Arquitectura de Integración para Servicios y Soluciones Smart Campus

Integration Architecture for Smart Campus Services and Solutions

DOI: http://doi.org/10.17981/ingecuc.16.2.2020.21

Artículo de Investigación Científica. Fecha de Recepción: 24/07/2020. Fecha de Aceptación: 29/09/2020.

Manuel Alejandro Pastrana Pardo
Institución Universitaria Antonio Jose Camacho. Cali (Colombia)
mapastrana@admon.uniajc.edu.co

Fernando Andrés Cifuentes Calderón
Institución Universitaria Antonio Jose Camacho. Cali (Colombia)
facifuentes@admon.uniajc.edu.co

Hugo Armando Ordoñez Eraso
Universidad del Cauca. Popayán (Colombia)
hugoordonez@unicauca.edu.co

Para citar este artículo:
M. Pastrana Pardo, F. Cifuentes Calderón & H. Ordoñez Eraso, "Arquitectura de Integración para Servicios y Soluciones Smart Campus", *INGE CUC*, vol. 16, no. 2, pp. 267–276. DOI: http://doi.org/10.17981/ingecuc.16.2.2020.21

Resumen

Introducción— La Institución Universitaria Antonio José Camacho (UNIAJC) ha identificado a través del proyecto de investigación "Ecosistema Smart campus", la existencia de problemas complejos de comunicación entre aplicaciones construidas como silos, donde la redundancia de información es común y los procesos que soportan requieren refinamiento. Esta problemática es bastante común, no solo para la UNIAJC, sino en general para cualquier universidad. La razón esta en la medida en que evolucionan las instituciones, los aplicativos que soportan sus procesos son construidos únicamente visionando dar solución a una necesidad específica, sin contar con todo el contexto de soluciones preexistentes, muchas veces por la premura de resolver. Las Arquitecturas Orientadas a Servicios (SOA por sus siglas en inglés), tienen como objetivo, resolver este tipo de situaciones de forma gradual, debido a la dificultad técnica de las implementaciones. Este trabajo se enfoca en exponer la definición de la arquitectura de integración que plantea Smart Campus para la UNIAJC, estableciendo la forma en que interactuarán los servicios y teniendo en cuenta las limitaciones de infraestructura actuales.

Objetivo— Definir una arquitectura de integración la cual permita el fácil escalamiento y buen desempeño de los diversos mecanismos los cuales comunican las aplicaciones que cohabitan en la institución Universitaria Antonio José Camacho.

Metodología— Se aborda el diseño arquitectural de la solución, partiendo del enfoque general, donde se describe las necesidades a resolver mediante el patrón Enterprise Service Bus (ESB) y se representan por medio de un diagrama de componentes. Luego se amplía la información hasta lograr un enfoque detallado. En este último, se selecciona el tipo de servicio a construir (SOAP o REST) y se expone su arquitectura. Por último, mediante ATAM (Architecture Tradeoff Analysis Method), se comprueba la efectividad de la solución.

Resultados— La definición de una Arquitectura de integración de aplicaciones basado en SOA, detallando la arquitectura de software para la construcción de los servicios web de la solución. Esto, teniendo en cuenta las limitaciones de infraestructura y recursos de la institución para lograr un rendimiento de entre 500 p/s a 1000 p/s (peticiones por segundo) que evite problemas de hilos o bloqueo de recursos.

Conclusiones— SOA permite dar solución a problemas de comunicación e integración de aplicaciones, tanto en entornos empresariales como educativos, incrementando la mantenibilidad, escalabilidad y reutilización de los sistemas y componentes desarrollados bajo esta arquitectura. Por otro lado, el modelo para la revisión de la arquitectura ATAM facilitó la definición y evaluación de la arquitectura, logrando tener una comprensión del entorno actual, la identificación de los atributos de calidad, las pruebas y resultados esperados.

Palabras clave— Arquitectura Orientada a Servicios; SOA; servicio web; arquitectura de integración; bus de servicio empresarial

Abstract

Introduction— The Antonio José Camacho University Institution (UNIAJC) has identified, through the "Ecosystem Smart campus" research project, there are complex communication problems between applications built as silos, where the redundancy of information is common and the processes they support require refinement. This problem is quite common, not only for UNIAJC, but in general for any university. The reason is that as institutions evolve, the applications that support their processes are built solely with the vision of providing a solution of a specific need, without counting on the entire context of pre-existing solutions, often due to the haste to resolve. Service-oriented architectures (SOA) aim to resolve these types of situations gradually, due to the technical difficulty of the implementations. This work focuses on exposing the definition of the integration architecture proposed by Smart Campus for UNIAJC, establishing the way in which the services will interact and taking into account the current infrastructure limitations.

Objective— To define an integration architecture that allows easy scaling and good performance of the mechanisms that communicate the applications, which cohabit at the Antonio José Camacho University institution.

Methodology— A brief description of the methodological design of the study. The architectural design of the solution is approached, starting from the general approach, where the needs to be solved are described using the Enterprise Service Bus (ESB) pattern and are represented by a component diagram. The information is then expanded to a detailed focus. In the latter, the type of service to be built (SOAP or REST) is selected and its architecture is exposed. Finally, using ATAM (Architecture Tradeoff Analysis Method), the effectiveness of the solution.

Results— The definition of an SOA-based application integration architecture, detailing the software architecture for building the solution's web services. This, taking into account the Institution's infrastructure and resource limitations to achieve a performance between 500 p/s to 1000 p/s (requests per second) that avoids thread problems or resource blocking.

