and Vollmer, 2005).

Orquestación de Procesos

Un ESB puede incluir un motor para ejecutar procesos de negocios descritos con el Lenguaje de Ejecución de Procesos de Negocios de Servicios Web (WS-BPEL). Este motor controlado por la descripción del proceso

through an XSL stylesheet. More complex transformers can invoke other resources that are connected to the bus, for example, a database to add additional information to a message. Special forms of transformation services are normalizers, content enrichers, content filters, or envelope wrappers (Silver, 2004).

Adapters

Many ESB solutions provide a wide range of application adapters. These can be adapters for popular application packages such as Enterprise Resource Planning (ERP), Supply Chain Management (SCM), and Customer Relationship Management (CRM). Those adapters connect to the native transaction interfaces, APIs, and data structures that these business applications expose and present a standard interface, making it easy to reuse business data and logic. Generally, most adapters from a particular vendor work the same way, minimizing the skills required to use each connected system. The use of prefabricated adapters reduces the work required to integrate applications in SOA environments (Chappell, 2004).

Security

An enterprise-class integration infrastructure has to provide secure messages. An ESB can encrypt and decrypt message content, handle authentication and access control for messaging endpoints, and use secure persistence mechanisms (Gilpin and Vollmer, 2005).

Administration

An ESB should provide auditing and logging services to monitor the infrastructure and integration scenario and possibly also to control the execution of the process. There should be a central mechanism for configuration and administration. In addition, tools for measuring use can be included (Gilpin and Vollmer, 2005).

Process Orchestration

An ESB can include an engine for executing business processes described with the Web Services Business Process Execution Language (WS-BPEL). This motor controlled by the process description then coordinates the collaboration of the services connected to the bus (Jordan et al., 2007).

Complex event processing

luego coordina la colaboración de los servicios conectados al bus (Jordan *et al*., 2007).

Procesamiento de eventos complejos

Un mensaje asincrónico puede verse como un evento, especialmente cuando se usa un canal de publicación-suscripción. Por lo tanto, un ESB puede incluir mecanismos para la interpretación de eventos, la correlación de eventos y la coincidencia de patrones de eventos que permiten arquitecturas controladas por eventos (Jordan *et al*., 2007).

Herramientas de integración

Para el desarrollo profesional existen herramientas de diseño gráfico en entornos ESB, así como herramientas de implementación y pruebas, haciendo de una forma más sencilla la visualización del entorno y la disminución de fallas (Jordan *et al*., 2007).

Propuesta de entorno **Implementaciones de ESB**

A nivel mundial utilizar un ESB es la mejor solución para casi todas las organizaciones porque brinda un conjunto amplio de funcionalidades. Además, tiene un impacto muy positivo en la arquitectura de software de una organización ya que permite resolver diferentes problemas tecnológicos.

En la actualidad existen una gran variedad de implementaciones de ESB, por lo que en ocasiones resulta difícil la elección de uno adecuado debido a que se disponen de muchas opciones. Cada uno con sus características propias que los hacen idóneos para entornos de desarrollo específicos.

Si además de realizar la integración con el ESB se requiere de otras funciones como la Gestión de Procesos de Negocio (BPM), el Monitoreo de Actividad Comercial, la Gestión de Datos Maestros o un Repositorio, las bibliografías recomiendan el uso de una suite. Una suite es una plataforma integral de middleware, que ofrece todas las características de un ESB (denominadas Suites ESB o Paquetes ESB), además de las funciones mencionadas. Con este grupo de herramientas toda la integración se puede realizar con un solo software.

Usar una suite en lugar de combinar varios marcos o productos para crear un conjunto de integración personalizado es muy recomendable ya que esta última opción suele ser innecesariamente costosa y conlleva muchas dificultades adicionales. Por lo tanto, dado que ya existen varias soluciones, es decir, varias suites en el mercado, no se recomienda crear una a partir de varias piezas (Wähner, 2013).

An asynchronous message can be seen as an event, especially when using a publish-subscribe channel. Thus, an ESB can include mechanisms for event interpretation, event correlation, and event pattern matching that enable event-driven architectures (Jordan et al., 2007).

Integration tools

For professional development there are graphic design tools in ESB environments, as well as implementation and testing tools, making it easier to visualize the environment and reduce failures (Jordan et al., 2007).

Environment proposal ESB implementations

Worldwide, using an ESB is the best solution for almost all organizations because it provides a wide set of functionalities. In addition, it has a very positive impact on the software architecture of an organization since it allows solving different technological problems.

There are now a wide variety of ESB implementations, so choosing the right one is sometimes difficult because there are so many options available. Each one with its own characteristics that make them ideal for specific development environments.

If, in addition to integrating with the ESB, other functions are required such as Business Process Management (BPM), Business Activity Monitoring, Master Data Management or a Repository, the bibliographies recommend the use of a suite. A suite is a comprehensive middleware platform, offering all the features of an ESB (called ESB Suites or ESB Packages), in addition to the aforementioned functions. With this group of tools, all integration can be done with a single software.

Using a suite rather than combining multiple frameworks or products to create a custom integration set is highly recommended as the latter option is often unnecessarily expensive and carries many additional difficulties. Therefore, since there are already several solutions, that is, several suites on the market, it is not recommended to create one from several pieces (Wähner, 2013).

Un repaso por el mercado de las Suites ESB ha permitido reconocer algunas de código abierto y otras de pago. La
Tabla **1** Ejemplos de Suite ESB de pago y de código abierto muestra un conjunto de suites ESB, se definen cuáles son de código abierto y cuales propietarias.

A review of the ESB Suites market has allowed us to recognize some of them open source and others for payment. Table 1 shows a set of ESB suites, which are open source and which are proprietary.

Software	Creator	Open Code	License
AdroitLogic UltraESB	AdroitLogic	No	Owner
Mule ESB	MuleSoft	No	Dual (CPAL or owner)
Apache Camel	Apache Software Foundation	Yes	Apache Software License
Apache Kafka	Apache Software Foundation	Yes	Apache Software License
Apache ServiceMix	Apache Software Foundation	Yes	Apache Software License
Apache Synapse	Apache Software Foundation	Yes	Apache Software License
Artix ESB	Progress Software	No	Owner
BizTalk Server	Microsoft	No	Owner
Flow Software	Flow Software Ltd	No	Owner
Fuse - Enterprise Camel	Red Hat	Yes	Apache Software License
IBM Integration Bus (formalmente WebSphere Message Broker)	IBM	No	Owner

Informatica Power Center	Informatica	No	Owner
JBoss Enterprise Service Bus (ESB)	JBoss, a division of Red Hat, Inc.	Yes	GNU LGPL
JBoss Enterprise SOA Platform	JBoss, a division of Red Hat, Inc.	Yes	GNU LGPL
Magic xpi Integration Platform	Magic Software Enterprises	No	Owner
Openadaptor	The Software Conservancy	Yes	Variante de MIT
OpenESB	OpenESB Community	Yes	CDDL
OpenLink Virtuoso	OpenLink Software	Si	Dual (GPL o propietaria)
Oracle BPEL Process Manager	Oracle Corporation	No	Owner
Oracle Enterprise Service Bus	Oracle Corporation	No	Owner
PEtALS ESB	OW2 Consortium	Yes	GNU LGPL
Sonic ESB	Progress Software	No	Owner
SAP NetWeaver Process Integration (short SAP PI)	SAP AG	No	Owner
ServiceMix	Apache Software Foundation	Yes	Apache Software License

Spagic	Engineering group	Yes	GNU LGPL
Sun Java Composite Application Platform Suite	Sun Microsystems	No	Owner
TrackerSuite.Net	Automation Centre	No	Owner
Unify NXJ	Unify Corporation	No	Owner
webMethods Integration Server	Software AG	No	Owner
WebSphere Message Broker (now known as IBM Integration Bus)	IBM	No	Owner
WSO2 Enterprise Integrator	WSO2	Yes	Apache License 2.0

Tabla 1 *Ejemplos de Suite ESB de pago y de código abierto*

Table 1 *Examples of paid and open source ESB Suite*

Proprietary solutions are usually fairly comprehensive and well-polished, but are nevertheless complex, expensive, and sometimes out of business budgets. On the other hand, open source ones are robust, much less expensive and with a large community of developers in charge of their maintenance and documentation. (Wähner, 2013) Table 2 obtained from Choosing the Right ESB for Your Integration Needs compares the advantages and disadvantages of proprietary versus open source ESBs (green = good, yellow = medium, red = bad).

Proprietary solutions are usually fairly comprehensive and well-polished, but are nevertheless complex, expensive, and sometimes out of business budgets. On the other hand, open source ones are robust, much less expensive and with a large community of developers in charge of their maintenance and documentation. (Wähner, 2013) Table 2 obtained from Choosing the Right ESB for Your Integration Needs compares the advantages and disadvantages of proprietary versus open source ESBs (green = good, yellow = medium, red = bad).

Criteria	Owners	Open Code
Easy to use	Very complex installation (consultants sometimes needed).	One Click Installer (One click install), can be used after minutes, unified platform.

Maintainability and Monitoring	Powerful tools (eg for administration and monitoring), no need for source code analysis, refactoring via GUI.	Less powerful tools (for example, for administration and monitoring, sometimes requires the integration of additional products from other vendors), no need for source code analysis, refactoring using GUI.
Comunity	You need to buy support, forums (but not a real community to help).	Based on open source projects, plus its own community.
Business support	24/7 business support, deployments with thousands of servers.	24/7 business support, less guarantees than patented support, (it is necessary to verify the consulting and local support).
Functionality	Integration features and many others.	Integration and other features (sometimes less than paid ones)
Flexibility	Ideologies of: (Make a change request + wait a lot + pay) OR (pay a lot + get it quickly).	Open source, the same team can make necessary modifications.
Extensibility	If it is decided to extend by the same work team it can be difficult (waste of time and resources). It is necessary to pay the company that owns the product.	It is based on international standards. It is easier to expand.
Connectors	Adapters for business products and technologies.	Adapters for business products and technologies.
Costs	Very expensive.	Little or none (you usually pay for support).
Licence	Complex price list, you pay for everything (updates, migration to Virtual Machines).	Subscription model, updates included, predictive costs, possible discounts. There are some free.

Tabla 2 *Principales ventajas y desventajas del uso de los productos propietarios y de código abierto*

Table 2 *Main advantages and disadvantages of using proprietary and open source products.*

Se puede apreciar que las soluciones patentadas generalmente ofrecen más funciones y soporte "potente". Sin embargo, los productos de código abierto ofrecen una usabilidad más simple, una mayor flexibilidad y una mayor extensibilidad.

Elección del ESB

El presente trabajo no proporciona una matriz que compare todos los productos disponibles con respecto a varios criterios. Desde la perspectiva de varios autores, es casi imposible crear una matriz adecuada y útil debido a que los productos ofrecen demasiadas funcionalidades y conceptos (a menudo diferentes). Además, la lista de características también cambia prácticamente todos los días en el mundo tecnológico actual.

You can see that proprietary solutions generally offer more features and "powerful" support. However, open source products offer simpler usability, greater flexibility, and greater extensibility.

Choice of ESB

The present work does not provide a matrix that compares all the available products with respect to various criteria. From the perspective of several authors, it is almost impossible to create a suitable and useful matrix because the products offer too many (often different) functionalities and concepts. Furthermore, the list of features also changes practically every day in today's tech world.

Por lo anterior expuesto se decide centrar la elección en **suite ESB de código abierto**: softwares libres (libre de costos), que den soporte a servicios web API REST, que sean intuitivos y fácil de utilizar, que estén bien documentados y sean escalables. También que cuente con una gran comunidad y tenga soporte internacional, que pertenezca a un proyecto de prestigio y con varios años de vida.

Una revisión de las suites ESB principales del mercado que se adecúen a los requerimientos anteriores arrojó que WSO2 Enterprise Integrator (suite contenedora de WSO2 ESB, perteneciente a WSO2) es una de las mejores opciones según la comunidad internacional. Sitios web como G2 (https://www.g2.com/), It Central Station (https://www.itcentralstation.com/) y Trust Radius (https://www.trustradius.com/), sitios dedicados a realizar ranking internacionales de uso de aplicaciones por la comunidad, colocan a WSO2 entre las primeras diez suites ESB recomendados por su gran número de funcionalidades y robusteza. Además de que se adapta a las necesidades de la organización.

WSO2 es un proveedor con alrededor de 10 años en el mercado. Son varias las empresas mundialmente conocidas que han apostado por utilizar su suite ESB. Empresas tales como eBay, Motorola, West, Verifone, QANTAS, Experian, Trimble, CellCard, StubHub, CitySprint, UNITED, VERIDIUM, WELLS FARGO, entre otras figuran la extensa lista. Varias de ellas expresan sus experiencias en la implementación de aplicaciones WSO2 acogidas como testimonios en su sitio oficial:

eBay

eBay utiliza la plataforma de integración de código abierto de WSO2 para procesar más de mil millones de transacciones por día. Fue seleccionada después de un proceso de evaluación intensivo, ya que eBay sintió que era el mejor producto capaz de manejar los requisitos de middleware de su mercado en línea. Además de superar a otros productos de software en términos de velocidad y confiabilidad, WSO2 demostró la flexibilidad para crecer y adaptarse a los requisitos cambiantes de eBay para manejar transformaciones, orquestaciones y flujos de mensajes complejos. eBay ha proporcionado con éxito una experiencia de compra confiable y eficiente a su vasta base global de clientes con la ayuda de WSO2 y continúa invirtiendo en ella (WSO2, 2019).

"El uso de WSO2 ESB nos ha ayudado a administrar nuestros requisitos de eficiencia, escalabilidad y

Due to the above, it was decided to focus the choice on an **open source ESB suite**: free software (free of charge), that supports REST API web services, that are intuitive and easy to use, that are well documented and are scalable. Also that it has a large community and international support, that it belongs to a prestigious project and has been around for several years.

A review of the main ESB suites on the market that meet the above requirements found that WSO2 Enterprise Integrator (WSO2 containing WSO2 ESB suite) is one of the best options according to the international community. Websites such as G2 (https://www.g2.com/), It Central Station (https://www.itcentralstation.com/) and Trust Radius (https://www.trustradius.com/), dedicated sites To carry out international ranking of application use by the community, they place WSO2 among the first ten ESB suites recommended for its large number of functionalities and robustness. In addition to adapting to the needs of the organization.

WSO2 is a supplier with around 10 years in the market. There are several world-renowned companies that have opted to use their ESB suite. Companies such as eBay, Motorola, West, Verifone, QANTAS, Experian, Trimble, CellCard, StubHub, CitySprint, UNITED, VERIDIUM, WELLS FARGO, among others feature the extensive list. Several of them express their experiences in the implementation of WSO2 applications hosted as testimonials on their official site:

eBay

eBay uses the WSO2 open source integration platform to process over 1 billion transactions per day. It was selected after an intensive evaluation process, as eBay felt it was the best product capable of handling the middleware requirements of its online marketplace. In addition to outperforming other software products in terms of speed and reliability, WSO2 demonstrated the flexibility to grow and adapt to eBay's changing requirements to handle complex message flows, orchestrations and transformations. eBay has successfully provided a reliable and efficient shopping experience to its vast global customer base with the help of WSO2 and continues to invest in it (WSO2, 2019).

"Using WSO2 ESB has helped us manage our requirements for efficiency, scalability and security.

seguridad. Desde una perspectiva comercial, hemos cumplido nuestros objetivos hasta la fecha". Afirmó Abhinav Kumar, (Gerente Senior de Ingeniería de Sistemas en eBay) acerca de su experiencia sobre la implementación de WSO2.

Motorola

Un colapso del sitio web posterior al Día de Acción de Gracias y la pérdida resultante de transacciones de los compradores llevaron a Motorola a pasar a la plataforma de integración ligera de WSO2. En un intento por mejorar la rentabilidad de su nueva arquitectura, Motorola utiliza las capacidades de integración de WSO2 en AWS y OpenStack. Actualmente tiene 35 aplicaciones con el 90 % del tráfico que pasa por OpenStack y el 10 % por AWS. Motorola ahora está equipado con operaciones comerciales más fluidas con la ayuda de la tecnología WSO2 (WSO2, 2019).

"WSO2 proporciona una plataforma de integración ligera con las capacidades técnicas adecuadas que necesitábamos". Comentó Sri Harsha Pulleti (arquitecto de integración de Motorola Mobility) acerca de su experiencia con la implementación de WSO2.