Conclusions— SOA enables solutions to communication and application integration problems, both in business and educational environments, increasing the maintainability, scalability and reuse of the systems and components developed under this architecture. On the other hand, the ATAM architecture review model facilitated the definition and evaluation of the architecture, achieving an understanding of the current environment, the identification of quality attributes, tests and expected results.

Keywords— Service Oriented Architecture; SOA, web service; integration architecture; enterprise service bus

I. Introducción

La Arquitectura Orientada a Servicios (SOA) conduce a una mejor comprensión de los procesos de negocio de una organización [1], logrando descomponer la lógica del negocio, en unidades de funcionalidad más pequeñas, que se pueden considerar como servicios, los cuales constituirán una plataforma transversal e independiente, en función de satisfacer las necesidades cambiantes del negocio.

Durante años se han realizado diferentes estudios entorno a la aplicación de la Arquitectura SOA y sus beneficios. Al evalúar su aplicación en empresas alemanas [2], donde obtienen resultados positivos relacionados con la optimización de procesos comerciales, agilidad y reducción de costos debido a la reutilización de servicios. Sin embargo, se exponen desafíos para su implementación en cuanto a rendimiento, seguridad y gestión de la arquitectura.

En la Institución Universitaria Antonio José Camacho (UNIAJC), durante una revisión detallada del ambiente en el que cohabitan las aplicaciones existentes, se encuentra que la mayoría no poseen comunicación entre sí. Por el contrario, funcionan como silos, aislados unas de otros y con redundancia de funcionalidades y datos que podrían ser expuestas de una manera reutilizable, disminuyendo los problemas actuales en cuanto a rendimiento, mantenibilidad y escalabilidad.

Lo anterior, expone un escenario que requiere del planteamiento de una solución pensada en términos de capacidad de integración, comunicación asíncrona y exposición de funcionalidades para reutilización. Aunque, en la institución se ha considerado una solución SOA, existe un nivel de incertidumbre en cuanto a su adopción e impacto, razón por la cual el proyecto de investigación "Ecosistema Smart Campus UNIAJC", busca la definición de una arquitectura de integración, partiendo de conocer y comprender el entorno actual, con sus limitaciones de infraestructura y la forma en que interactuarán los servicios.

El resto del articulo está organizado de la siguiente manera, la sección II proporciona una revisión de la literatura relacionada con el diseño de Arquitectura de software. La sección III expone la arquitectura general de integración propuesta como parte de una solución SOA. La sección IV menciona el uso y ventajas del patrón Enterprise Service Bus. La sección V expone la arquitectura detallada de la solución en donde se especifican los patrones utilizados. La sección VI detalla la arquitectura para la construcción de los servicios web. La sección VII define el estilo de interacción para la construcción de los servicios. La sección VIII presenta los resultados y el enfoque utilizado para su evaluación. La sección XI presenta las conclusiones y el trabajo futuro.

II. Marco teórico

Los diseños arquitecturales del software tienen relación con la estructura y las decisiones que se toman para soportar los requisitos funcionales y desempeño de los sistemas, así como también otros requisitos no funcionales que están relacionados con la confiabilidad, escalabilidad, operatividad y disponibilidad [3]. El modelo de 4 vistas más 1 busca exponer las decisiones que guían la estructura para la construcción de un software [4]. Cada vista está enfocada en un público objetivo, que espera mediante diagramas UML, ver los elementos que componen la solución. Para poder diagramar de manera correcta, se deben identificar con claridad los atributos de calidad o comportamientos deseados del sistema como la escalabilidad, mantenibilidad y modificabilidad, entre otros, que permiten exponer las características deseadas de la estructura. Cuando el equipo ha identificado estos atributos, procede a generar soluciones a nivel de implementación para resolver estos atributos. Lo anterior, se conoce como patrones de diseño, que son soluciones comprobadas a problemas recurrentes [5].

Cabe mencionar que, sin importar el modelo de desarrollo de software a implementar, todos concuerdan en sus primeras etapas en realizar el diseño de la arquitectura, la cual será la base para la construcción del software, como es el caso de modelo tradicional en cascada, cuya segunda fase hace referencia al diseño del sistema y para el caso del desarrollo ágil, el framework Scrum, el diseño de la arquitectura de software hace parte de su fase principal [3].

Con el diseño de la arquitectura, es posible identificar los principales componentes estructurales en un sistema y la relación entre ellos. La importancia en la definición de la arquitectura radica en que afecta el desempeño, así como la capacidad de distribución y mantenimiento de un sistema [6]. Similarmente, si la arquitectura no facilita el mantenimiento de forma natural los cambios en esta pueden generar altos costos a futuro. Por otra parte, la arquitectura de software tiene la capacidad de cumplir con dos funciones una como plan de diseño para negociación de requerimientos y la segunda como medio para generar debate con clientes, desarrolladores y administradores [5].

III. Arquitectura General

En esta etapa se establece, mediante un diagrama de componentes, la representación de la vista de despliegue, donde se visualiza la definición del uso de una capa de servicios que proporcionará el "Ecosistema Smart Campus". Esta capa consiste en una composición de servicios expuestos dentro de un servidor de aplicaciones, que serán consumidos por un ESB (Enterprise Service Bus), permitiendo a las aplicaciones acceder a la información académica y administrativa (Fig. 1).

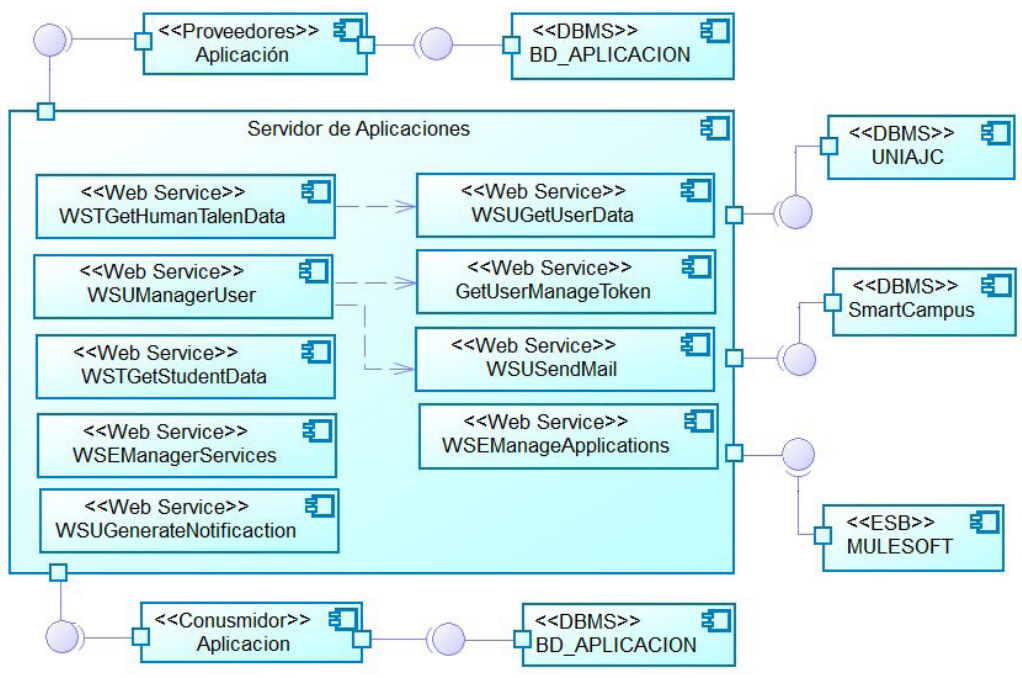

Fig. 1. Arquitectura general de integración.
Fuente: Autores.

Sí bien, esto soluciona una parte del problema, aún no es una solución SOA completa. Dentro de SOA, se requiere que los servicios sean gobernables y orquestados para garantizar mantenibilidad, escalabilidad y performance. Por lo anterior se hace necesario el uso del patrón ESB.

IV. Uso del Patrón Enterprise Service Bus en la Solución

Un ESB define un entorno diseñado para la interconectividad entre servicios [6]. Establece una capa intermedia de procesamiento que puede ayudar a superar problemas comunes asociados con confiabilidad, escalabilidad y la disparidad en las comunicaciones.

Una de las principales ventajas que presenta la implementación de este patrón es la orquestación, definida como la capacidad de colaboración, orden o secuencia de la ejecución de una serie de servicios independientes, para llevar a cabo la ejecución de un proceso dentro de una organización [6]. Adicionalmente, el uso de este patrón siempre lleva una correlación con otros patrones como son las comunicaciones asíncronas a través de colas, la transformación de formatos de datos que llevan los mensajes y las fachadas de servicio como único punto de inicio transaccional de un flujo de proceso.

Para la implementación del ESB se evalúan IBM y Mulesoft como posibilidades. Para tomar una correcta decisión sobre cual seleccionar, se toma como referente el análisis de Gartner Peer Insights [7], el cual ofrece una evaluación de las características proporcionadas por los proveedores, de acuerdo a las experiencias de varias organizaciones en su implementación, en una escala de calificación de 1 a 5, siendo 1 la peor calificación y 5 la mejor.

De acuerdo a los resultados de la Tabla 1, la mejor opción sugerida por la investigación realizada incida que la herramienta recomendada es Mulesoft [17], debido a que permite conectar aplicaciones de forma rápida y fácil, intercambiar datos, la creación, alojamiento y mediación de servicios, así como también el enrutamiento de mensajes, transformación de datos, gestionar la comunicación entre servicios web, y funcionar como una capa intermedia, logrando proporcionar servicios de integración de distintas apps a través de mensajería basada en estándares y servicios de sincronización. Otra ventaja es que el ESB Mule es de código abierto, y cuenta con una plataforma de desarrollo denominada Anypoint estudio, la cual permite abarcar desde el inicio de la API hasta la administración de esta en producción, facilitando la gestión del ciclo de vida completo de la misma.

Tabla 1. Comparación Mulesoft, IBM.

Carácteristicas	Mulesoft	IBM
Capacidad del producto	4.5	4,5
Conectores del protocolo de comunicación	4.6	4,3
Formatos de datos	4.8	4,3
Transformación y mapeo de datos	4.4	4,3
Calidad de los datos	4.5	3,8
Enrutamiento y orquestación	4.4	4
Gestión del ciclo de vida completo API	4.2	3,8
Facilidad de implementación	4.2	4
Servicio y soporte, calidad de apoyo técnico	4.6	4

Fuente: [7].

V. Arquitectura Detallada

El patrón ESB guiará a la arquitectura SOA de la solución, como su patrón dominante, por los beneficios expuestos anteriormente, complementando el modelo expuesto en la Fig. 2, que será descrito con mayor detalle a continuación, donde adicionalmente, otras decisiones se resaltan en conjunto para ampliar la propuesta.