Verifone

Verifone es uno de los proveedores de terminales de punto de venta (POS) más grandes del mundo y un proveedor líder de soluciones de pago y comercio. Con operaciones en más de 150 países y casi 6000 empleados en todo el mundo, Verifone procesa 7.6 mil millones de transacciones anualmente. Las capacidades de IoT (Internet of Things)[6] de WSO2 se utilizaron para crear Verifone Carbon, un terminal de pago que establece un nuevo estándar para una experiencia de consumo valiosa y atractiva. El uso de Verifone habilitado para WSO2 evita el bloqueo del proveedor, permite tener control sobre el código fuente e integrarse fácilmente con la infraestructura de administración existente (WSO2, 2019).

"Con WSO2, pudimos crear una solución que se ajusta a nuestras necesidades exactas. También es flexible y de bajo costo". Afirma Ulrich Herberg, Arquitecto Senior de Java en Verifone sobre su experiencia en la implementación de WSO2.

WSO2 ESB

From a business perspective, we have met our objectives to date. "Said Abhinav Kumar, (Senior Manager of Systems Engineering at eBay) about his experience on WSO2 implementation.

Motorola

A post-Thanksgiving website crash and resulting loss of shopper transactions led Motorola to move to WSO2's lightweight integration platform. In an attempt to improve the cost effectiveness of its new architecture, Motorola uses the integration capabilities of WSO2 on AWS and OpenStack. It currently has 35 applications with 90% of the traffic going through OpenStack and 10% through AWS. Motorola is now equipped with smoother business operations with the help of WSO2 technology (WSO2, 2019).

"WSO2 provides a lightweight integration platform with the right technical capabilities that we needed." Sri Harsha Pulleti (Motorola Mobility Integration Architect) commented on his experience with implementing WSO2.

Verifone

Verifone is one of the world's largest point-of-sale (POS) terminal providers and a leading provider of commerce and payment solutions. With operations in more than 150 countries and nearly 6,000 employees around the world, Verifone processes 7.6 billion transactions annually. WSO2's Internet of Things (IoT) capabilities were used to create Verifone Carbon, a payment terminal that sets a new standard for a rich and engaging consumer experience. Using Verifone enabled for WSO2 bypasses vendor lock-in, allows control over source code, and easily integrates with existing management infrastructure (WSO2, 2019).

"With WSO2, we were able to create a solution that meets our exact needs. It is also flexible and inexpensive ". Ulrich Herberg, Senior Java Architect at Verifone, says about his experience implementing WSO2.

WSO2 ESB

[6] IoT (Internet of things o Internet de las Cosas) es un concepto que se refiere a una interconexión digital de objetos cotidianos con internet. Es, en definitiva, la conexión a internet más con objetos que con las personas. También se conoce como internet de todas las cosas.

WSO2 ESB es el principal motor de integración de WSO2 Enterprise Integrator (WSO2 EI). Basado en Java, permite a los desarrolladores integrar servicios y aplicaciones de una manera fácil, eficiente y productiva. También proporciona la facilidad de conectar aplicaciones en la nube utilizando una amplia gama de conectores que vienen listos para usarse (WSO2, 2019).

Permite conectar y reutilizar los activos y sistemas existentes implementados utilizando tecnologías heterogéneas, incluidos servicios web, micro servicios, HTTP, JMS, JDBC, entre otros. Se incluye una amplia gama de patrones de integración empresarial en el ESB; patrones como el encadenamiento de servicios, almacenamiento y reenvío. Se pueden usar para ayudar a conectar múltiples servicios y componer servicios de valor agregado en un tiempo rápido (WSO2, 2019).

WSO2 proporciona la gama completa de componentes de una suite, incluidos Business Process Server, Business Rules Server, Business Activity Monitor o Governance Registry. Toda la plataforma WSO2 se puede instalar muy fácilmente y ofrece un estudio de desarrollo ligero basado en Eclipse. Incluye principalmente proyectos de código abierto como Apache Synapse (ESB ligero), Axis (Implementación de servicios web) u ODE (Business Process Engine) en sus componentes (Wähner, 2013).

WSO2 es de los pocos proveedores que ofrece un conjunto completo que se basa en una única base de código y un único entorno de desarrollo. Por lo tanto, nada se interpone en el camino para un proceso de desarrollo iterativo, comenzando con un pequeño par de características y agregando funcionalidades más adelante paso a paso (Wähner, 2013).

Es una plataforma configurable y no programable que tiene como base principal un ESB basado en Apache. Brinda una mejor gestión, desarrollo y capacidades SOA, por ese motivo reduce el tiempo de preparación y aprendizaje del desarrollador (Benalcazar *et al*., 2017).

Entre sus principales componentes se encuentran los siguientes (Benalcazar *et al*., 2017):

- ESB, es el elemento más rápido al momento de implementar un bus de servicios, es el núcleo de la misma.
- Governance Registry, es un registro de los artefactos de gobierno.

WSO2 ESB is the main integration engine of WSO2 Enterprise Integrator (WSO2 EI). Based on Java, it allows developers to integrate services and applications in an easy, efficient and productive way. It also provides the facility to connect cloud applications using a wide range of connectors that come out of the box (WSO2, 2019).

It allows to connect and reuse existing assets and systems implemented using heterogeneous technologies, including web services, microservices, HTTP, JMS, JDBC, among others. A wide range of business integration patterns are included in the ESB; patterns like chaining of services, store and forward. They can be used to help connect multiple services and compose value-added services in a quick time (WSO2, 2019).

WSO2 provides the full range of components in a suite, including Business Process Server, Business Rules Server, Business Activity Monitor or Governance Registry. The entire WSO2 platform can be installed very easily and offers a lightweight development studio based on Eclipse. It mainly includes open source projects such as Apache Synapse (lightweight ESB), Axis (Web Services Implementation) or ODE (Business Process Engine) in its components (Wähner, 2013).

WSO2 is one of the few vendors that offers a complete suite that is based on a single code base and a single development environment. Thus, nothing stands in the way of an iterative development process, starting with a small couple of features and adding functionality later step by step (Wähner, 2013).

It is a configurable and non-programmable platform based on an ESB based on Apache. It provides better management, development and SOA capabilities, for that reason it reduces the developer's preparation and learning time (Benalcazar et al., 2017).

Among its main components are the following (Benalcazar et al., 2017):

• ESB, is the fastest element when implementing a service bus, it is the core of it.

• Governance Registry, is a registry of government artifacts.

- BAM (Bussines Activity Monitor), permite llevar un monitoreo del negocio.
- DSS (Data Service Server), permite traducir un conjunto de datos a un servicio web.

Arquitectura de WSO2 Benalcazar *et al*. (2017):

- La aplicación envía un mensaje al ESB.
- El mensaje ingresa por uno de los puertos configurados.
- El protocolo envía el mensaje por un canal de mensajería el cual utiliza aspectos de seguridad.
- ESB al ser un medio para transferir el mensaje decide la ruta que deberá tomar el mismo.
- El mensaje es enviado por el canal de mensajería de salida.

Componentes de la arquitectura WSO2 (Benalcazar *et al*., 2017):

- Protocolos: WSO2 ESB utiliza varios protocolos para el proceso de ingreso de mensajes y su respectiva distribución.
- Constructores de mensajes y formateadores: El constructor procesa el dato ingresado en el mensaje y lo convierte en un XML común, que puede ser leído y entendido por el motor de mediación del ESB. El formateador de mensajes se utiliza para construir el flujo de salida a partir de un mensaje nuevo, el formateador es seleccionado mediante el tipo de contenido del mensaje.
- Endpoints: Un endpoint se define como un destino externo (servidor) para un mensaje. Un endpoint se conecta con cualquier servicio externo después de haber sido configurado con los atributos necesarios para la comunicación con ese servicio, es independiente de los medios de transporte o protocolos. Al configurar la secuencia de mensajes de mediación o un servicio de proxy se especifica el protocolo que se utilizará y el endpoint por donde se enviará el mensaje.
- Servicios Proxy: Son servicios virtuales que reciben los mensajes y los procesan antes de enviarlos al servicio determinado. Los protocolos se pueden utilizar para enviar y recibir los mensajes de los servicios proxy. El servicio proxy es visible desde el exterior y se accede utilizando una URL muy similar a las utilizadas en Internet.
- API: Una API está anclada en un contexto URL definido por el usuario, al igual que una aplicación web implementada con un contenedor de servlets está anclada a una URL fija.
- Inbound Endpoints: Los inbound endpoints permiten a los protocolos trabajar en un modo multi-inquilino.

• BAM (Business Activity Monitor), allows you to monitor the business.

• DSS (Data Service Server), allows you to translate a set of data to a web service.

Architecture of WSO2 Benalcazar et al. (2017):

• The application sends a message to the ESB.

• The message enters through one of the configured ports.

• The protocol sends the message through a messaging channel which uses security aspects.

• ESB, being a means to transfer the message, decides the route that it should take.

• The message is sent through the outgoing messaging channel.

Components of the WSO2 architecture (Benalcazar et al., 2017):

• Protocols: WSO2 ESB uses several protocols for the message input process and their respective distribution.

• Message constructors and formatters: The constructor processes the data entered in the message and converts it into a common XML, which can be read and understood by the ESB's mediation engine. The message formatter is used to build the output flow from a new message, the formatter is selected by the content type of the message.

• Endpoints: An endpoint is defined as an external destination (server) for a message. An endpoint connects to any external service after having been configured with the necessary attributes for communication with that service, it is independent of the means of transport or protocols. When configuring the sequence of mediation messages or a proxy service, you specify the protocol to be used and the endpoint through which the message will be sent.

• Proxy Services: These are virtual services that receive the messages and process them before sending them to the specific service. The protocols can be used to send and receive messages from proxy services. The proxy service is visible from the outside and is accessed using a URL very similar to those used on the Internet.

• API: An API is anchored in a user-defined URL context, just as a web application implemented with a servlet container is anchored to a fixed URL.

- Topics: Permite recibir mensajes cuando un evento se produce mediante la subscripción de mensajes que se han publicado en un tópico específico.
- Mediadores: Son unidades de procesamiento individuales que realizan una función específica como el envío, la transformación o el filtrado de mensajes.
- Secuencias: Son un conjunto de mediadores organizados en una secuencia lógica que permite implementar patrones de canales de comunicación y filtros. Agrega secuencias de los servicios proxy y API.
- Tareas: Una tarea le permite ejecutar una parte de código activado por un temporizador, es capaz de establecer tareas personalizadas mediante una interfaz JAVA.
- Componente QoS: El componente permite establecer calidad del servicio mediante la seguridad y mensajería confiable en un servicio proxy.
- Registros: Un registro es un almacén de contenido y repositorio de metadatos. Se puede utilizar registros externos de recursos tales como WSDL, esquemas, guiones, XSLT y transformaciones XQUERY.
- Gestión y configuración GUI: WSO2 contiene una interfaz gráfica lo cual permite configurar fácilmente los componentes anteriormente mencionados.
- Plataforma Carbon: Permite el entorno de ejecución del ESB. Contiene las características como la seguridad, logueo, el agrupamiento, el almacenamiento en caché, etc. Esta plataforma permite instalar características adicionales como: Business Activity Monitor, Identity Server, Message Broker, entre otras.

WSO2 es fácil de instalar, cuenta con licencia Apache 2.0 y tiene un alto rendimiento incluso en sistemas complejos. También proporciona una mensajería confiable en un servicio proxy. Su suite está basada en código único y expansible y con altas funcionalidades. Un punto importante en WSO2 es su agilidad al cambio ya que permite agregar o quitar funcionalidades al entorno sin que afecte al funcionamiento del mismo (Benalcazar *et al*., 2017).

WSO2 ESB ofrece una amplia gama de capacidades de integración desde enrutamiento de mensajes simples hasta integración fluida de sistemas propietarios complejos. Es totalmente compatible con estándares para patrones de integración empresarial (EIP). También potencia escenarios de integración locales y basados en la nube con numerosos conectores que le permiten integrarse sin problemas a

• Inbound Endpoints: Inbound endpoints allow protocols to work in a multi-tenant mode.

• Topics: Allows you to receive messages when an event occurs by subscribing to messages that have been published on a specific topic.

• Mediators: These are individual processing units that perform a specific function such as sending, transforming or filtering messages.

• Sequences: They are a set of mediators organized in a logical sequence that allows the implementation of communication channel patterns and filters. Add API and proxy services streams.

• Tasks: A task allows you to execute a piece of code activated by a timer, it is capable of establishing custom tasks through a JAVA interface.

• QoS component: The component allows establishing quality of service through security and reliable messaging in a proxy service.

• Records: A record is a content store and metadata repository. You can use external resource records such as WSDL, schemas, scripts, XSLT, and XQUERY transformations.

• Management and GUI configuration: WSO2 contains a graphical interface which allows the previously mentioned components to be easily configured.

• Carbon Platform: Allows the ESB execution environment. It contains the features like security, logging, grouping, caching, etc. This platform allows you to install additional features such as: Business Activity Monitor, Identity Server, Message Broker, among others.

WSO2 is easy to install, Apache 2.0 licensed, and has high performance even on complex systems. It also provides reliable messaging on a proxy service. Its suite is based on unique and expandable code and with high functionalities. An important point in WSO2 is its agility to change since it allows adding or removing functionalities to the environment without affecting its operation (Benalcazar et al., 2017).

WSO2 ESB offers a wide range of integration capabilities from simple message routing to seamless integration of complex proprietary systems. It is fully compliant with standards for Enterprise Integration Patterns (EIP). It also powers local and cloud-based

populares servicios como Salesforce, PayPal y Twitter (Indrasiri, 2016).

El rendimiento y la latencia de cualquier solución ESB es un factor vital cuando se trata de manejar grandes volúmenes de mensajes. En este punto WSO2 ESB es de los mejores. La estabilidad también es otro aspecto que va de la mano con el rendimiento. Miles de implementaciones de producción de WSO2 ESB muestran su estabilidad y su madurez (Indrasiri, 2016).

WSO2 ESB es parte de la plataforma integral de middleware o suite de integración WSO2, esto hace que usar WSO2 ESB sea muy ventajoso porque además de un ESB se tiene una suite con todos sus componentes (Indrasiri, 2016).

Cuando se están construyendo soluciones de integración empresarial en el mundo real, es necesario la integración de las capacidades ofrecidas por WSO2 ESB, así como otras capacidades de middleware como: gestión de API, gestión de identidad, servicios de datos, análisis, procesamiento incluso complejo, que están más allá del alcance de un ESB, para esto es necesario usar una suite. Pocos proveedores de ESB que no están basados en un concepto de plataforma han tratado de tener todas estas características en un producto ESB, pero han fallado porque tales soluciones no pueden abordar los requisitos modernos de tecnologías de la información de la empresa. Sin embargo, una plataforma de middleware WSO2 está construida desde cero facilitando todos los requisitos de middleware empresarial (Indrasiri, 2016).

Entre las principales características de WSO2 se aprecian que (Benalcazar *et al*., 2017):

- Proporciona un uso completo de XML y servicios web, listo para satisfacer las necesidades del sistema que se integre.
- Permite la interoperabilidad con servicios web incluyendo con Microsoft.NET.
- Optimiza los recursos al utilizar altos niveles de rendimiento, es decir, permite varias conexiones utilizando un espacio de memoria constante y con esto permite ser escalable y funcionar sin problemas.
- Utiliza mínimo desarrollo y soporta los requisitos más comunes y puede extender sus capacidades.
- Es multiprotocolo, permite integrar los que se encuentren en la red.
- WSO2 establece la capacidad de programar tareas como tareas repetitivas.
- Incorpora soporte para leer o escribir bases de datos, llamadas de clases Java y scripts.

integration scenarios with numerous connectors that allow it to seamlessly integrate with popular services such as Salesforce, PayPal, and Twitter (Indrasiri, 2016).

The performance and latency of any ESB solution is a vital factor when it comes to handling large volumes of messages. At this point WSO2 ESB is one of the best. Stability is also another aspect that goes hand in hand with performance. Thousands of production implementations of WSO2 ESB show its stability and maturity (Indrasiri, 2016).

WSO2 ESB is part of the integral middleware platform or WSO2 integration suite, this makes using WSO2 ESB very advantageous because in addition to an ESB there is a suite with all its components (Indrasiri, 2016).

When you are building business integration solutions in the real world, you need to integrate the capabilities offered by WSO2 ESB, as well as other middleware capabilities such as: API management, identity management, data services, analytics, even complex processing , which are beyond the scope of an ESB, for this you need to use a suite. Few ESB vendors that are not based on a platform concept have tried to have all of these features in an ESB product, but have failed because such solutions cannot address the modern IT requirements of the enterprise. However, a WSO2 middleware platform is built from scratch facilitating all enterprise middleware requirements (Indrasiri, 2016).