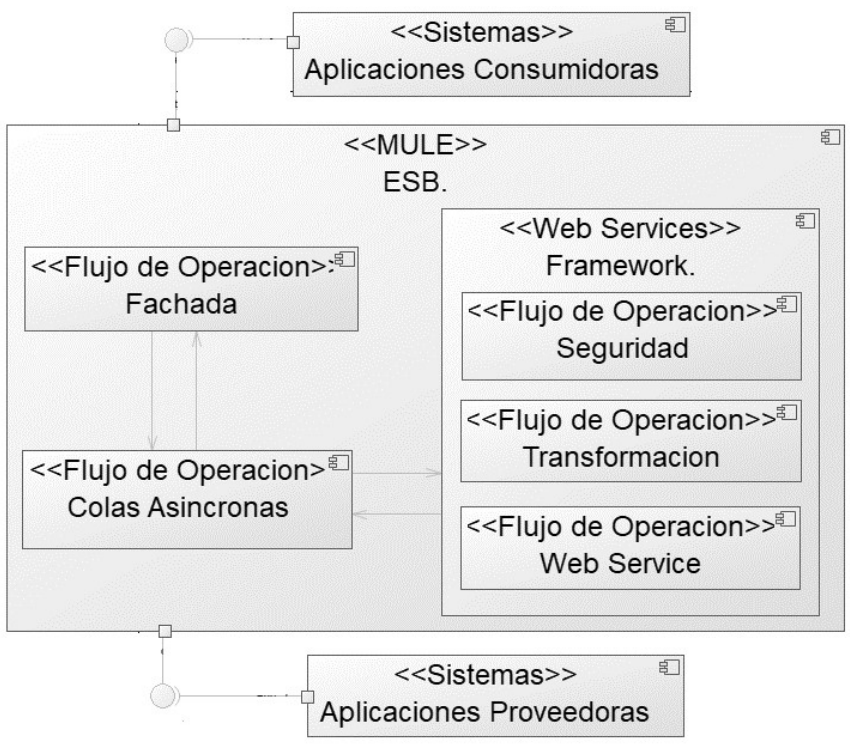

Fig. 2. Arquitectura detallada.
Fuente: Autores.

La arquitectura detallada se centra en la definición del bus de servicio empresarial, la cual permitirá la orquestación de los servicios y la interacción entre aplicaciones consumidoras y proveedoras de información.

Con el objetivo de simplificar la interface de comunicación de las aplicaciones con los servicios expuestos, se define el uso del patrón fachada [8], el cual es implementado a través de un componente, encargado de recibir y enrutar los mensajes hacia un flujo de operación, que busca garantizar todas las peticiones sean atendidas; y en caso de una falla, el sistema sea capaz de mantener la petición y realizar reintentos, esto se logra en conjunto con el uso del patrón de Colas asíncronas. Adicionalmente, dentro del ESB se ha definido el uso de un framework propio, cuyo objetivo es proporcionar componentes de seguridad y transformación que garanticen una adecuada interacción con los servicios web.

Para el componente de seguridad de la información se pretende garantizar a través del patrón *Confidencialidad de los datos*: el cual consiste en cifrar el contenido del mensaje independientemente del transporte [6].

Por otro parte, para interactuar con sistemas existentes o sistemas legados, esto es posible con el patrón de *Transformación del Formato de los Datos*, permitiendo crear un canal de comunicación con sistemas legados a través de la transformación de los datos [6].

Con el fin de proporcionar manejo excepciones y evitar que los servicios divulguen información de su implementación cuándo ocurre una excepción, se implementa el patrón el *Blindaje de Excepción*: con el cual se pretende proteger u ocultar los mensajes de excepción en cada servicio ante los usuarios. Permitiendo manejar errores inesperados del sistema, sin exponer la estructura del servicio y se vea comprometida la seguridad de este [6].

Una vez definidos los patrones de diseño, el siguiente paso es definir la arquitectura del servicio.

VI. Detalle Arquitectural del Servicio Web dentro de la Solución

Un servicio es posible considerarlo como una unidad funcional que contiene lógica del negocio, y puede encapsular una tarea llevada a cabo en un único o varios pasos [9]. SOA define a los servicios con características de reutilización, agilidad, autonomía, interoperabilidad y acoplamiento flexible, estos deben responder a una estandarización en su construcción que facilite a futuro su gobernanza, garantice estabilidad, mantenimiento y escalabilidad de los mismos [10].

SOA no impone para la construcción de los servicios una arquitectura en específico, si no que define, sugiere una serie de patrones de diseño y buenas prácticas, las cuales requieren una selección para su aplicación acorde a un previo análisis de las necesidades de una organización. Así que, para la definición de la arquitectura de los servicios se inicia por determinar qué características se deben proporcionar para satisfacer las necesidades de la institución, posteriormente se identifican los atributos de calidad, y los respectivos patrones de diseño para garantizar el cumplimiento de estos atributos. Los patrones tenidos en cuenta, en su mayoría son patrones de diseño SOA, con el fin de garantizar no solo los atributos de calidad, sino que también su orientación en función a servicios de integración. Los patrones seleccionados y expuestos en la Tabla 2 están basados en estudios sobre el tema [6], [8].

Tabla 2. Atributos de calidad y patrones de diseño.

Requerimiento	Atributo de calidad	Patrón de diseño	Descripción
La capacidad de que el sistema esté total o parcialmente operativo al mismo tiempo que es requerido para manejar eficazmente las fallas que puedan afectar la disponibilidad del sistema.	Disponibilidad	Colas Asincronas	Maneja las peticiones de las aplicaciones consumidoras y evitar el bloqueo innecesario de recursos. Guarda quien envía. Mientras tenga un estado de pendiente se dispara un flujo que hace que las peticiones se reenvíen hasta que cambie su estado a finalizada.
Lograr conectividad con todos los actores y aplicaciones de los distintos sistemas.	Inter-operabilidad	Data Format Transformation	Se encarga de la lógica de transformación de formato de datos intermedios, para interacción entre servicios y/o programas.
		Request Response con acuse de recibo	Logrando asignar una respuesta a cada solicitud, junto con información adicional acerca del resultado de la transacción realizada, facilitando la comunicación entre los servicios.