Among the main characteristics of WSO2, it can be seen that (Benalcazar et al., 2017):

• Provides full use of XML and web services, ready to meet the needs of the system to be integrated.

• Allows interoperability with web services including with Microsoft.NET.

• It optimizes resources by using high levels of performance, that is, it allows multiple connections using a constant memory space and thus allows it to be scalable and work without problems.

• Uses minimal development and supports the most common requirements and can extend its capabilities.

• It is multiprotocol, it allows integrating those found in the network.

- WSO2 proporciona una interfaz gráfica para configurar, gestionar y supervisar el servidor donde se encuentra el ESB.

Son varias las ventajas del uso de WSO2 (Benalcazar *et al*., 2017):

- Permite la integración de varias aplicaciones.
- Es capaz de utilizar varios entornos con diferentes tecnologías y protocolos.
- Es de código abierto.
- El costo y mantenimiento del ESB es reducido al ser una herramienta de código abierto, con esto no quiere decir que sea gratuita, pero resulta mucho más económica que otras plataformas al ser libre de licenciamiento.

Validación
Características del entorno actual

En la UNAH se hacen uso de estándares internacionales para el desarrollo de aplicaciones de software. En los últimos años se ha apostado por el uso de aplicaciones basadas en servicios API REST, por su versatilidad y facilidad de uso. Por lo general se crean aplicaciones utilizando el lenguaje de programación JAVA (lenguaje que se imparte en la carrera de Ingeniería Informática).

Java es un lenguaje de programación de alto nivel dedicado a la realización de aplicaciones multiplataforma. A pesar de poseer aplicaciones en el desarrollo de páginas web llamado JAVAEE web con framework como JSP y JSF, se ha quedado atrás en el tiempo y no permite un nivel de personalización adecuado a los estándares actuales. Lo cual llevó a un análisis por los especialistas del centro a buscar alternativas para el desarrollo de aplicaciones que fomentaran una mejor utilización de las tecnologías actuales.

Producto de la investigación se potenció el uso de SpringBoot, un framework para el desarrollo de aplicaciones Back-end[7] perteneciente a la suite Spring. Este framework genera conexiones API REST con sus clientes, funcionando como Back-end de las aplicaciones. Al incluirse con PostgreSQL (gestor de bases de datos relacional utilizado en el centro) mediante el uso de la librería Persistence (librería JAVA dedicada a la conexión de proyectos JAVA con bases de datos relacionales) se obtiene el Back-end del software deseado. Cabe destacar que pueden ser

• WSO2 establishes the ability to schedule tasks as repetitive tasks.

• Incorporates support for reading or writing databases, Java class calls and scripts.

• WSO2 provides a graphical interface to configure, manage and monitor the server where the ESB is located.

There are several advantages of using WSO2 (Benalcazar et al., 2017):

• Allows the integration of various applications.

• It is capable of using various environments with different technologies and protocols.

• It is open source.

• The cost and maintenance of the ESB is reduced as it is an open source tool, this does not mean that it is free, but it is much cheaper than other platforms as it is free of licensing.

Validation
Characteristics of the current environment

At UNAH, international standards are used for the development of software applications. In recent years, it has opted for the use of applications based on API REST services, due to its versatility and ease of use. In general, applications are created using the JAVA programming language (a language taught in the Computer Engineering career).

Java is a high-level programming language dedicated to building cross-platform applications. Despite having applications in the development of web pages called JAVAEE web with frameworks such as JSP and JSF, it has been left behind in time and does not allow a level of customization adequate to current standards. This led to an analysis by the specialists of the center to look for alternatives for the development of applications that would promote a better use of current technologies.

As a result of the research, the use of SpringBoot, a framework for the development of Back-end applications belonging to the Spring suite, was promoted. This framework generates API REST connections with its clients, functioning as Back-end of the applications. By being included with PostgreSQL (relational database manager used in the

[7] Parte del desarrollo web que se encarga de que toda la lógica de una página web funcione. Se trata del conjunto de acciones que pasan en una web no visible por el usuario, como, por ejemplo, las comunicaciones con el servidor.

utilizados otros lenguajes de programación como Python (con su framework Django) u otros lenguajes siempre y cuando este permita el uso de servicios API REST y sea software libre. Así mismo se pueden utilizar otros gestores de bases de datos como MariaDB o MongoDB.

Para garantizar el Front-end se ha utilizado en el centro el framework Angular, aunque en algunos casos se ha propuesto el uso de React o VueJs. Para la presente investigación se probarán las conexiones con las API REST utilizando el software Postman[8] y la opción de testeo de API de WSO2 por defecto Swagger[9].

Se tomó como ejemplo un pequeño proyecto basado en esta estructura, la

Figura 2 Estructura del software de ejemplomuestra la composición del entorno.

center) by using the Persistence library (JAVA library dedicated to connecting JAVA projects with relational databases), the desired software Back-end is obtained. It should be noted that other programming languages such as Python (with its Django framework) or other languages can be used as long as it allows the use of REST API services and is free software. Likewise, other database managers such as MariaDB or MongoDB can be used.

To guarantee the Front-end, the Angular framework has been used in the center, although in some cases the use of React or VueJs has been proposed. For this research, the connections with the REST APIs will be tested using the Postman software and the default WSO2 API testing option Swagger.

A small project based on this structure was taken as an example, Figure 2 shows the composition of the environment.

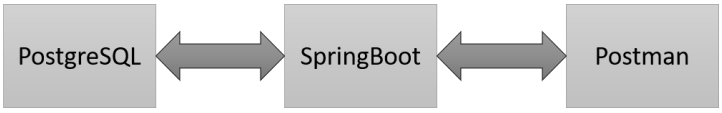

Figura 2 *Estructura del software de ejemplo*

Figure 2 *Sample software structure*

El sistema contiene una simple base de datos que se compone de cuatro tablas. La Figura 3 Proyecto Librería usado como ejemplo para esta investigaciónmuestra la estructura de la base de datos.

The system contains a simple database that is made up of four tables. Figure 3 shows the structure of the database.

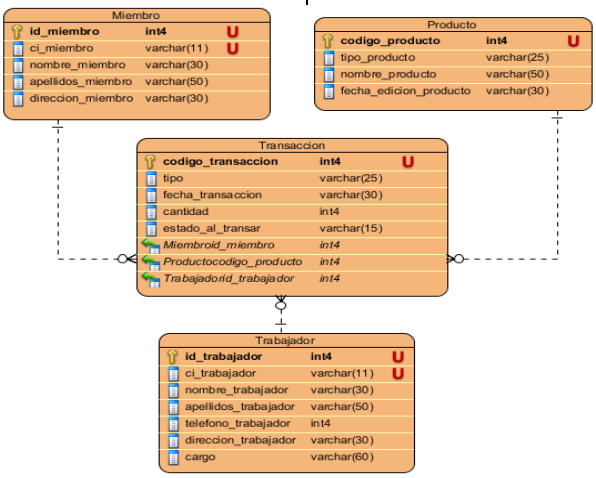

Figura 3 *Proyecto Librería usado como ejemplo para esta investigación*

[8] Herramienta utilizada sobre todo para el testing de API REST. Se pueden testar, consumir y depurar API REST; así como, monitorizarlas, documentarlas, simularlas, etc.

[9] Framework de código abierto para diseñar, construir, consumir y documentar servicios web REST.

Figure 3 *Library Project used as an example for this research*

Para probar las API REST del proyecto antes mencionado se realizaron servicios web genéricos de las operaciones simples FIND_ALL, FINDBY_ID, INSERT, EDIT y DELETE de cada una de las tablas de la base de datos.

De acuerdo a lo anterior planteado se muestran en la Tabla 3 Direcciones API REST generadas por SpringBoot para la tabla Miembro (Direcciones URL y Métodos de conexión) las direcciones URL de las API REST generadas por SpringBoot para la tabla Miembro.

To test the REST APIs of the aforementioned project, generic web services were performed with the simple FIND_ALL, FINDBY_ID, INSERT, EDIT and DELETE operations of each of the database tables.

In accordance with the foregoing, Table 3 shows the URL addresses of the REST APIs generated by SpringBoot for the Member table.

Tabla 3 *Direcciones API REST generadas por SpringBoot para la tabla Miembro (Direcciones URL y Métodos de conexión)*

Table 3 *SpringBoot-generated REST API addresses for the Member table (URLs and Connection Methods)*

Method	URL	JSON parameters (example)
GET	http://192.168.137.1:8080/Miembro/findAll	-
GET	http://192.168.137.1:8080/Miembro/find/(Alicante, Arroyave and Cardona, Azad, Box et al., Box et al., Chappell, Clark and Deach, Clark and DeRose, Fallas, Fernández Núñez, Iyer and Balasundaram, Jordan et al., Kusák, Rodríguez Freire, Siddiqui et al., Snyder, UCI)	-
POST	http://192.168.137.1::8080/Miembro/créate	{ "ci_miembro": "980412232014", "nombre_miembro": "Fernando", "apellidos_miembro": "Lugones", "direccion_miembro": "calle 21" }
PUT	http://192.168.137.1::8080/Miembro/update	{ "id_miembro": 32, "ci_miembro": "980412232014", "nombre_miembro": "Fernando", "apellidos_miembro": "Lugones", "direccion_miembro": "calle 21" }
DELETE	http://192.168.137.1:8080/Miembro/delete/{id}	-
Nota: el caso de las URL que tienen {id} en su ruta, corresponde a los identificadores (campo id_miembro) de los Miembros que se pasan como parámetros y se buscan en la base de datos.. **Nota: the case of URLs that have {id} in their route, corresponds to the identifiers (campo id_miembro) of the Members that are passed as parameters and searched in the database.**		

Características del entorno al desplegar WSO2 Enterprise Integrator

Con la inclusión de WSO2 Enterprise Integrator (WSO2 EI) la conexión entre sistemas quedaría como muestra la Figura 4 Conexiones de los proyectos utilizando WSO2 EI**¡Error! No se encuentra el origen de la referencia.**. En la misma se aprecian una serie de proyectos API REST y un conjunto de aplicaciones que consumen estos servicios. La cantidad de proyectos API REST (Back-end) no necesariamente tiene que coincidir con la cantidad de sistemas consumidores de estos servicios (Front-end).

Environment characteristics when deploying WSO2 Enterprise Integrator

With the inclusion of WSO2 Enterprise Integrator (WSO2 EI), the connection between systems would be as shown in Figure 4. It shows a series of API REST projects and a set of applications that consume these services. The number of API REST projects (Back-end) does not necessarily have to match the number of systems consuming these services (Front-end).

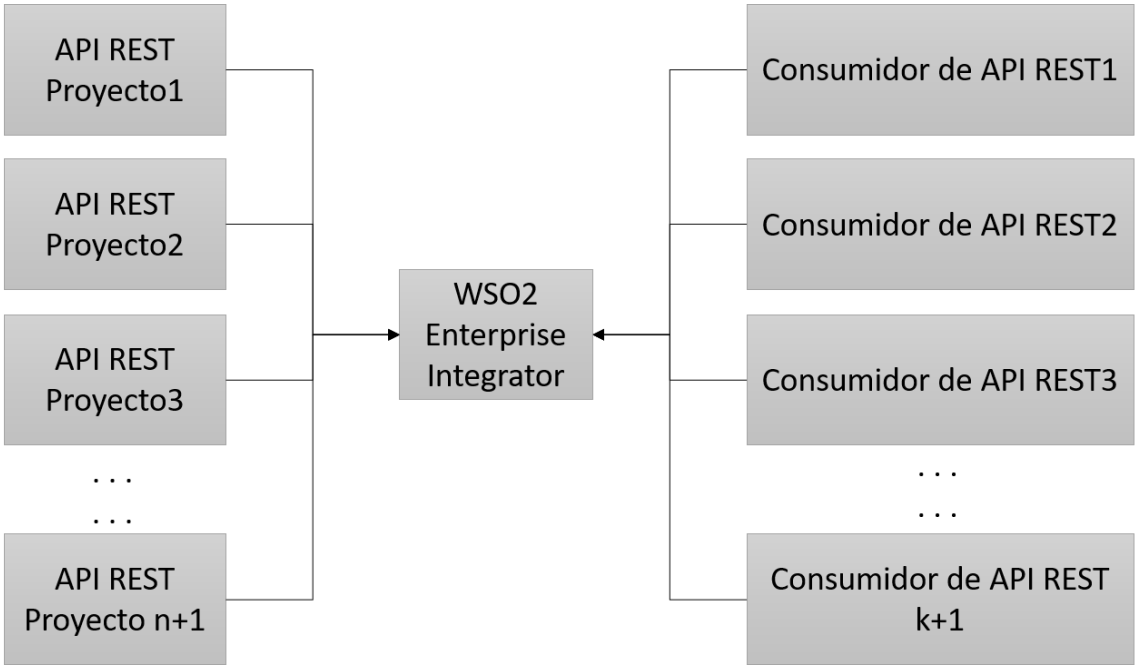

Figura 4 *Conexiones de los proyectos utilizando WSO2 EI*

Figure *4 Project connections using WSO2 EI*

Como se ha explicado a lo largo de este trabajo una de las funcionalidades principales de WSO2 EI es funcionar como intermediario entre Front-end y Back-end. Por lo cual la funcionalidad más utilizada en la presente investigación es la de consumir los servicios API REST de las aplicaciones Back-end y proporcionar enlaces URL del tipo API REST para las aplicaciones Front-end.

Instalación y despliegue de WSO2 EI

Para desplegar WSO2 EI es necesario descargar desde el sitio oficial el instalador. En este caso se descarga WSO2 EI, sistema que cuenta con todas las funcionalidades de ESB más algunas herramientas de Proxy, API Manager y Business Intelligence. Hasta el momento de la presente investigación la versión más

As it has been explained throughout this work, one of the main functionalities of WSO2 EI is to function as an intermediary between Front-end and Back-end. Therefore, the most used functionality in this research is to consume the REST API services of the Back-end applications and provide URL links of the REST API type for the Front-end applications.

Installation and deployment of WSO2 EI

To deploy WSO2 EI it is necessary to download the installer from the official site. In this case, WSO2 EI is downloaded, a system that has all the ESB functionalities plus some Proxy, API Manager and Business Intelligence tools. Until the moment of the present investigation, the most updated version of

actualizada de WSO2 EI es la 6.6.0, la cual fue utilizada durante el transcurso de la misma.

En sistemas Windows se descarga el instalador con formato **.MSI**. La instalación cuenta con pocas personalizaciones, sólo se modifica la localización donde será instalado (por defecto en Program Files). Una vez instalado se ejecuta el fichero *~/WSO2/Enterprise Integrator/6.6.0/bin/integrator.bat*. Es importante destacar que una vez instalado en sistemas Windows puede ejecutarse en sistemas GNU/Linux copiando el directorio y ejecutando el fichero *~/WSO2/Enterprise Integrator/6.6.0/bin/integrator.sh*. Se ejecuta una consola, tener en cuenta que al cerrarla se detiene abruptamente el servicio de WSO2 EI.

WSO2 EI is 6.6.0, which was used during the course of the investigation.

On Windows systems, the installer is downloaded in **.MSI** format. The installation has few customizations, only the location where it will be installed is modified (by default in Program Files). Once installed, the file *~ / WSO2 / Enterprise Integrator / 6.6.0 / bin / integrator.bat* is executed. It is important to note that once installed on Windows systems it can be run on GNU / Linux systems by copying the directory and executing the file ~ / WSO2 / Enterprise Integrator / 6.6.0 / bin / integrator.sh. A console is run, bear in mind that closing it stops the WSO2 EI service abruptly.