Requerimiento	Atributo de calidad	Patrón de diseño	Descripción
La información que será expuesta, contiene información confidencial acerca de los estudiantes, personal administrativo, o información relacionado con las transacciones realizadas para el funcionamiento diario de la institución.	Seguridad	Exception Shielding	Proteger u ocultar los mensajes de excepción en cada servicio a los usuarios.
		Autenticación por token	Las solicitudes son cifradas con un usuario y contraseña válidos, se ve reflejado a nivel de la comunicación de los servicios.
		Service Facade	Simplifica la comunicación entre servicios, recibe las peticiones, delega quien atiende las peticiones.
Los servicios deben tener la capacidad de ser modificados para corregir fallos o realizar mejoras de una forma rápida y que no implique gran costo.	Escalabilidad, Mantenibilidad	Multi Capas	Separar la lógica que compone un software en varias capas que permitan minimizar la dependencia entre estas facilitando su desarrollo y mantenibilidad.
Los servicios deben ofrecer un adecuado rendimiento y manejo de los recursos.	Eficiencia	Inyección de dependencias	Permitir suministrar los objetos a otras clases a través de una interfaz en lugar de estas crear una instancia para el objeto cada vez que lo requiera.
		Singleton	Garantizar que las transacciones se realicen una sola vez.
Garantizar un control sobre el acceso a la información de las bases de datos.	Confiabilidad	DAO	Permite tener objetos de acceso a datos, siendo esta la única capa que se comunica con la base de datos.

Fuente: Autores [6], [8].

Con los patrones identificados, se genera un diseño arquitectural del servicio como se aprecia en la Fig. 3, bajo la cual se construirán los servicios.

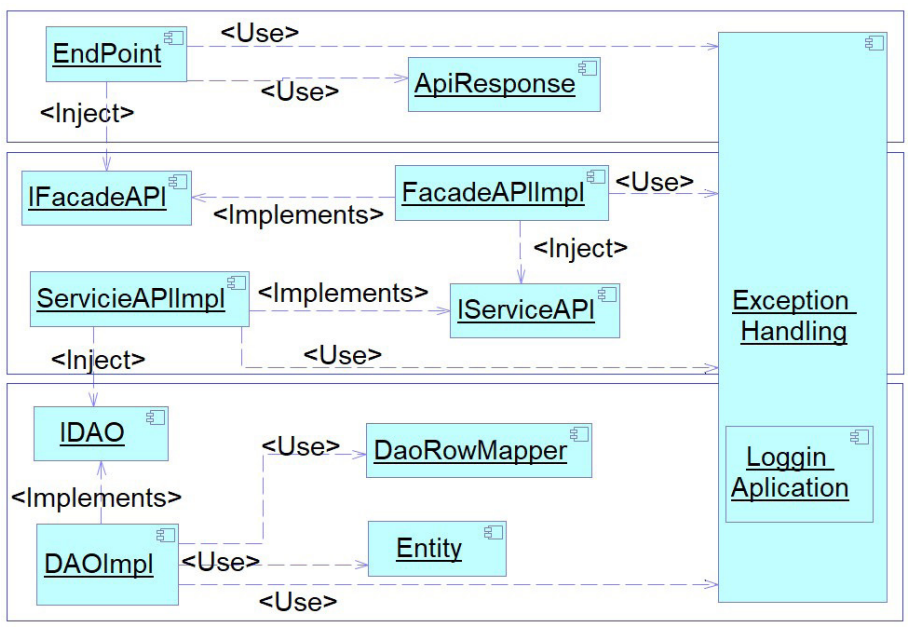

Fig. 3. Arquitectura del servicio.
Fuente Autores.

Una vez definida la arquitectura del servicio, el siguiente paso es iniciar con la construcción del servicio. Para la construcción del servicio se debe empezar por evaluar el estilo de interacción, y las características dependiendo del estilo seleccionado.

VII. Construcción del Servicio

Los servicios web se pueden desarrollar de acuerdo a dos estilos de interacción [11]. El primero basado en el Protocolo de Acceso a Objetos Simple (SOAP) y el otro en un estilo arquitectural de software denominado Transferencia de Estado Representacional (REST).

En un servicio REST los datos y la funcionalidad se consideran recursos [11]. La forma de acceder a ellos, es utilizando Identificadores Uniformes de Recursos (URI). No existe un formato definido para estos mensajes, y por esto la estructura es definida por el tipo de datos y/o criterio del desarrollador. Para el caso de un servicio web SOAP, también se indica que se referencia a la forma en que el servicio realizará el intercambio de información [11]. Es decir

que la arquitectura, el formato de los mensajes y las funcionalidades, está definido por el Protocolo Simple de acceso a Objetos (SOAP), basado en XML y expuesto por un contrato de servicio llamado WSDL [11]. Debido a su estructura, requieren de un análisis sintáctico de los mensajes que puede llegar a ser intensivo en memoria y computación, consumiendo mayor ancho de banda y recursos. En contraposición, los servicios web REST, no tienen especificaciones estrictas como SOAP, permitiendo un procesamiento e intercambio de mensajes más rápido y consumo menor de recursos.