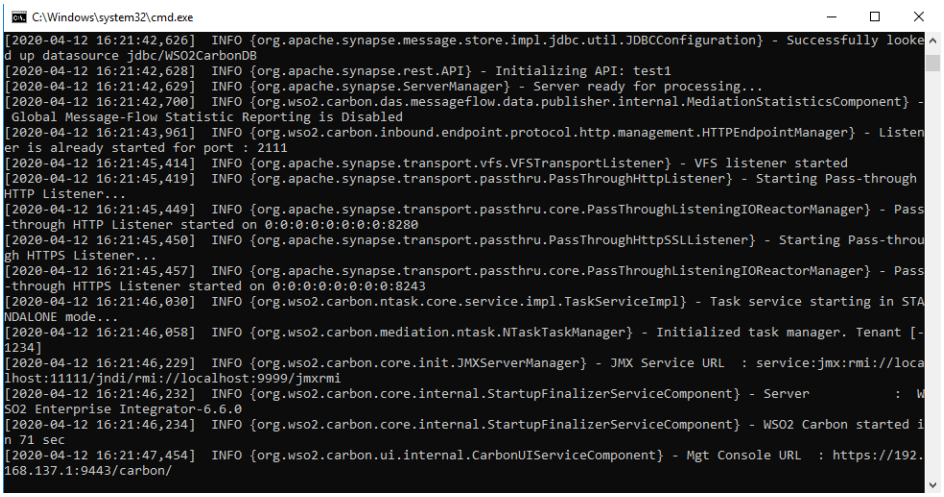

Figura 5 *Consola de ejecución de WSO2 EI*

Figure 5 *WSO2 EI Execution Console*

La Figura 5 Consola de ejecución de WSO2 EI muestra la lista de sucesos una vez ejecutado el fichero *integrator.bat* en Windows. En la consola aparecen cada una de las operaciones de tipo Log que se ejecutan a lo largo del tiempo en el sistema. En la imagen se muestra en la última línea la ruta de acceso del panel de administración (https://192.168.137.1:9443/carbon/) que corresponde a la dirección web de la interfaz de administración de WSO2 EI.

Al colocar la URL en el navegador se aprecia la página de autenticación de WSO2 EI como se muestra en la Figura 6 Página de autenticación de WSO2 EI. Inicialmente se autentica utilizando el usuario *admin* y la contraseña *admin* que corresponde al usuario súper administrador del sistema.

Figure 5 shows the list of events after executing the integrator.bat file on Windows. Each of the Log type operations that are executed over time on the system appear in the console. The image shows in the last line the path of the administration panel (https://192.168.137.1:9443/carbon/) that corresponds to the web address of the WSO2 EI administration interface.

When placing the URL in the browser, the WSO2 EI authentication page can be seen as shown in Figure 6. Initially it is authenticated using the *admin* user and the *admin* password corresponding to the super administrator user of the system.

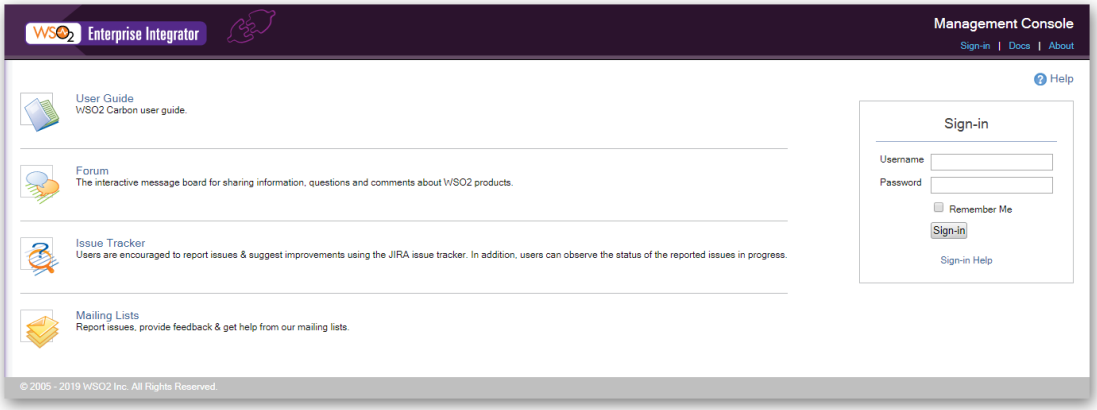

***Figura** 6 Página de autenticación de WSO2 EI*

***Figure 6** WSO2 EI Authentication Page*

Una vez autenticado en el sistema se muestra la interfaz de administración de WSO2 EI. La Figura 7 Interfaz de administración de WSO2 EI muestra la pantalla general de administración del sistema donde se pueden apreciar los datos generales del servidor donde se encuentra desplegado WSO2. Además de ello se pueden seleccionar cada una de las funcionalidades que contiene WSO2 EI. En la sección de configuración se pueden administrar los usuarios, así como los roles en el sistema.

Once authenticated in the system, the WSO2 EI administration interface is displayed. Figure 7 shows the general system administration screen where you can see the general data of the server where WSO2 is deployed. In addition, each of the functionalities that WSO2 EI contains can be selected. In the configuration section you can manage the users, as well as the roles in the system.

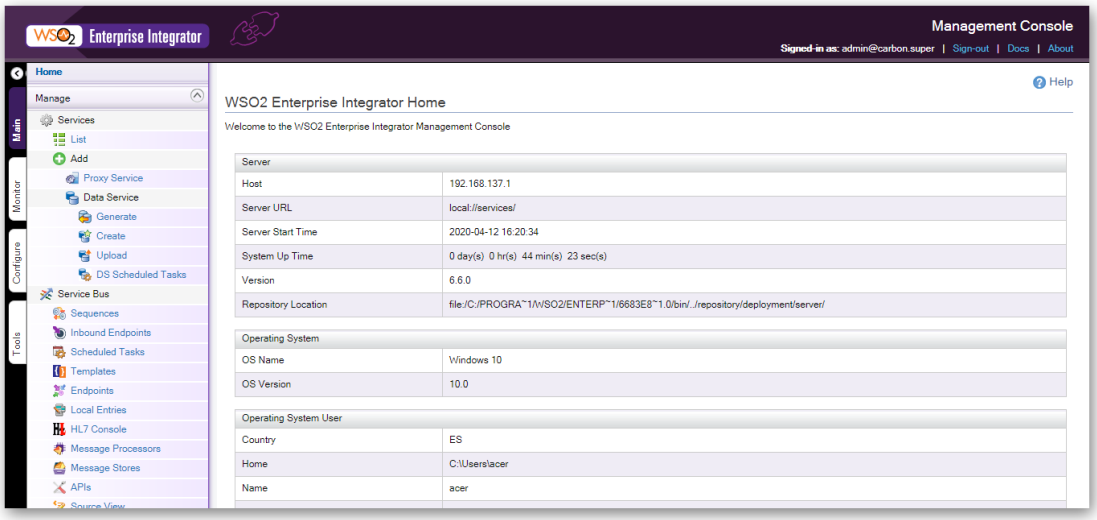

***Figura 7** Interfaz de administración de WSO2 EI*

***Figure** 7 WSO2 EI Management Interface*

Configuración de WSO2 EI

El punto clave de la presente investigación se centra en el apartado APIs, seleccionable entre las opciones a la izquierda. Se configurarán cada una de las API

WSO2 EI Configuration

The key point of this research is focused on the APIs section, selectable from the options on the left. Each

REST que controlará WSO2 EI. Una vez en el apartado APIs (como se muestra en la Figura 8 Ventana de API desplegadas de WSO2 EI) se pueden observar las APIs creadas y en funcionamiento, así como las operaciones realizables sobre las mismas y la creación de nuevas.

Se muestran el nombre y la URL de la API (context de la API, raíz de las URL de los Resources). Asimismo, se muestran las operaciones que pueden ser realizadas sobre la API, como Editar y Probar API. Este último permitirá mediante una interfaz intuitiva probar cada uno de los Resources de la API.

of the REST APIs that WSO2 EI will control will be configured. Once in the APIs section (as shown in Figure 8) you can see the APIs created and in operation, as well as the operations that can be carried out on them and the creation of new ones.

The name and URL of the API (API context, root of Resources URLs) are displayed. Likewise, the operations that can be performed on the API are shown, such as Edit and Test API. The latter will allow an intuitive interface to test each of the API Resources.

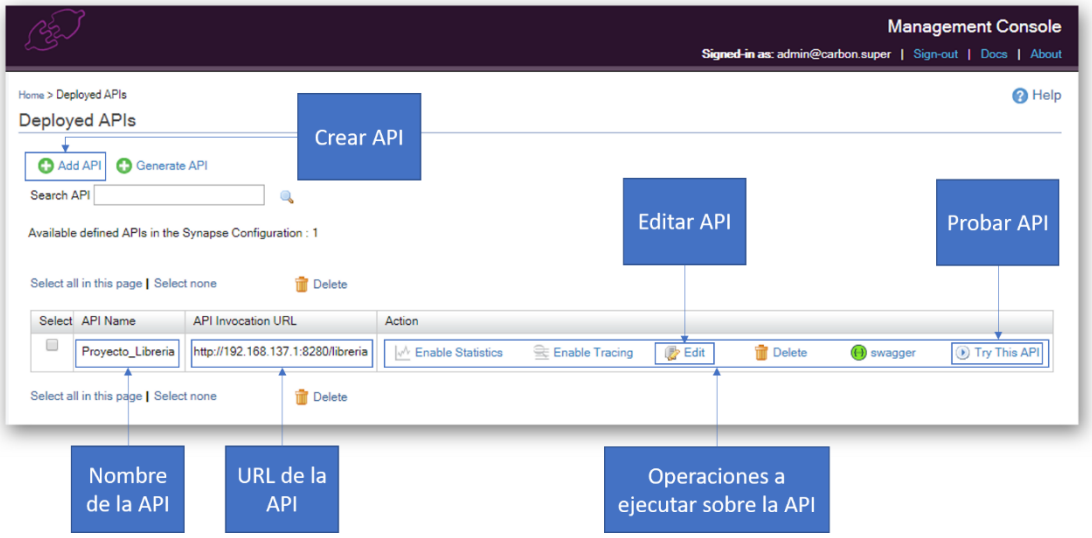

Figura 8 *Ventana de API desplegadas de WSO2 EI*

Figure 8 *WSO2 EI API deployed window*

Para crear una nueva API se selecciona Crear API, que muestra un formulario con los datos necesarios para realizar la operación (mostrado en la Figura 9 Crear API en WSO2 EI). Dicho formulario contiene los datos generales de un API, se debe añadir el nombre que tendrá (el identificador con que será reconocido en el sistema), el Context, que corresponde a la ruta por la que responderá la API, o sea, si el IP donde se encuentra instalado WSO2 EI es 192.168.137.1 la URL de acceso a la API sería https://192.168.137.1:8280/<context>/<resource>. Donde Resource corresponde a cada uno de los recursos que serán configurados. El resto de las configuraciones del formulario no son necesarias.

To create a new API, select Create API, which displays a form with the necessary data to perform the operation (shown in Figure 9). This form contains the general data of an API, you must add the name that it will have (the identifier with which it will be recognized in the system), the Context, which corresponds to the route through which the API will respond, that is, if the IP where WSO2 is installed EI is 192.168.137.1 the API access URL would be https://192.168.137.1:8280/<context>/ <resource>. Where Resource corresponds to each of the resources that will be configured. The rest of the form settings are not necessary.

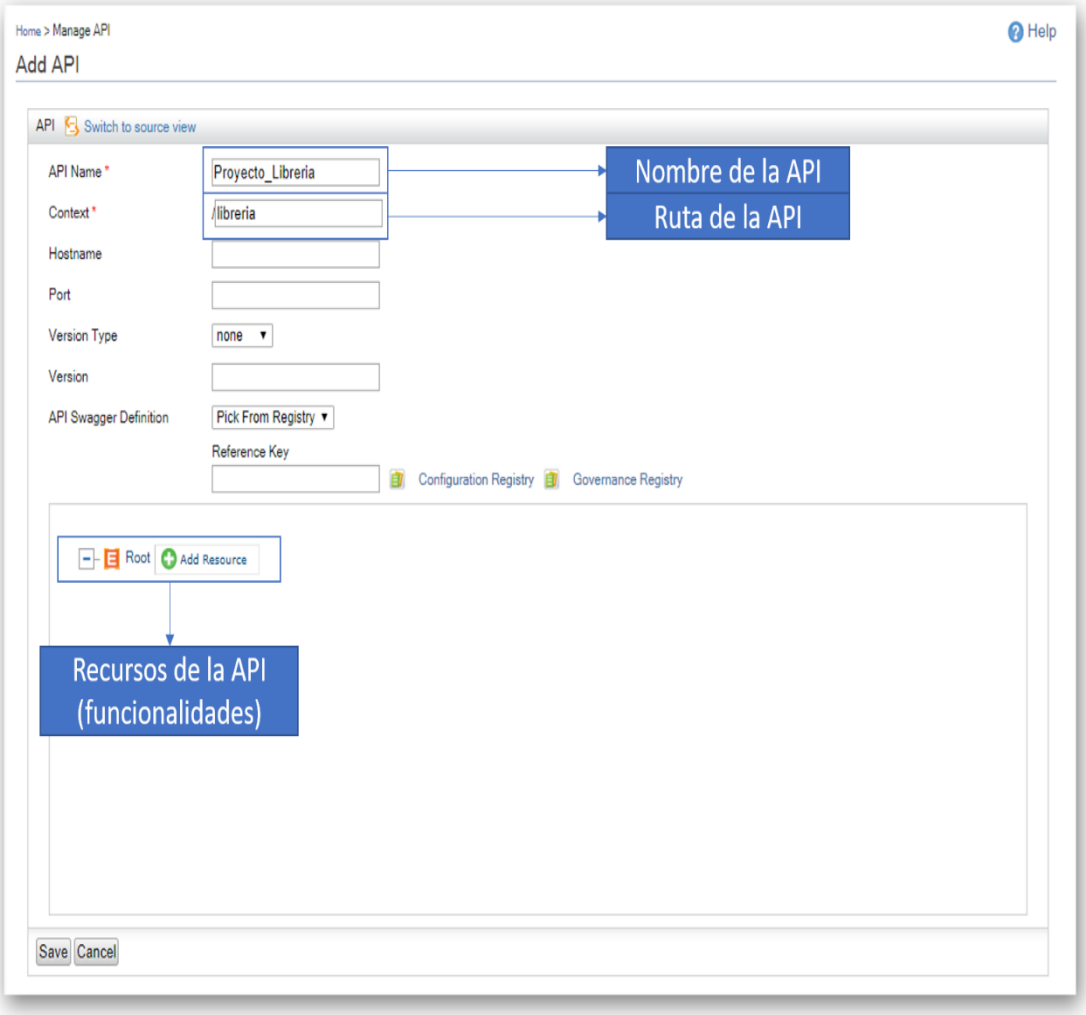

Figura 9 *Crear API en WSO2 EI*

Figure 9 *Create API in WSO2 EI*

Una vez rellenado el formulario se ha configurado de forma básica la API, se tiene la ruta de acceso y su identificador, pero con solo esto la API no realiza ninguna función. Las funcionalidades de la API se configuran en el apartado Resources de API. Para añadir un nuevo Resource se selecciona la opción Add Resource como se muestra en la Figura 9 Crear API en WSO2 EI. Automáticamente aparece un nuevo formulario en la parte inferior de la página donde será configurado (como muestra la Figura 10 Configuración de un Resource). Es necesario destacar que puede crearse una API para cada proyecto de SpringBoot y configurar un recurso por cada API REST. Una API puede tener varios Resources.

Once the form has been filled in, the API has been configured in a basic way, we have the path and its identifier, but with this alone the API does not perform any function. The API functionalities are configured in the API Resources section. To add a new Resource, select the Add Resource option as shown in Figure 9. A new form automatically appears at the bottom of the page where it will be configured (as shown in Figure 10). It should be noted that an API can be created for each SpringBoot project and a resource can be configured for each REST API. An API can have multiple Resources.

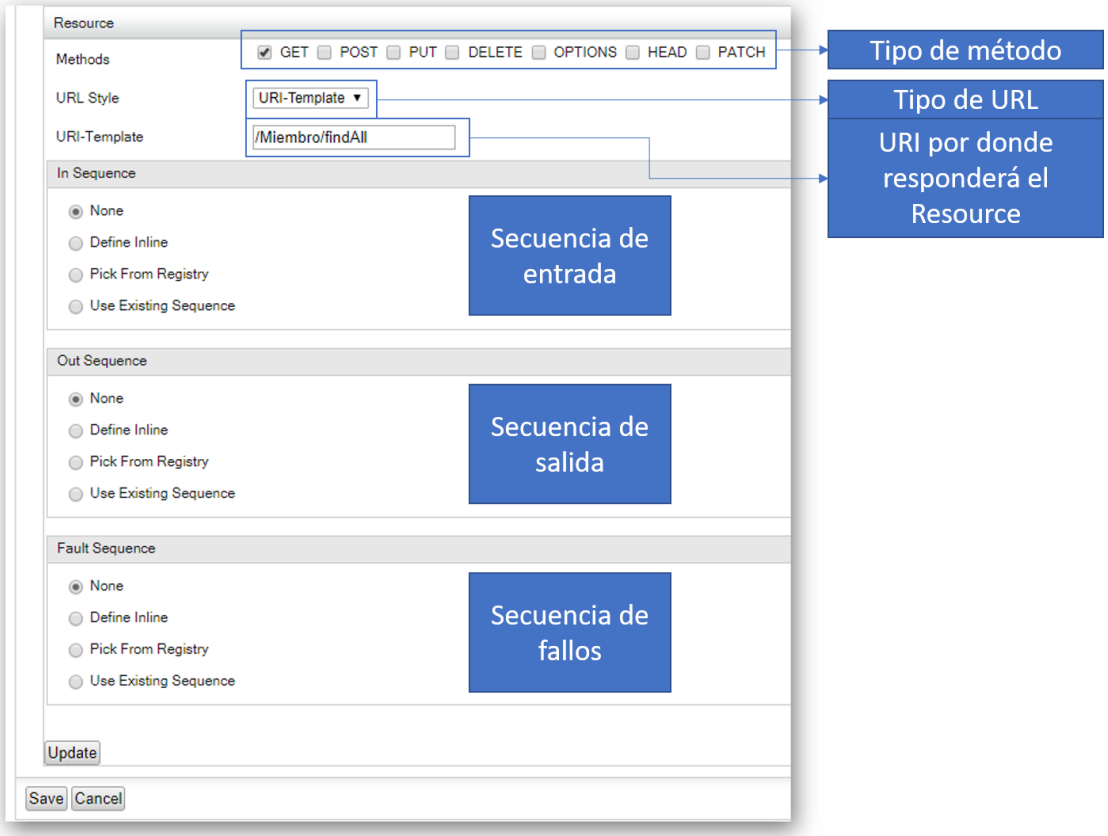

Figura 10 *Configuración de un Resource*

Figure 10 *Configuration of a Resource*

Inicialmente se selecciona el/los métodos que va a aceptar el recurso (en el caso de prueba se ha seleccionado un método por cada Resource, que coincide con su homólogo API REST del proyecto SpringBoot). Luego se selecciona el estilo de la URL, que se marca la opción URI-Template (Plantilla URI). Posteriormente se escribe el estilo de la URI-Template, que corresponde al final de la ruta URL del Resource que irá a continuación del context (en este caso se ha decidido mantener las rutas de las API REST de SpringBoot para mantener la misma lógica), pero no por ello indica que no se puedan utilizar en una misma ruta varios métodos (en ese caso el tratamiento de la entrada de datos sería diferente).

Se ha definido la URL y el método de conexión para el Resource, para definir el funcionamiento del recurso se definen las Sequence (Secuencia). A pesar de existir tres configuraciones posibles (In Sequence, Out Sequence y Fault Sequence), para la presente investigación solo es necesaria configurar In Sequence. Se define el comportamiento de los datos de entrada al Resource (se configurará el HTTP Endpoint de Spring Boot al que se realizará la conexión siguiendo la lógica de la Figura 4

Initially, the method (s) that the resource will accept is selected (in the test case, a method has been selected for each Resource, which matches its REST API counterpart from the SpringBoot project). Then the style of the URL is selected, the option URI-Template is checked. Subsequently, the style of the URI-Template is written, which corresponds to the end of the URL path of the Resource that will follow the context (in this case it has been decided to keep the routes of the SpringBoot REST APIs to maintain the same logic), but this does not mean that several methods cannot be used in the same route (in that case the data entry treatment would be different).

The URL and the connection method for the Resource have been defined, to define the operation of the resource, the Sequence (Sequence) are defined. Although there are three possible configurations (In Sequence, Out Sequence and Fault Sequence), for the present investigation it is only necessary to configure In Sequence. The behavior of the input data to the Resource is defined (the Spring Boot HTTP Endpoint to which the connection will be made will be

Conexiones de los proyectos utilizando WSO2 EI). En caso de ser necesario se puede configurar la Out Sequence si se necesitara hacer un procesamiento de datos o si la salida llevara un tratamiento especial.

Para definir la In Sequence en el grupo de radio buttons correspondiente se selecciona Define Inline (se definirá un comportamiento nuevo para cada Resource), al seleccionarlo aparece un nuevo botón a la derecha de la opción llamado Create, como se muestra en Figura 11 Sección In Sequence al seleccionar Define Inline.

configured following the logic in Figure 4). If necessary, the Out Sequence can be configured if data processing is required or if the output has a special treatment.

To define the In Sequence in the corresponding group of radio buttons, select Define Inline (a new behavior will be defined for each Resource), when selected, a new button appears to the right of the option called Create, as shown in Figure 11.

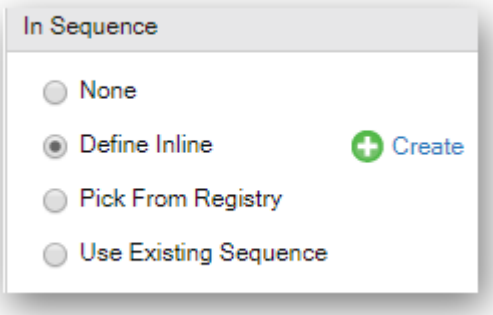

Figura 11 *Sección In Sequence al seleccionar Define Inline*

Figure 11 *In Sequence section when selecting Define Inline*

Al seleccionar el botón Create aparece una nueva ventana en la que se podrá definir el comportamiento de entrada del Context seleccionado de la API recién creada. La Figura 12 Creación del Resource de una API (In Sequence) muestra el entorno en el que se pueden añadir nuevas funcionalidades al Resource en la In Sequence al definirla Inline.

By selecting the Create button, a new window appears in which you can define the input behavior of the selected Context of the newly created API. Figure 12 shows the environment in which new functionalities can be added to the Resource in the In Sequence by defining it Inline.

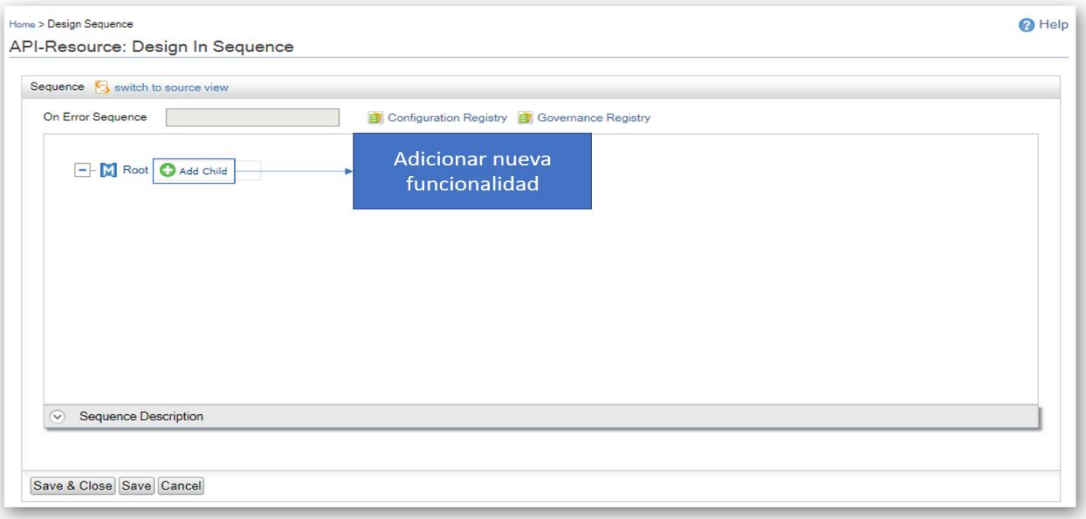

Figura 12 Creación del Resource de una API (In Sequence)

Figure 12 *Resource creation of an API (In Sequence)*

Al seleccionar el botón Add Child aparece un menú contextual que contiene los diferentes grupos de funcionalidades disponibles. Al seleccionar sobre cada uno aparecen los subgrupos. La Figura 13 Menú contextual de una Sequence muestra el menú contextual con las opciones. Para el presente trabajo fue necesario utilizar los componentes del grupo Core (Log y Send).

Selecting the Add Child button brings up a contextual menu containing the different groups of available functionalities. When selecting on each one the subgroups appear. Figure 13 shows the context menu with the options. For the present work it was necessary to use the components of the Core group (Log and Send).

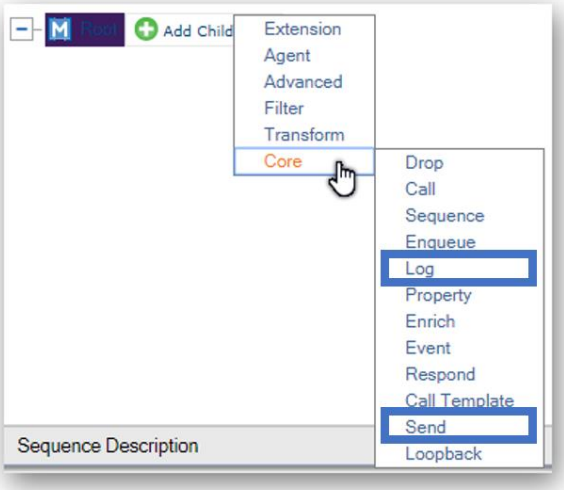

Figura 13 *Menú contextual de una Sequence*

Figure 13 *Contextual menu of a Sequence*

El componente Log no resulta imprescindible, pero se puede utilizar para monitorizar en la consola de WSO2 los resultados de las conexiones de los Resources, o sea, si la conexión devolvió una excepción, que la administración conozca la razón del problema o si la conexión fue satisfactoria. Al seleccionarlo en el menú contextual aparece un nuevo elemento Log en el árbol de la Sequence (Figura 14 Sequence al adicionar un elemento Log) y un formulario en la parte inferior de la página para configurarlo (Figura 15 Formulario de configuración de un elemento Log).

The Log component is not essential, but it can be used to monitor in the WSO2 console the results of the Resources connections, that is, if the connection returned an exception, that the administration knows the reason for the problem or if the connection was satisfactory. When selected in the contextual menu, a new Log element appears in the Sequence tree (Figure 14) and a form at the bottom of the page to configure it (Figure 15).

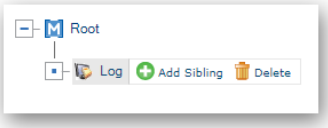

Figura 14 *Sequence al adicionar un elemento Log*

Figure 14 *Sequence when adding a Log element*

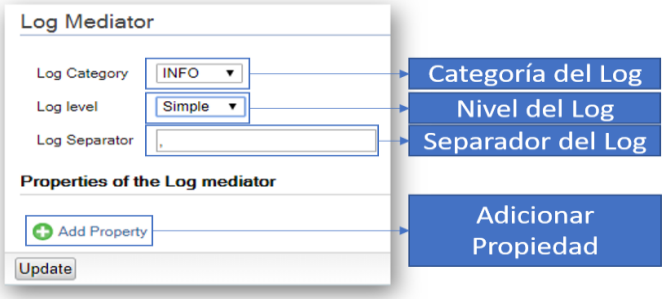

Figura 15 *Formulario de configuración de un elemento Log*

Figure 15 *Log element configuration form*

La Figura 15 Formulario de configuración de un elemento Log muestra la configuración básica (por defecto) que tiene un elemento Log. En caso de necesitarse más información para la revisión de errores en la consola de WSO2 pueden configurarse parámetros (configuración FULL) como muestra la Figura 16 Configuración FULL de un elemento Log. Para configurar un Log FULL se adicionan tres parámetros, el primero muestra un mensaje indicando que se realizará una revisión de fallos (no es imprescindible, pero permite que sea más sencillo de visualizar en la consola). Luego se añaden dos expresiones: *get-property('ERROR_CODE')* que mostrará el código de error obtenido y *get-property('ERROR_MESSAGE')* que mostrará el mensaje de error obtenido. Utilizar esta configuración mostrará con más detalle el resultado de las conexiones de los Resources, puede ser utilizado en todo tipo de conexiones.

Figure 15 shows the basic (default) configuration that a Log element has. If more information is needed for the error review in the WSO2 console, parameters can be configured (FULL configuration) as shown in Figure 16. To configure a FULL Log, three parameters are added, the first one shows a message indicating that a review will be performed of failures (it is not essential, but it makes it easier to view on the console). Then two expressions are added: get-property ('ERROR_CODE') which will show the obtained error code and get-property ('ERROR_MESSAGE') which will show the obtained error message. Using this configuration will show in more detail the result of the Resources connections, it can be used in all types of connections.

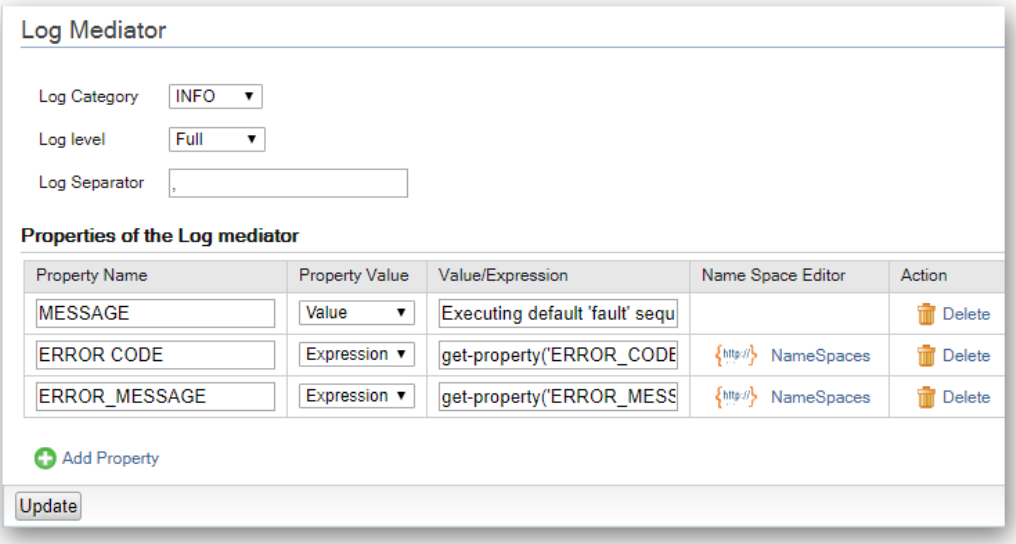

Figura 16 *Configuración FULL de un elemento Log*

Figure 16 *FULL configuration of a Log element*

El otro componente de In Sequence que se utilizó en este trabajo es el componente Send, que resulta imprescindible. Este componente realizará la conexión con la API REST de los proyectos SpringBoot. La Figura 18 Opciones de configuración de un componente Send muestra las opciones de configuración de un componente Send. En este formulario se configurará el Endpoint (configuración de la API REST donde se conectará el servicio) definiéndose la URL y el método de conexión. Existen otras configuraciones como el tipo de Sequence y la posibilidad de crear el mensaje antes de enviar, que no son necesarios de configurar para el presente trabajo y quedan configurados por defecto. Se seleccionará en el grupo de radio buttons la opción Define Inline, para configurar un Endpoint específico para ese Resource.

The other component of In Sequence that was used in this work is the Send component, which is essential. This component will make the connection with the REST API of the SpringBoot projects. Figure 18 shows the configuration options for a Send component. In this form the Endpoint will be configured (configuration of the REST API where the service will connect), defining the URL and the connection method. There are other configurations such as the type of Sequence and the possibility of creating the message before sending, which are not necessary to configure for this job and are configured by default. The Define Inline option will be selected in the group of radio buttons to configure a specific Endpoint for that Resource.

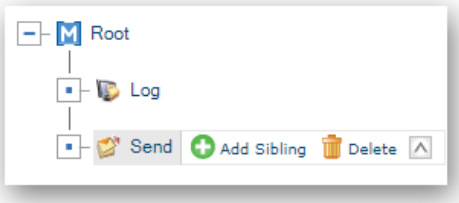

Figura 17 *Sequence al adicionar un elemento Send*

Figure 17 *Sequence when adding a Send element*

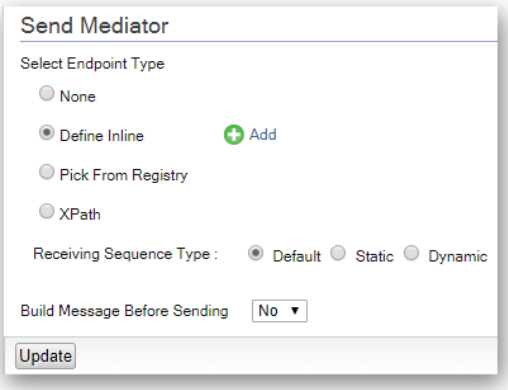

Figura 18 *Opciones de configuración de un componente Send*

Figure 18 *Configuration options of a Send component*

Al seleccionar la opción Add del formulario aparece una nueva ventana, como se muestra en la Figura 20 Configuración de un Endpoint donde se seleccionará el tipo de Endpoint a utilizar. En este caso se utilizará HTTP Endpoint.