Con base a otro análisis [11], aunque REST obtenga mejor rendimiento, se considera que es conveniente crear la aplicación con SOAP debido al soporte adecuado de las herramientas existentes para su construcción, esto apoyado también en varios trabajos de diversos investigadores [12], [13]. Por otro lado, es posible considerar SOAP más seguro, debido a que cuenta con una especificación estándar que define su seguridad [14].

Por lo anterior, para el desarrollo de los servicios web en la institución, se elige el desarrollo bajo el protocolo SOAP, debido a que la mayoría de las aplicaciones que consumirán los servicios, manejarán información que requiere garantizar confidencialidad y niveles de seguridad. Dicha decisión se toma basada también en que los servicios bajo el protocolo SOAP ofrecen una estructura más robusta para entornos empresariales, que requieren garantizar estandarización, interoperabilidad, integridad y confiabilidad de la información. Para suplir deficiencias en el rendimiento se tendrá en cuenta técnicas para la optimización.

No todas las comunicaciones requieren la totalidad de las funcionalidades ofrecidas por SOAP, por lo que se debe analizar y seleccionar cuales son las apropiadas de acuerdo a las necesidades, y de esta forma lograr aumentar la capacidad de rendimiento del servicio. Ciertas técnicas de optimización [15], se pueden tener en cuenta se encuentran para esta investigación.

Una vez seleccionado el protocolo SOAP, sea hace necesario tener en cuenta que existen dos versiones, 1.1 y 1.2; para la construcción de los servicios se seleccionó la versión 1.2, porque está basado en el conjunto de información XML, lo que facilita describir el documento con el esquema XSD. También ofrece la posibilidad de que el protocolo de transporte no sea necesariamente http. Disminuye lo problemas de interoperabilidad entre distintos proveedores en cuanto al manejo de los perfiles WS-I. SOAP 1.2 cuenta con SAAJ un mecanismo que da la posibilidad de representar los mensajes SOAP 1.1 y los mensajes adicionales con formato SOAP 1.2. En la especificación del WSDL es capaz de describir ambas versiones, logrando realizar la migración y mantener la interoperabilidad con aplicaciones legados que utilizan la versión 1.1.

Una vez definido el estilo de interacción, se construye un servicio web de acuerdo a lo establecido, con la finalidad de comprobar a nivel arquitectural mediante pruebas de carga, volumen y estrés, que los servicios cumplen con la escalabilidad y performance esperados. El servicio es construido utilizando "Spring Framework" para el desarrollo en *Java*, debido a que facilitará la implementación de la arquitectura definida. Para la herramienta de gestión y comprensión de proyectos de software se utiliza *Maven*, por último, para las pruebas del servicio se utiliza *SOAP UI*.

VIII. Resultados

Para evaluar los resultados es posible utilizar varios enfoques por escenarios. Uno de los más utilizados es ATAM (Architecture Tradeoff Analysis Method) [16], debido a su facilidad de aplicación. A continuación, se indican los pasos a realizar para el análisis con su respectivo resultado.

A. *Método de evaluación*

En este punto se explica brevemente las técnicas utilizadas y los resultados que se pueden obtener [16]. El tipo de pruebas a realizar son de carga, volumen y estrés. Las características del computador donde se ejecutarán son: Procesador Intel® Core™ i7-7500U a 2.70 Hz - 2.90 Hz; Disco Duro SATA3 tipo M.2 SSD; Memoria RAM de 8 Gb DDR4 a 2 133 MHz; Sistema Operativo: Windows 10 a 64 bits.

Las pruebas de volumen se realizan mediante la herramienta *SoapUI*, donde se debe definir el número de hilos, con una demora especificada entre cada ejecución para simular un espacio de respiración en el servidor. En este caso se desea ejecutar una prueba funcional con 500 sub-

procesos, con demora aleatorias entre 10 y 15 segundos. Los 500 subprocesos son definidos en base a un promedio de usuarios proyectados bajo un funcionamiento cercano a una hora pico de la aplicación consumidora. Una vez definidos los parámetros, se ejecutan las pruebas y se procede a revisar el Monitor de Recursos del equipo.

Para realizar pruebas de estrés, se utiliza la misma herramienta y la estrategia de varianza, que renueva el número de sub procesos a lo largo del tiempo en un feudo de «diente de sierra». En este caso, se inicia con 500 hilos, un intervalo en 60 y la Varianza en 0.8. La cantidad de hilos aumentará de 500 a 740 en los primeros 15 segundos, luego disminuirá nuevamente a 500 y continuará hasta 4 hilos después de 45 segundos. Finalmente vuelve al valor inicial después de 60 segundos. En los diagramas de estadísticas de salud del equipo, es posible observar la varianza fácilmente.

B. *Metas de negocio*

Explica el contexto de la arquitectura y el resultado que se espera desde la visión del negocio [16]. Se espera un rendimiento bajo con una carga de entre 500 p/s 1000 p/s, capaz de soportar la transaccionalidad de las horas pico.

C. *Arquitectura*

Presenta la arquitectura y el enfoque de la solución [16]. Esta fue detallada en la sección 3, 4, 5 y 6 de este artículo.

D. *Escenario(s)*

Aquí se identifican los atributos de calidad que deben ser medidos para evaluar la efectividad de la solución [16]. En este caso se busca medir el rendimiento y la escalabilidad.

E. *Análisis*

Se analiza el enfoque arquitectural, de acuerdo con el escenario específico y las metas de negocio, aplicando los métodos de evaluación propuestos como indica [16].