When selecting the Add option of the form, a new window appears, as shown in Figure 19, where the type of Endpoint to be used will be selected. In this case, HTTP Endpoint will be used.

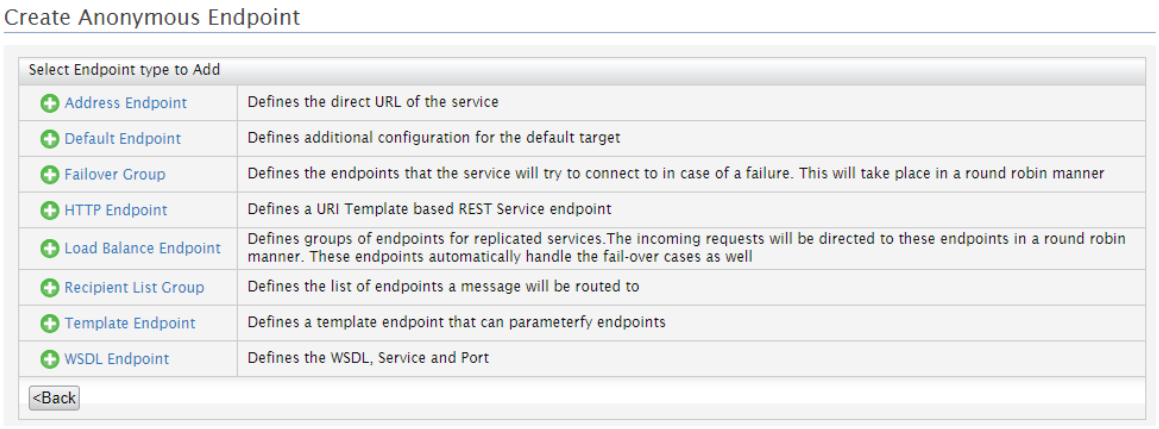

Figura 19 *Crear nuevo Endpoint*

Figure 19 *Create new endpoint*

Al seleccionarlo aparece un nuevo formulario que contiene la configuración del Endpoint (como muestra la Figura 20 Configuración de un Endpoint), se rellenará la configuración de la conexión con la API REST. Aparece un campo llamado URI Template donde se coloca la URL, a su lado aparece un botón llamado Test que permitirá comprobar la conectividad de WSO2 EI con la API REST. Otro elemento importante del formulario es el HTTP Method donde se seleccionará el tipo de método a utilizar en la conexión, en este caso coincide con el método seleccionado en la configuración del Resource en la Figura 9 Crear API en WSO2 EI. En caso de que sea necesario pueden adicionarse propiedades utilizando el botón Add Property.

When selected, a new form appears containing the configuration of the Endpoint (as shown in Figure 20), the configuration of the connection with the REST API will be filled. A field called URI Template appears where the URL is placed, next to it there is a button called Test that will allow you to check the connectivity of WSO2 EI with the REST API. Another important element of the form is the HTTP Method where the type of method to be used in the connection will be selected, in this case it coincides with the method selected in the Resource configuration in Figure 9. If necessary, properties can be added using the Add Property button.

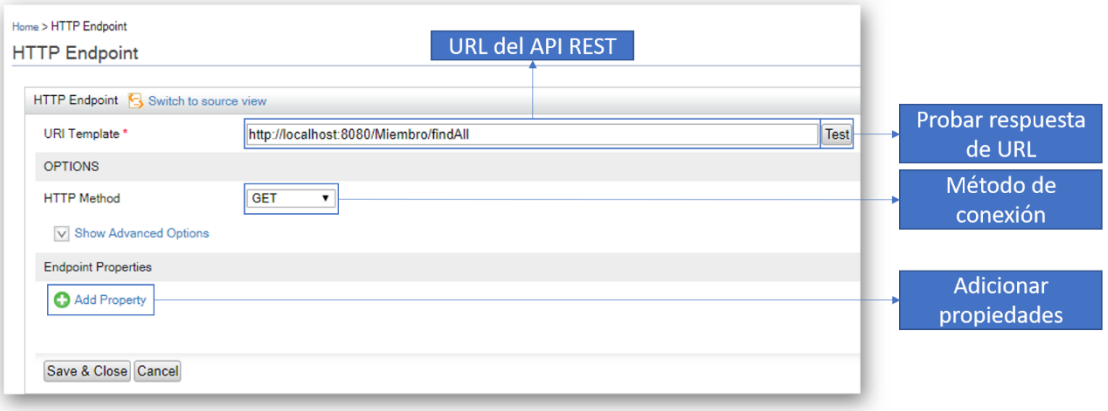

Figura 20 *Configuración de un Endpoint*

Figure 20 *Endpoint configuration*

Una vez configurado el HTTP Endpoint se tienen todos los elementos necesarios para el primer Resource, solo resta seleccionar el botón Save & Close para regresar a la configuración del In Sequence, luego

Once the HTTP Endpoint is configured, you have all the necessary elements for the first Resource, it only remains to select the Save & Close button to return to the In Sequence configuration, then click the Update

se da clic en el botón Update para actualizar la configuración del componente Send. Posteriormente se selecciona Save & Close para guardar los cambios y regresar a la configuración del Resource. Al dar clic en la opción Update y posteriormente en Save & Close guardará todos los cambios en la configuración de la API. Se tiene entonces una API de WSO2 que se conectará a un servidor que brinda servicios API REST donde se realiza la búsqueda findAll de una tabla de una base de datos.

Configuración de conexiones PUT, POST y CREATE

Para completar el ejemplo anterior son necesarios otros Resources para cada una de las API REST brindadas en la **¡Error! No se encuentra el origen de la referencia.**. En el caso de las API REST de tipo POST (Create) y PUT (Update) no es necesario adicionar configuraciones extras para pasar el JSON por parámetros. WSO2 EI comprende de forma automática que se enviará un fichero JSON y lo envía hacia el servidor API REST. Es necesario tener en cuenta que para que este sistema funcione correctamente con la configuración actual se debe seleccionar en el tipo de método al configurar el Resource (Figura 10 Configuración de un Resource) el mismo método que será configurado en el Endpoint (Figura 20 Configuración de un Endpoint).

Configuración de rutas con parámetros (el caso de findByID y DELETE)

Las API REST GET (findByID) y DELETE (delete) necesitan de un tratamiento diferente pues requieren del paso de un parámetro por la URL que corresponde al elemento en la búsqueda. Para ello es necesario realizar varios ajustes a la configuración antes descrita.

Para realizar el findByID inicialmente se debe configurar en el URI-Template del Resource el paso del identificador (id) por parámetros que será buscado al conectarse al API REST en la URL. Para ello se indica entre llaves ({ }) una variable, que almacenará el id del elemento a buscar, en este caso sería la variable id. La Figura 21 Configuración de un Resource con paso de parámetros por URI muestra la configuración del Resource con parámetros por URI. Se puede apreciar cómo en la URI-Template se tiene la variable id entre llaves, ello permite que al introducir la URL completa se reconozca lo

button to update the Send component configuration. Subsequently, Save & Close is selected to save the changes and return to the Resource configuration. Clicking on the Update option and then on Save & Close will save all the changes in the API configuration. There is then a WSO2 API that will connect to a server that provides REST API services where a findAll search of a database table is performed.

Setting up PUT, POST and CREATE connections

To complete the previous example, other Resources are necessary for each of the REST APIs provided in Table 1. In the case of the POST (Create) and PUT (Update) type REST APIs, it is not necessary to add extra configurations to pass the JSON by parameters. WSO2 EI automatically understands that a JSON file will be sent and sends it to the REST API server. It is necessary to bear in mind that for this system to work correctly with the current configuration, the method type must be selected when configuring the Resource (Figure 10) the same method that will be configured in the Endpoint (Figure 20).

Configuration of routes with parameters (the case of findByID and DELETE)

The GET (findByID) and DELETE (delete) REST APIs need a different treatment as they require the passing of a parameter through the URL that corresponds to the element in the search. This requires making various adjustments to the configuration described above.

To carry out the findByID initially, the passage of the identifier (id) by parameters must be configured in the URI-Template of the Resource that will be searched for when connecting to the REST API in the URL. To do this, a variable is indicated between braces ({}), which will store the id of the element to search, in this case it would be the id variable. Figure 21 shows the Resource configuration with parameters by URI. You can see how in the URI-Template you have the variable id between braces, this allows that when you enter the complete URL, what is entered after the "/" of find is recognized as the identifier to search.

introducido después del "/" de find como el identificador a buscar.

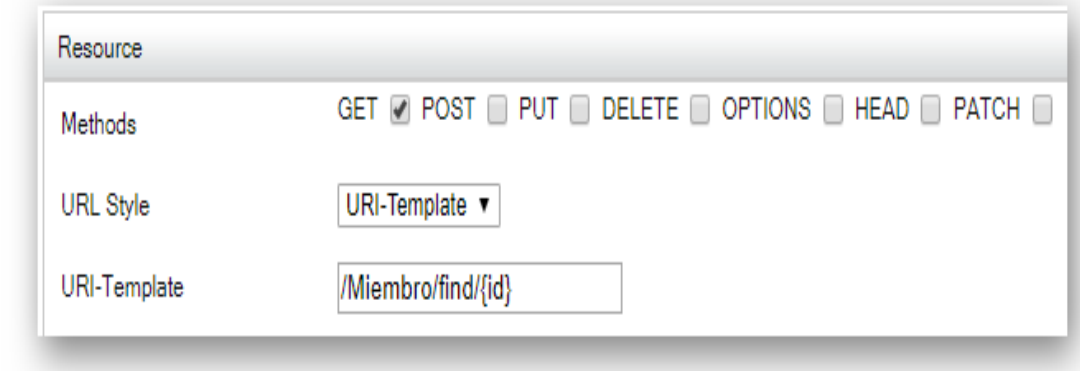

Figura 21 *Configuración de un Resource con paso de parámetros por URI*

Figure 21 *Configuration of a Resource with parameter passing by URI*

Otro cambio necesario a la configuración anterior descrita es la configuración del HTTP Endpoint en el componente Send del In Sequence. Será necesario configurar en la URI-Template del HTTP Endpoint la configuración de la variable id (capturada anteriormente en la configuración del Resource) para que sea pasada como parámetros en la conexión al API REST. Para ello se le añade entre llaves ({ }) uri.var.id que hace referencia a la variable id que fue pasada por parámetros en el Resource (como se muestra en la Figura 22 Configuración del HTTP Endpoint de un findByID (paso de parámetros por URL)). Debe tenerse en cuenta que debe mantenerse el mismo nombre de variable en el Resource y en el HTTP Endpoint (en este caso id).

Another necessary change to the previous configuration described is the configuration of the HTTP Endpoint in the Send component of the In Sequence. It will be necessary to configure in the URI-Template of the HTTP Endpoint the configuration of the id variable (previously captured in the Resource configuration) so that it is passed as parameters in the connection to the REST API. To do this, uri.var.id is added between braces ({}) that refers to the id variable that was passed by parameters in the Resource (as shown in Figure 22). It should be noted that the same variable name must be kept in the Resource and in the HTTP Endpoint (in this case id).

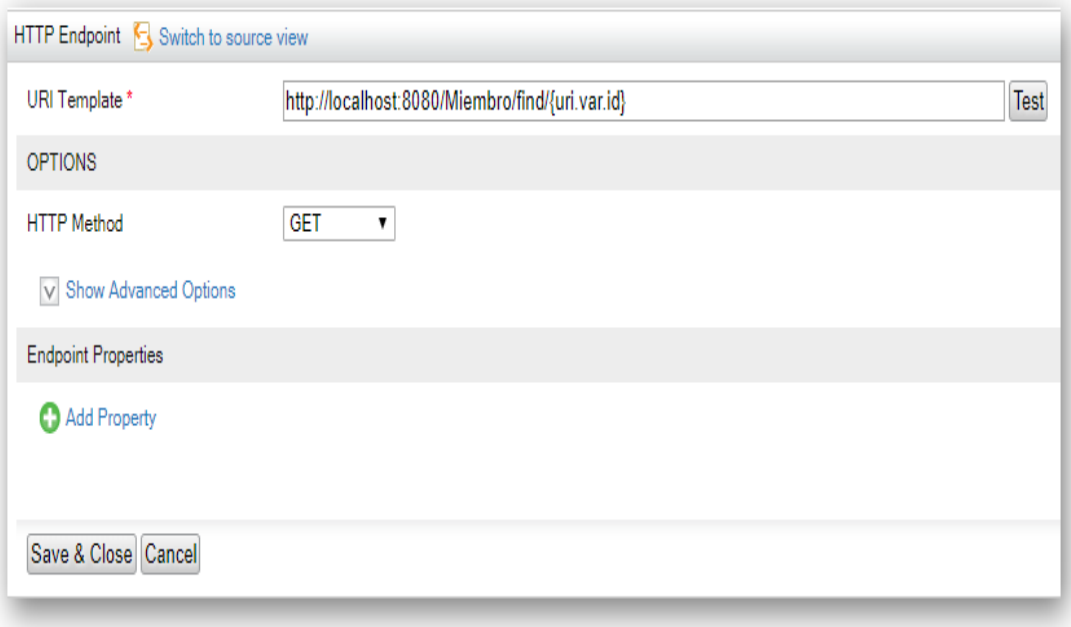

Figura 22 *Configuración del HTTP Endpoint de un findByID (paso de parámetros por URL)*

Figure 22 *HTTP Endpoint configuration of a findByID (passing parameters by URL)*

En el caso del método DELTE se realiza la misma configuración, sólo es necesario configurar el método de conexión pertinente (DELETE) en el Resource y en el HTTP Endpoint.

Lo anterior descrito puede extrapolarse para varios proyectos, lo cual traería como resultado la obtención de un servidor que centraliza varios servicios de otros servidores. La Figura 23 Varias API configuradas en WSO2 EI muestra varias API que han sido configuradas y desplegadas en un servidor WSO2 EI.

In the case of the DELTE method, the same configuration is carried out, it is only necessary to configure the relevant connection method (DELETE) in the Resource and in the HTTP Endpoint.

The above described can be extrapolated to various projects, which would result in obtaining a server that centralizes various services from other servers. Figure 23 shows several APIs that have been configured and deployed on a WSO2 EI server.

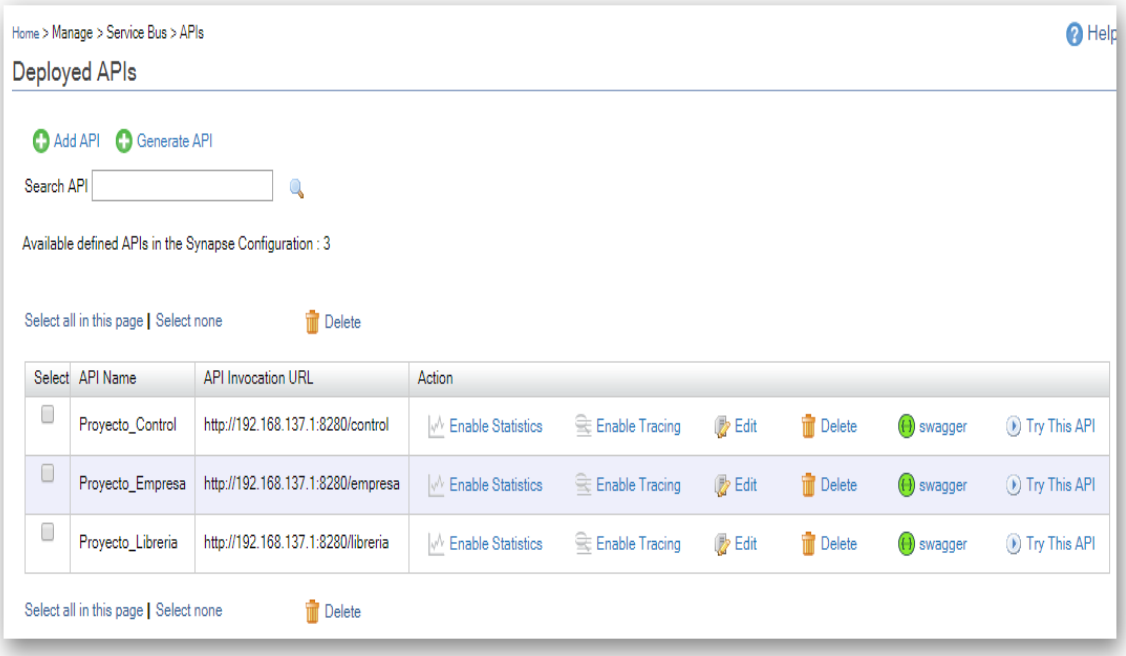

Figura 23 *Varias API configuradas en WSO2 EI*

Figure 23 *Various APIs configured in WSO2 EI*

Configuración en forma de código

WSO2 no sólo permite realizar las configuraciones de sus componentes de forma visual, tiene una vista de código simultánea que permite su edición. Cada configuración de un componente es traducida a un fichero XML que puede ser modificado por el usuario desde la propia página web. En cada una de las vistas anteriormente descritas (Resources, In Sequence y HTTP Endpoint) se puede apreciar un enlace llamado "Switch to source view" (cambiar a la vista de código). A veces resulta más simple para los administradores del sistema realizar las configuraciones desde esta vista debido a que se puede configurar desde la vista de la API los Resource, In Sequence y los HTTP Endpoint; por lo que resulta más simple de configurar una API si los Resources mantienen una estructura similar. Para regresar a la vista de diseño se selecciona el enlace "Switch to design view" (cambiar a vista de diseño). El código de la API Librería una vez finalizada quedaría así.

Configuration in code form

WSO2 not only allows you to make the configurations of its components visually, it has a simultaneous code view that allows its editing. Each configuration of a component is translated into an XML file that can be modified by the user from the web page itself. In each of the previously described views (Resources, In Sequence and HTTP Endpoint) you can see a link called "Switch to source view". Sometimes it is simpler for system administrators to configure settings from this view because Resource, In Sequence and HTTP Endpoints can be configured from the API view; so it is simpler to configure an API if the Resources maintain a similar structure. To return to the design view, select the link "Switch to design view" (switch to design view). The code of the Library API once finished would look like this.

```
<api xmlns="http://ws.apache.org/ns/synapse" name="Proyecto_Libreria"
context="/libreria">
   <resource methods="GET" uri-template="/Miembro/findAll">
      <inSequence>
         <log/>
         <send>
            <endpoint>
               <http method="GET"
               uri-template="http://localhost:8080/Miembro/findAll"/>
            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="GET" uri-template="/Miembro/find/{id}">
      <inSequence>
         <log/>
         <send>
            <endpoint>
               <http method="GET"
               uri-template="http://localhost:8080/Miembro/find/{uri.var.id}"/>

            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="POST" uri-template="/Miembro/create">
      <inSequence>
         <log level="full">
            <property name="MESSAGE" value="Executing default 'fault'
            sequence"/>
            <property name="ERROR CODE"
            expression="get-property('ERROR_CODE')"/>
            <property name="ERROR_MESSAGE"
            expression="get-property('ERROR_MESSAGE')"/>
         </log>
         <send>
            <endpoint>
               <http method="POST"
               uri-template="http://localhost:8080/Miembro/create"/>
            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="PUT" uri-template="/Miembro/update">
      <inSequence>
         <log level="full">
            <property name="MESSAGE" value="Executing default 'fault'
            sequence"/>
            <property name="ERROR CODE"
            expression="get-property('ERROR_CODE')"/>
            <property name="ERROR_MESSAGE"
            expression="get-property('ERROR_MESSAGE')"/>
         </log>
         <send>
            <endpoint>
               <http method="PUT"
               uri-template="http://localhost:8080/Miembro/update"/>
            </endpoint>
         </send>
      </inSequence>
```

```
</resource>
<resource methods="DELETE" uri-template="/Miembro/delete/{id}">
   <inSequence>
      <log/>
      <send>
         <endpoint>
            <http method="DELETE"
            uri-template="http://localhost:8080/Miembro/delete/{uri.var.id}"/>

         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="GET" uri-template="/Producto/findAll">
   <inSequence>
      <log/>
      <send>
         <endpoint>
            <http method="GET"
            uri-template="http://localhost:8080/Producto/findAll"/>
         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="GET" uri-template="/Producto/find/{id}">
   <inSequence>
      <log/>
      <send>
         <endpoint>
            <http method="GET"
            uri-template="http://localhost:8080/Producto/find/{uri.var.id}"/>

         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="POST" uri-template="/Producto/create">
   <inSequence>
      <log level="full">
         <property name="MESSAGE" value="Executing default 'fault'
         sequence"/>
         <property name="ERROR CODE"
         expression="get-property('ERROR_CODE')"/>
         <property name="ERROR_MESSAGE"
         expression="get-property('ERROR_MESSAGE')"/>
      </log>
      <send>
         <endpoint>
            <http method="POST"
            uri-template="http://localhost:8080/Producto/create"/>
         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="PUT" uri-template="/Producto/update">
   <inSequence>
      <log level="full">
         <property name="MESSAGE" value="Executing default 'fault'
         sequence"/>
         <property name="ERROR CODE"
```

```
            expression="get-property('ERROR_CODE')"/>
            <property name="ERROR_MESSAGE"
            expression="get-property('ERROR_MESSAGE')"/>
         </log>
         <send>
            <endpoint>
               <http method="PUT"
               uri-template="http://localhost:8080/Producto/update"/>
            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="DELETE" uri-template="/Producto/delete/{id}">
      <inSequence>
         <log/>
         <send>
            <endpoint>
               <http method="DELETE"
               uri-template="http://localhost:8080/Producto/delete/{uri.var.id}"/>

            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="GET" uri-template="/Trabajador/findAll">
      <inSequence>
         <log/>
         <send>
            <endpoint>
               <http method="GET"
               uri-template="http://localhost:8080/Trabajador/findAll"/>

            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="GET" uri-template="/Trabajador/find/{id}">
      <inSequence>
         <log/>
         <send>
            <endpoint>
               <http method="GET"
               uri-template="http://localhost:8080/Trabajador/find/{uri.var.id}"/>

            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="POST" uri-template="/Trabajador/create">
      <inSequence>
         <log level="full">
            <property name="MESSAGE" value="Executing default 'fault'
            sequence"/>
            <property name="ERROR_CODE"
            expression="get-property('ERROR_CODE')"/>
            <property name="ERROR_MESSAGE"
            expression="get-property('ERROR_MESSAGE')"/>
         </log>
         <send>
            <endpoint>
```

```
            <http method="POST"
            uri-template="http://localhost:8080/Trabajador/create"/>

         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="PUT" uri-template="/Trabajador/update">
   <inSequence>
      <log level="full">
         <property name="MESSAGE" value="Executing default 'fault'
         sequence"/>
         <property name="ERROR CODE"
         expression="get-property('ERROR_CODE')"/>
         <property name="ERROR_MESSAGE"
         expression="get-property('ERROR_MESSAGE')"/>
      </log>
      <send>
         <endpoint>
            <http method="PUT"
            uri-template="http://localhost:8080/Trabajador/update"/>

         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="DELETE" uri-template="/Trabajador/delete/{id}">
   <inSequence>
      <log/>
      <send>
         <endpoint>
            <http method="DELETE"
            uri-template="http://localhost:8080/Trabajador/delete/{uri.var.id}"
            />
         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="GET" uri-template="/Transaccion/finAll">
   <inSequence>
      <log/>
      <send>
         <endpoint>
            <http method="GET"
            uri-template="http://localhost:8080/Transaccion/findAll"/>

         </endpoint>
      </send>
   </inSequence>
</resource>
<resource methods="GET" uri-template="/Transaccion/find/{id}">
   <inSequence>
      <log/>
      <send>
         <endpoint>
            <http method="GET"
            uri-template="http://localhost:8080/Transaccion/find/{uri.var.id}"/
            >
         </endpoint>
      </send>
```

```
      </inSequence>
   </resource>
   <resource methods="POST" uri-template="/Transaccion/create">
      <inSequence>
         <log level="full">
            <property name="MESSAGE" value="Executing default 'fault'
            sequence"/>
            <property name="ERROR CODE"
            expression="get-property('ERROR_CODE')"/>
            <property name="ERROR_MESSAGE"
            expression="get-property('ERROR_MESSAGE')"/>
         </log>
         <send>
            <endpoint>
               <http method="POST"
               uri-template="http://localhost:8080/Transaccion/create"/>

            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="PUT" uri-template="/Transaccion/update">
      <inSequence>
         <log level="full">
            <property name="MESSAGE" value="Executing default 'fault'
            sequence"/>
            <property name="ERROR CODE"
            expression="get-property('ERROR_CODE')"/>
            <property name="ERROR_MESSAGE"
            expression="get-property('ERROR_MESSAGE')"/>
         </log>
         <send>
            <endpoint>
               <http method="PUT"
               uri-template="http://localhost:8080/Transaccion/update"/>

            </endpoint>
         </send>
      </inSequence>
   </resource>
   <resource methods="DELETE" uri-template="/Transaccion/delete/{id}">
      <inSequence>
         <log/>
         <send>
            <endpoint>
               <http method="DELETE"
               uri-template="http://localhost:8080/Transaccion/delete/{uri.var.id}
               "/>
            </endpoint>
         </send>
      </inSequence>
   </resource>
</api>
```

Prueba de configuración

Existen varias maneras de probar la configuración de las API anteriormente descrita. WSO2 tiene incorporado un sistema de probado de API basado en Swagger[10] bastante intuitivo y fácil de utilizar. Por otra parte, puede utilizarse un framework web o una aplicación de terceros. En este caso se decidió utilizar Swagger y Postman para comprobarlas. La Tabla 4 Direcciones URL del Proyecto Libreria en SpringBoot y WSO2 muestra las direcciones URL de los API REST procedentes de SpringBoot y su homólogo en WSO2, cada uno de estos enlaces pueden ser probados.

Configuration test

There are several ways to test the API settings described above. WSO2 has built-in a fairly intuitive and easy-to-use Swagger-based API testing system. On the other hand, a web framework or a third-party application can be used. In this case it was decided to use Swagger and Postman to check them. Table 4 shows the URLs of the REST APIs coming from SpringBoot and its counterpart in WSO2, each of these links can be tested.

Tabla 4 *Direcciones URL del Proyecto Libreria en SpringBoot y WSO2*

Table 4 *URLs of the Library Project in SpringBoot and WSO2*

Método	URL SpringBoot	URL WSO2	Parámetro
GET	http:// 192.168.137.1:8080/ Miembro/findAll	https://192.168.137.1:8243/ libreria/Miembro/findAll	-
GET	http:// 192.168.137.1:8080/ Miembro/find/{id}	https://192.168.137.1:8243/ libreria/Miembro/find/{id}	URI
POST	http:// 192.168.137.1:8080/ Miembro/create	https://192.168.137.1:8243/ libreria/Miembro/create	JSON
PUT	http:// 192.168.137.1:8080/ Miembro/update/	https://192.168.137.1:8243/ libreria/Miembro/update	JSON
DELETE	http:// 192.168.137.1:8080/ Miembro/delete/{id}	https://192.168.137.1:8243/ libreria/Miembro/delete/{id}	URI
GET	http:// 192.168.137.1:8080/ Producto/findAll	https://192.168.137.1:8243/ libreria/Producto/findAll	-
GET	http:// 192.168.137.1:8080/ Producto/find/{id}	https://192.168.137.1:8243/ libreria/Producto/find/{id}	URI
POST	http:// 192.168.137.1:8080/ Producto/create	https://192.168.137.1:8243/ libreria/Producto/create	JSON
PUT	http:// 192.168.137.1:8080/ Producto/update/	https://192.168.137.1:8243/ libreria/Producto/update	JSON
DELETE	http:// 192.168.137.1:8080/ Producto/delete/{id}	https://192.168.137.1:8243/ libreria/Producto/delete/{id}	URI
GET	http:// 192.168.137.1:8080/ Trabajador/findAll	https://192.168.137.1:8243/ libreria/Trabajador/findAll	-
GET	http:// 192.168.137.1:8080/ Trabajador/find/{id}	https://192.168.137.1:8243/ libreria/Trabajador/find/{id}	URI
POST	http:// 192.168.137.1:8080/ Trabajador/create	https://192.168.137.1:8243/ libreria/Trabajador/create	JSON
PUT	http:// 192.168.137.1:8080/ Trabajador/update/	https://192.168.137.1:8243/ libreria/Trabajador/update	JSON

[10] Framework de software de código abierto respaldado por un gran ecosistema de herramientas que ayuda a los desarrolladores a diseñar, construir, documentar y consumir servicios web REST. Permite generar clientes Scala, Java, JavaScript, Ruby, PHP y ActionScrip 3.

DELETE	http:// 192.168.137.1:8080/ Trabajador/delete/{id}	https://192.168.137.1:8243/ libreria/Trabajador/delete/{id}	URI
GET	http:// 192.168.137.1:8080/ Transaccion/findAll	https://192.168.137.1:8243/ libreria/Transaccion/findAll	-
GET	http:// 192.168.137.1:8080/ Transaccion/find/{id}	https://192.168.137.1:8243/ libreria/Transaccion/find/{id}	URI
POST	http:// 192.168.137.1:8080/ Transaccion/create	https://192.168.137.1:8243/ libreria/Transaccion/create	JSON
PUT	http:// 192.168.137.1:8080/ Transaccion/update/	https://192.168.137.1:8243/ libreria/Transaccion/update	JSON
DELETE	http:// 192.168.137.1:8080/ Transaccion/delete/{id}	https://192.168.137.1:8243/ libreria/Transaccion/delete/{id}	URI

3.4.4.1. Prueba de configuración utilizando Swagger

3.4.4.1. Configuration test using Swagger

Para probar la API utilizando Swagger en la lista de API de WSO2 (como muestra la Figura 23 Varias API configuradas en WSO2 EI), se selecciona a la derecha de la API la opción Try this API (probar esta API) como muestra la Figura 8 Ventana de API desplegadas de WSO2 EI.

Al seleccionar este enlace se mostrará en una nueva página que contiene los datos relacionados a los recursos de la API. En la Figura 24 Prueba de API utilizando Swagger se aprecia la interfaz de Swagger para la API seleccionada. Se muestran los datos generales de la API y en barras de colores se observan cada uno de los Resources configurados.

Al seleccionarse sobre cada uno se despliegan e indican y muestran los datos generales del Resource, así como las opciones disponibles, en caso de necesitar parámetros lo indicará.

To test the API using Swagger in the WSO2 API list (as Figure 23 shows), select the Try this API option to the right of the API as Figure 8 shows.

Selecting this link will display a new page containing data related to API resources. Figure 24 shows the Swagger interface for the selected API. The general data of the API is displayed and each of the configured Resources is shown in colored bars.

When selected on each one, the general data of the Resource, as well as the available options, are displayed and indicated and shown, in case of needing parameters it will indicate it.

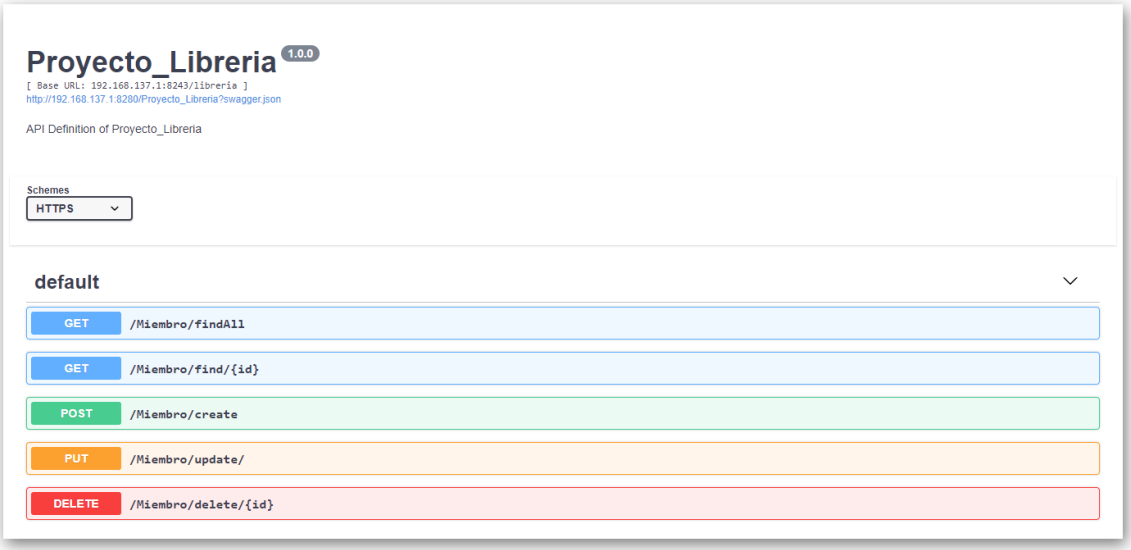

Figura 24 *Prueba de API utilizando Swagger*

Figure 24 *API test using Swagger*

Al desplegarse la primera de las barras se pueden apreciar los datos del Resource perteneciente a la consulta findAll de la tabla Miembro. Como muestra la Figura 25 Prueba de la consulta FindAll utilizando Swagger este Resource no requiere de parámetros para su ejecución. Se selecciona el botón Execute y al esperar unos pocos segundos se muestra el resultado de la consulta. En la parte inferior se muestra el enlace real de la consulta utilizando WSO2, así como los resultados de la misma en formato JSON. En caso de no poder efectuarse la consulta se muestra un mensaje indicando el error de forma simple con formato JSON. Cabe destacar que depende del nivel en el que se hayan configurado los mensajes para el control de errores desarrollado en el proyecto SpringBoot pues sólo se mostrarán los mensajes de retorno que estén en estilo JSON. Por ejemplo, si al añadir un elemento existe una llave duplicada y en el proyecto de SpringBoot no se ha configurado un mensaje JSON para indicar el error, el sistema mostrará un mensaje de error tipo 500 (Internal Server Error, conocido como error interno en el servidor) que resulta genérico para el cliente, siendo difícil conocer la naturaleza del problema.