F. *Presentación de resultados*

Aquí se exponen las conclusiones de acuerdo a los resultados obtenidos [16]. Con la ejecución de la prueba de volumen, se logra afirmar que no hay problemas de hilos o bloqueo de recursos. Se garantiza que el servicio funciona como se espera bajo una carga moderada, cumpliendo con la capacidad de 500 p/s y escalando hasta 1 000 p/s. El procesador tiene un pico de uso hasta el 100% pero se recupera rápidamente, como se evidencia en la Fig. 4.

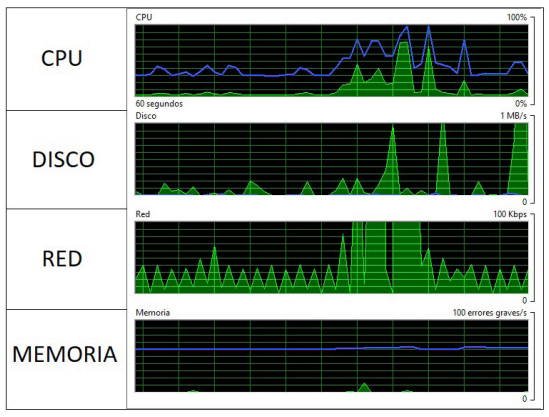

Fig. 4. Monitor de Recursos bajo la ejecución del escenario.
Fuente: Autores.

En cuanto a las pruebas de estrés, al variar la carga a lo largo del tiempo entre 500 a 749 subprocesos, sin problemas operativos, se demuestra que el servicio está diseñado para la recuperación, evitando bloqueo de recursos. Aunque alcanza un pico máximo del 100% en el uso del procesador este se recupera rápido. El uso de la memoria se mantiene en su estado inicial 49% - 52% en uso, lo que no representa una carga significativa, como se evidencia en la Fig. 5.

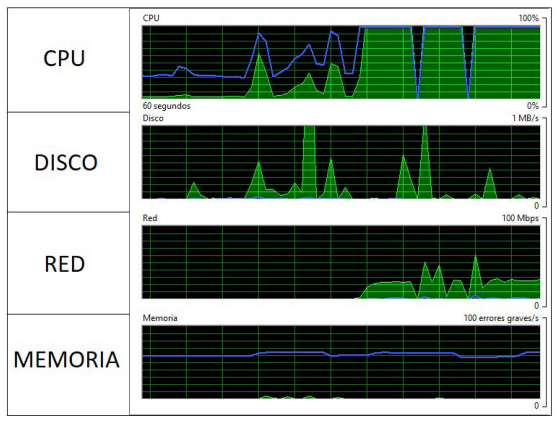

Fig. 5. Monitor de Recursos bajo la ejecución del escenario prueba de estrés.
Fuente: Autores.

Lo anterior permite evidenciar que las decisiones arquitecturales como el uso inyección de dependencias y la implementación de múltiples capas, generan un manejo dinámico de los recursos, lo que garantiza un buen rendimiento y escalabilidad de la solución.

IX. Conclusiones

La implementación de soluciones SOA dentro de la UNIAJC, permitirá la reutilización de funcionalidades comunes entre diversos sistemas, a partir de su exposición por medio de servicios. Esto incrementa la mantenibilidad y escalabilidad de los sistemas que se desarrollen. Así mismo, el uso de componentes genéricos, evita la redundancia de información entre las diversas bases de datos de la institución, generando una mayor integridad y confiabilidad de la misma. Esto es de vital importancia para la institución debido a sus limitaciones de infraestructura.

El trabajo realizado, evidencia que resulta útil para soluciones SOA empezar por un diseño de arquitectura global, que permite entender de manera genérica los componentes que requiere la implementación, a fin de determinar el rol que estos desempeñaran y comprender en cuales se requiere un nivel de detalle más amplio.

Por otra parte, el uso de pruebas de carga, volumen y estrés, permite comprobar si se satisface o no los atributos de rendimiento y escalabilidad de la solución. En este caso, de acuerdo a los resultados obtenidos al construir y probar el desarrollo de un servicio web, a partir de la arquitectura propuesta, se determina que cumple con los atributos de calidad esperados y que soportará la demanda planeada. Es decir, que el diseño del servicio cumple, en su implementación, con soportar los requisitos no funcionales de la solución determinados en las metas del negocio del modelo ATAM. Por lo que se implementará en el desarrollo de los servicios web dentro el ecosistema Smart Campus.

El enfoque propuesto por ATAM (Architecture Tradeoff Analysis Method), representa un modelo para la revisión de una arquitectura, donde al tener claro los objetivos o metas de negocio y los atributos de calidad arquitectural del escenario que se desean satisfacer, es posible definir el tipo de pruebas a realizar y los resultados que se deben obtener. Esto genera un orden adecuado para el análisis y la exposición de los resultados, permitiendo contrastar de una manera clara el éxito o la necesidad de mejora de la solución.

En futuras investigaciones se espera evaluar la implementación de la arquitectura general y el comportamiento en la orquestación de los servicios dentro del Bus de Servicio Empresarial.

Financiamiento

Artículo de investigación científica derivado del proyecto de investigación Ecosistema Smart Campus, financiado por la institución Universitaria Antonio José Camacho. Año de inicio: 2019, año de finalización: 2020.

Agradecimientos

Los autores desean expresar su agradecimiento a la profesora Ana Milena Rojas Calero, directora del proyecto de investigación Ecosistema Smart Campus y a la institución Universitaria Antonio José Camacho, por la oportunidad de ayudar a transformar la institución hacia una versión cada vez más eficiente para toda su comunidad.

Referencias

[1] T. Erl, *SOA Design Patterns*. BS, USA: Prentice Hall, 2009.

[2] A. Becker, T. Widjaja & P. Buxmann, "Value Potentials and Challenges of Service-Oriented Architectures," *Bus Inf Syst Eng*, vol. 3, no. 4, pp. 199–210, Aug. 2011. https://doi.org/10.1007/s12599-011-0167-3

[3] R. S. Pressman & B. R. Maxim, Software Engineering: A Practitioner's Approach*, 8 Ed*. BS, USA: McGraw-Hill, 2015.

[4] P. Kruchten, "The 4+1 View Model of architecture," *IEEE Softw*, vol. 12, no. 6, pp. 42–50, 1995. https://doi.org/10.1109/52.469759

[5] H. Gomaa, *Software Modeling and Design*. Cambs, EN: Cambridge University Press, 2011.

[6] M. Peleg & S. W. Tu, "Design patterns for clinical guidelines," *Artif Intell Med*, vol. 47, no. 1, pp. 1–24, Sep. 2009. https://doi.org/10.1016/j.artmed.2009.05.004

[7] Gartner Peer Insights. "IBM vs MuleSoft: Gartner Peer Insights 2021." *gartner.com*. https://www.gartner.com/reviews/market/application-integration-platforms/compare/ibm-vs-mulesoft (accessed 2017)

[8] I. Tounsi, M. H. Kacem, A. H. Kacem & K. Drira, "An Approach for SOA Design Patterns Composition," presented *8th International Conference on Service-Oriented Computing and Applications*, SOCA, ROM, IT, 19-21 Oct. 2015. https://doi.org/10.1109/SOCA.2015.43

[9] E. Hewitt, *Java SOA cookbook*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2009.

[10] O. Zimmermann, "Architectural Decisions as Reusable Design Assets," *IEEE Softw*, vol. 28, no. 1, pp. 64–69, Jan. 2011. https://doi.org/10.1109/MS.2011.3

[11] K. Wagh & R. Thool, "A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host," *J Inf Eng Appl*, vol. 2, no. 5, pp. 2–16, 2012. Available: https://www.iiste.org/Journals/index.php/JIEA/article/view/2063

[12] S. Malik & D.-H. Kim, "A comparison of RESTful vs. SOAP web services in actuator networks," presented *2017 Ninth International Conference on Ubiquitous and Future Networks*, ICUFN, MI, IT, 4-7 July 2017. https://doi.org/10.1109/ICUFN.2017.7993893

[13] S. Kumari & S. K. Rath, "Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," presented *2015 International Conference on Advances in Computing, Communications and Informatics*, ICACCI, COK, IN, 10-13 Aug. 2015. https://doi.org/10.1109/ICACCI.2015.7275851

[14] K. Bhargavan, R. Corin, C. Fournet & A. D. Gordon, "Secure sessions for web services," presented *2004 workshop on Secure web service,* SWS '04, FFX, VA, USA, 24 Oct. 2004. https://doi.org/10.1145/1111348.1111355

[15] A. W. Mohamed & A. M. Zeki, "Web services SOAP optimization techniques," presented *4th IEEE International Conference on Engineering Technologies and Applied Sciences,* ICETAS, Salmabad, BHR, 29 Nov.-1 Dec. 2017. https://doi.org/10.1109/ICETAS.2017.8277881

[16] B. Costa, P. F. Pires, F. C. Delicato & P. Merson, "Evaluating REST architectures—Approach, tooling and guidelines," *J. Syst. Softw.*, vol. 112, no. C, pp. 156–180, Feb. 2016. http://dx.doi.org/10.1016/j.jss.2015.09.039

[17] Mulesoft Plataforma Anypoint. (2021), MuleSoft, LLC. SF, CA, USA. [*Online*]. Available: https://www.mulesoft.com/platform/enterprise-integration

Manuel Alejandro Pastrana Pardo es Ingeniero de sistemas de la Universidad Santiago de Cali (Colombia). Esp. en procesos para desarrollo de software y MSc. en Ingeniería de software de la Universidad San Buenaventura (Cali, Colombia). De 2010 a 2016 trabajó para compañías desarrolladoras de software a la medida para el ámbito nacional e internacional. Desde el 2016 ha trabajado como consultor independiente, orientando consultorías para la mejora de los procesos de desarrollo de software. Se vinculó a la Institución Universitaria Antonio José Camacho de Colombia, Sede Cali en el año 2015 y es profesor ocasional tiempo completo desde el año 2018. Actualmente, pertenece al grupo de investigación Grintic. https://orcid.org/0000-0002-6506-0659

Fernando Andrés Cifuentes Calderón es Ingeniero de sistemas en la Institución Universitaria Antonio José Camacho (Cali, Colombia), obteniendo mención de honor y tesis meritoria. Esp. en procesos para desarrollo de software de la Universidad San Buenaventura (Cali, Colombia). Actualmente, trabaja en la misma institución como Analista programador en el proyecto de investigación Ecosistema Smart Campus, en donde contribuyó en la definición de la arquitectura y metodología de trabajo. Hace parte del grupo de investigación Grintic. https://orcid.org/0000-0001-8186-8060

Hugo Armando Ordoñez Eraso es Ingeniero de sistemas en la Fundación Universitaria San Martin (Pasto, Colombia). Esp. en gerencia informática de la Corporación Universitaria Remington (Colombia). Magister en Computación y Phd. en Ingeniería Telemática de la universidad del Cauca (Popayán, Colombia). Actualmente, trabaja en la universidad del Cauca. Hace parte del grupo de investigación GTI. https://orcid.org/0000-0002-3465-5617