When the first of the bars is displayed, the Resource data belonging to the findAll query of the Member table can be seen. As shown in Figure 25, this Resource does not require parameters for its execution. The Execute button is selected and after waiting a few seconds the result of the query is displayed. The bottom part shows the actual link of the query using WSO2, as well as the results of the query in JSON format. If the query cannot be made, a message is displayed indicating the error in a simple way with JSON format. It should be noted that it depends on the level at which the messages for error control developed in the SpringBoot project have been configured, since only return messages that are in JSON style will be displayed. For example, if when adding an element there is a duplicate key and in the SpringBoot project a JSON message has not been configured to indicate the error, the system will display an error message type 500 (Internal Server Error, known as an internal error in the server) that is generic for the client, being difficult to know the nature of the problem.

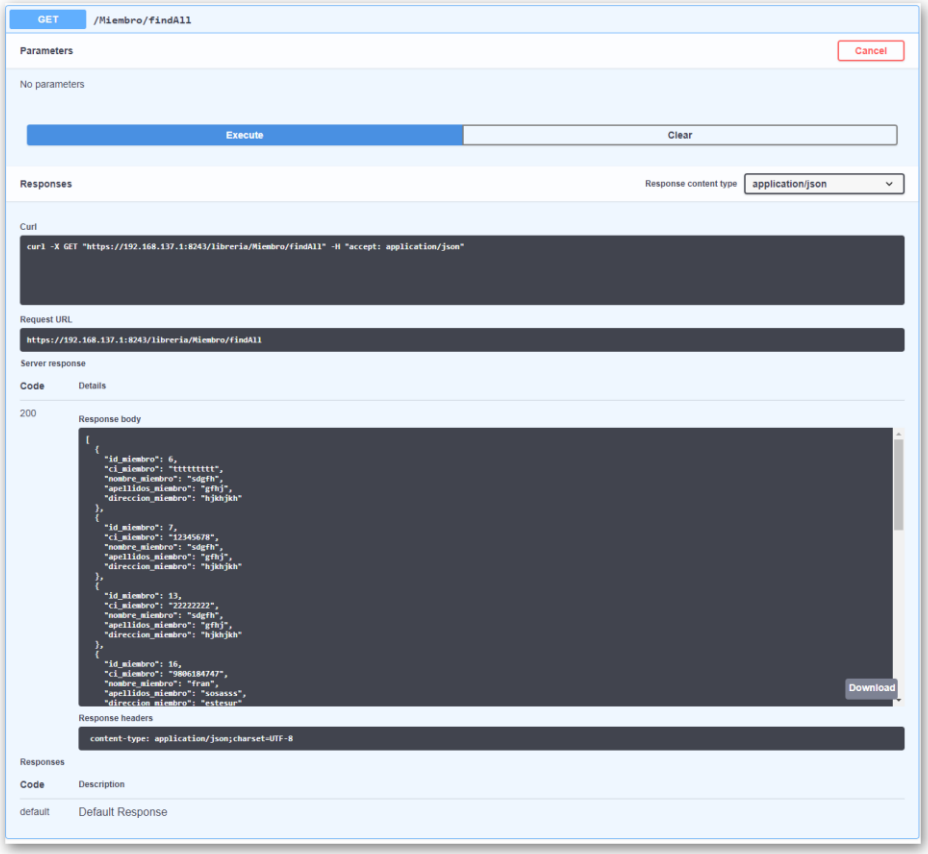

Figura 25 *Prueba de la consulta FindAll utilizando Swagger*

Figure 25 *Testing the FindAll query using Swagger*

Prueba de configuración utilizando Postman

Para probar en Postman se coloca en la barra de direcciones la ruta del Resource (se pueden obtener de las interfaces de prueba de Swagger), se selecciona el método correspondiente y se ejecuta como se muestra en la Figura 26 Ventana de Postman ejecutando la consulta findAll de Miembro utilizando el API creada en WSO2. En caso de que sean necesarios parámetros se indicará antes de realizar la consulta.

Configuration test using Postman

To test in Postman, place in the address bar the path of the Resource (they can be obtained from the Swagger test interfaces), select the corresponding method and execute it as shown in Figure 26. If necessary parameters will be specified before querying.

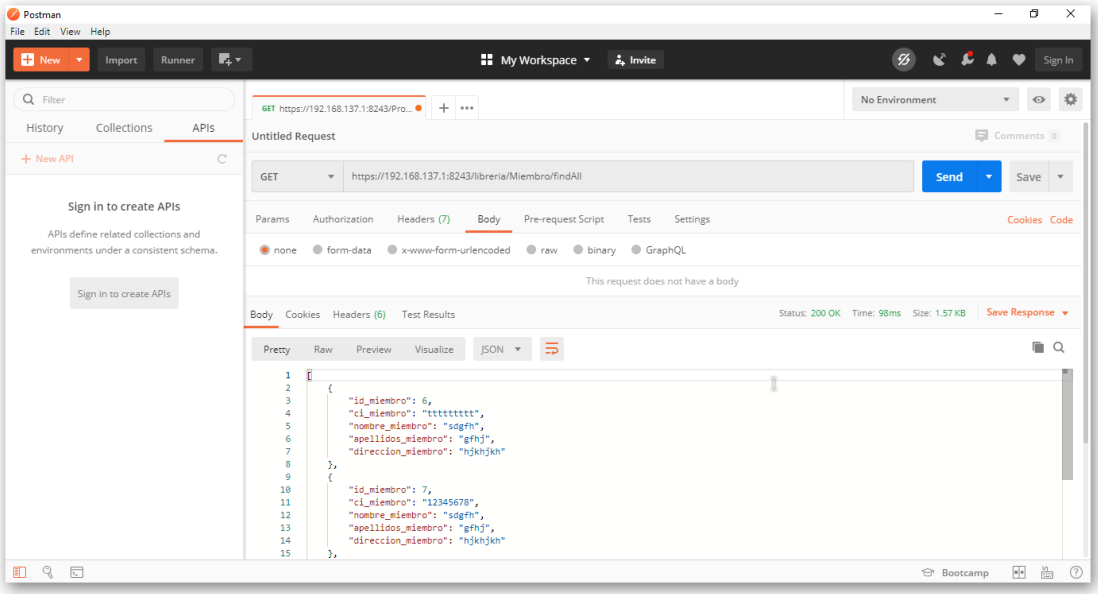

Figura 26 *Ventana de Postman ejecutando la consulta findAll de Miembro utilizando el API creada en WSO2*

Figure 26 *Postman window executing the Member findAll query using the API created in WSO2*

Las rutas de la API original provenientes de SpringBoot, descritas en la **¡Error! No se encuentra el origen de la referencia.**, pueden ser probadas también en Postman como muestra la Figura 27 Ventana de Postman ejecutando la consulta findAll proveniente del proyecto de SpringBoot. La Figura 26 Ventana de Postman ejecutando la consulta findAll de Miembro utilizando el API creada en WSO2 realiza la consulta findAll utilizando el API de WSO2, mientras que la Figura 27 Ventana de Postman ejecutando la consulta findAll proveniente del proyecto de SpringBoot muestra el mismo resultado para la consulta utilizando el API REST proporcionado por

The original API routes coming from SpringBoot, described in Table 1, can also be tested in Postman as shown in Figure 27. Figure 26 performs the findAll query using the WSO2 API, while Figure 27 shows the same result for the query using the REST API provided by SpringBoot, so it can be deduced that the result has been the same and that the WSO2 API works correctly.

SpringBoot, por lo que se puede deducir que el resultado ha sido el mismo y que la API de WSO2 funciona correctamente.

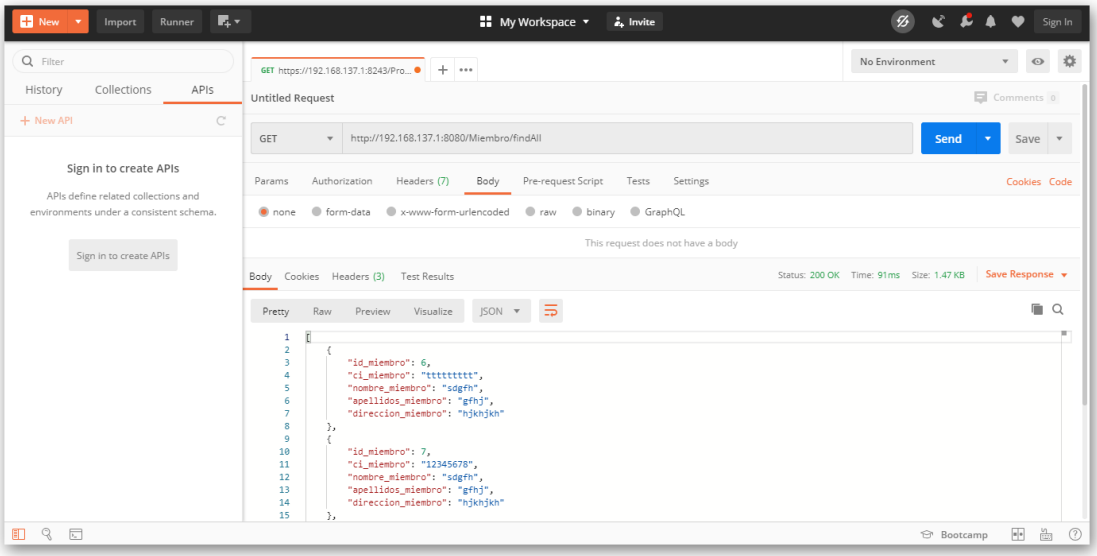

Figura 27 *Ventana de Postman ejecutando la consulta findAll proveniente del proyecto de SpringBoot*

Figure 27 *Postman window executing findAll query from SpringBoot project*

Conclusiones

Tras finalizar la investigación se pudo llegar a las siguientes conclusiones:

- Pueden utilizarse mecanismos de WSO2 para realizar investigaciones de Inteligencia de Negocio y BPMN.
- La comunicación entre los sistemas se realiza mediante protocolo seguro, posibilitando la protección de los datos.
- WSO2 cuenta con una gran gama de funcionalidades capaces de llevar a cabo el seguimiento preciso de las API REST que maneja, posibilitando así conocer la frecuencia del flujo de información entre los sistemas.

Conclusions

After completing the investigation, the following conclusions could be reached:

• WSO2 mechanisms can be used to conduct Business Intelligence and BPMN investigations.

• Communication between the systems is carried out through a secure protocol, enabling data protection.

• WSO2 has a wide range of functionalities capable of accurately monitoring the REST APIs it handles, thus making it possible to know the frequency of the information flow between systems.

Bibliografía / References

Ahuja, S. P., Patel, A. J. C. & Network 2011. Enterprise Service Bus: A Performance Evaluation. 3**,** 133-140 pp.

Alicante, U. D. 2014. *Introducción a los Servicios Web. Invocación de servicios web SOAP*. [Online]. Available: http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html [Accessed 2020].

Almazán, D. A., Tovar, Y. S. & Quintero, J. M. M. J. C. Y. A. 2017. Influencia de los sistemas de información en los resultados organizacionales. 62**,** 303-320 pp.

Arroyave, M. H. T. & Cardona, D. 2012. *Criterios de evaluación de plataformas de desarrollo de aplicaciones empresariales para ambientes web.* Universidad Tecnológica de Pereira. Facultad de Ingenierías Eléctrica .

Azad, T. 2016. *Decision support for middleware performance benchmarking.*

Benalcazar, P., Guillermo, F. & Villagóme, P. F. 2017. *Arquitectura orientada a servicios: instalación y evaluación del ESB de WSO2.*

Box, D., Christensen, E., Curbera, F., Ferguson, D., Frey, J., Hadley, M., Kaler, C., Langworthy, D., Leymann, F. & Lovering, B. J. W. W. W. C. 2004. Web Services Addressing (WS-Addressing), W3C Member Submission 10 August 2004.

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. & Winer, D. 2000. Simple object access protocol (SOAP) 1.1.

Chappell, D. A. 2004. *Enterprise service bus*, " O'Reilly Media, Inc.".

Clark, J. & Deach, S. J. W. W. W. C. W. D. 1998. Extensible Stylesheet Language (XSL), Version 1.0.

Clark, J. & Derose, S. 1999. XML Path Language (XPath) 1.0—W3C Recommendation 16 November 1999. Technical Report REC-xpath-19991116, World Wide Web Consortium.

Fallas, J. J. P. R. E. M. D. V. S. Y. E. D. C. A. U. N. H. C. R. 2003. Conceptos básicos de cartografía.

Fernández, H. M. J. R. C. D. H. Y. E. 2006. SIG-ESAC: Sistema de Información Geográfica para la gestión de la estadística de salud de Cuba. 44**,** 0-0.

Flores, E. J. R. C. D. C. I. 2015. Implementación de lenguajes de contrato electrónico en Oracle Service Bus. 9**,** 63-77 pp.

Gilpin, M. & Vollmer, K. J. T. C. 2005. The Forrester Wave: Enterprise Service Bus.

Indrasiri, K. 2016. Introduction to WSO2 ESB. *Beginning WSO2 ESB.* Springer.

Iyer, R. & Balasundaram, C. 2012. Best practices and case study for open source middleware migration: Egate to apache camel migration.

Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M. & Goland, Y. J. O. S. 2007. Web services business process execution language version 2.0. 11**,** 5 pp.

Kusák, D. 2010. Comparison of Enterprise Application Integration Platforms.

Maréchaux, J.-L. J. I. D. W. 2006. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. 1269-1275 pp.

Menge, F. Enterprise service bus. Free and open source software conference, 2007. 1-6.

Mulesoft. 2019. *Sitio Oficial MuleSoft* [Online]. Available: https://www.mulesoft.com/platform/studio [Accessed 2019].

Newcomer, E. & Lomow, G. 2005. *Understanding SOA with Web services*, Addison-Wesley.

Ortiz, S. J. C. 2007. Getting on board the enterprise service bus. 40**,** 15-17 pp.

Rodríguez , D. A. 2019. Rendimiento de los sistemas de autenticación Single Sign One (SSO) WSO2 IDENTITY SERVER y CAS en Agrocalidad.

Siddiqui, Z., Abdullah, A. H., Khan, M. K. & Alghathbar, K. J. J. O. P. S. 2011. Analysis of enterprise service buses based on information security, interoperability and high-availability using Analytical Hierarchy Process (AHP) method. 6**,** 35-42 pp.

Silver, B. 2004. Enterprise Service Bus Technology for Real-World Solutions. Bruce Silver Associates.

Snyder, B. J. U. H. P. S. N. B. S. S.-I.-A.-S. A. 2008. Service Oriented Integration With Apache ServiceMix. 4**,** 2016.

UCI, U. D. L. C. I. 2019. *Sitio Web Oficial Universidad de las Ciencias Informáticas* [Online]. Available: https://www.uci.cu/investigacion-y-desarrollo/productos/xilema/genesig-20 [Accessed 2019].

Vollmer, K. & Gilpin, M. J. B. S. 2006. The Forrester Wave™: Enterprise Service Bus, Q2 2006.

Wähner, K. 2013. Choosing the Right ESB for Your Integration Needs. Available: https://www.infoq.com/articles/ESB-Integration/.

WSO2. 2019. *Sitio Oficial WSO2* [Online]. Available: https://wso2.com/library/articles/2017/07/what-is-wso2-esb/ [Accessed 2019].