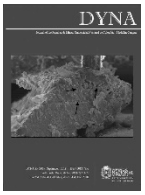
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
FACULTAD DE MINAS



DYNA
http://dyna.medellin.unal.edu.co/

Integrating SOA and MAS in Intelligent Environments

Manuel Sánchez [a,c], Jose Aguilar [b] & Ernesto Exposito [c]

[a] Departamento de Ing. en Informática, Universidad Nacional Experimental del Táchira, Venezuela. mbsanchez@unet.edu.ve
[b] Centro de Estudios en Microelectrónica y Sistemas Distribuidos (CEMISID), Universidad de Los Andes, Mérida, Venezuela. aguilar@ula.ve
[c] Univ Pau & Pays Adour/E2S UPPA, Laboratoire d'informatique de L'universite de Pau et des Pays de L'adour, Pau, France. m.sanchez@univ-pau.fr, ernesto.exposito@univ-pau.fr

Received: November 2nd, 2017. Received in revised form: April 27th, 2018. Accepted: May 7th, 2018.

Abstract
Service Oriented Architecture (SOA) has emerged as the dominant architecture for interoperability between applications, by using a weak-coupled model, based on the flexibility provided by Web Services, which has led to a wide range of applications, in what is known as cloud computing. On the other hand, Multiagent System (MAS) is widely used in the industry, because it provides an appropriate solution to complex problems in a proactive and intelligent way. Specifically, the Educational Smart Environments (AmI) obtain great benefits by using both architectures, because MAS endows intelligence to the environment, while SOA enables users to interact with academic services in the cloud. The purpose of this article is to propose an architecture for bidirectional integration SOA-MAS to be used on educational AmI. The proposed solution allows taking advantages of both architectures, and solves integration problems raised in previous researches.

Keywords: ambient intelligence; SOA; MAS; cloud computing.

Integración SOA-MAS en Ambientes Inteligentes

Resumen
SOA (Service Oriented Architecture) ha emergido como una arquitectura dominante para la interoperabilidad entre aplicaciones, por medio de un modelo de acoplamiento débil basado en la flexibilidad que proveen los servicios web, esto ha dado lugar a una amplia gama de aplicaciones, en lo que se conoce hoy en día como computación en la nube. Por otro lado, los MAS (Multiagent System, por sus siglas en inglés) son usados ampliamente en la industria, ya que brindan soluciones apropiadas a problemas complejos, de forma proactiva e inteligente. En particular, los Ambientes Inteligentes (AmI) educativos se benefician de estás dos arquitecturas, ya que por un lado los MAS dotan al AmI de inteligencia, mientras que SOA permite a los usuarios interactuar con servicios académicos en la nube. El propósito de este artículo es proponer una arquitectura de integración bidireccional SOA-MAS para AmI educativos. La solución propuesta aprovecha las ventajas de ambas tecnologías (SOA-MAS), y resuelve problemas de integración planteados en investigaciones previas.

Palabras clave: ambiente inteligente; SOA; MAS; integración; computación en la nube.

1. Introducción

En educación, la Inteligencia Ambiental permite aportar grandes beneficios, al adaptar el ambiente de forma dinámica a las necesidades de enseñanza-aprendizaje de los usuarios [1-5]; por ejemplo: brindando contenidos digitales basados en el estilo de aprendizaje de los estudiantes, adecuando el ambiente (aula) para realizar las actividades adecuadas al contexto educativo, poniendo en contacto estudiantes que estudien temas similares para fomentar el aprendizaje colaborativo, entre otras cosas.

Por otro lado, el aprendizaje en la nube (C-Learning) consiste en mantener y usar servicios educativos de manera distribuida, usando los mecanismos y herramientas que ofrece la computación en la nube [6-7]. Los servicios educativos clásicos que brinda C-Learning son cursos, actividades y contenidos. El C-Learning incluye actividades síncronas y asíncronas, para fomentar el aprendizaje colaborativo por medio del uso de la nube. C-Learning brinda servicios educativos disponibles en la nube, tal que los recursos educativos son almacenados en un ambiente virtual que puede ser accedido de maneras y ubicaciones distintas, por medio del del paradigma SOA [8-9].

En investigaciones pasadas se ha propuesto AmICL, un

How to cite: Sánchez, M., Aguilar, J. and Exposito, E., Integración SOA-MAS en ambientes inteligentes. DYNA, 85(206), pp. 268-282, September, 2018.

medio de gestión de servicios (middleware) para Ambientes Inteligentes Educativos basado en el aprendizaje en la nube [10-15]. La inteligencia de AmICL está soportada por un MAS cuya arquitectura es descrita en [11]. Para implementar este MAS se utiliza el estándar FIPA [16-17] por medio de un conjunto de agentes básicos descritos en [18]. Por otro lado, la arquitectura SOA permite a AmICL hacer uso de los servicios académicos de la nube en la búsqueda por mejorar el proceso de enseñanza-aprendizaje de los usuarios del ambiente.

Debido a que SOA y MAS [19] utilizan estándares y especificaciones diferentes [20-22], no es posible su comunicación de manera natural y directa. Mientras que un MAS utiliza protocolos FIPA para la comunicación, y más específicamente el lenguaje FIPA-ACL [16, 23-24], en SOA se utiliza generalmente el protocolo de comunicación SOAP [25-27], el cual es un estándar que define cómo objetos en diferentes procesos pueden comunicarse usando mensajes escritos en XML [28-29] con un protocolo de transporte como HTTP [30]. Es así que desde hace algún tiempo se vienen proponiendo soluciones que permiten la integración de ambas arquitecturas [20,31-37], de tal manera que éstas puedan descubrirse e invocar instancias de la otra, para aprovechar las ventajas de ambas tecnologías.

Por otra parte, en AmICL se hizo una primera propuesta que busca dar solución al problema planteado, a través de los agentes SMA y WSA definidos en [10]; Sin embargo, estos agentes solo permiten la integración de forma unidireccional entre el MAS y la nube (SOA). Esto significa que AmICL puede hacer uso de los servicios académicos en la nube, pero las aplicaciones externas basadas en SOA no pueden usar los servicios de AmICL, lo que limita que los servicios de la nube puedan integrarse con el ambiente, y a su vez, se limitan las capacidades de AmICL, ya que su proactividad e inteligencia depende en gran medida de estos servicios.

En definitiva, el objetivo fundamental de este artículo es analizar las posibles arquitecturas de integración MAS-SOA, que puedan ser usadas en AmICL, las cuales, a su vez, deben ser genéricas, tal que puedan ser de utilidad para cualquier sistema que requiera interconectar MAS y SOA.

En consecuencia, el artículo se organiza de la siguiente manera: En la sección 2 se especifica la arquitectura de AmICL, en la sección 3 se presenta el estado del arte en cuanto a que se ha hecho en integración MAS-SOA, para posteriormente, en la sección 4 presentar posibles soluciones de integración, analizando sus ventajas y desventajas, finalizando con algunas conclusiones y trabajos futuros.

2. Especificación de AmICL

2.1. Caracterización de AmICL

AmICL es un Middleware Reflexivo Autonómico [38-40] que permite la integración de los objetos (inteligentes o no) del ambiente educativo, con recursos educativos en la nube, de una manera flexible, proactiva y adaptativa, tal que estos objetos pueden ajustar su comportamiento de acuerdo al contexto y requerimientos del usuario (Ver Fig. 1, arquitectura multi-capa de AmICL), posibilitando ubicar, analizar y compartir el conocimiento).

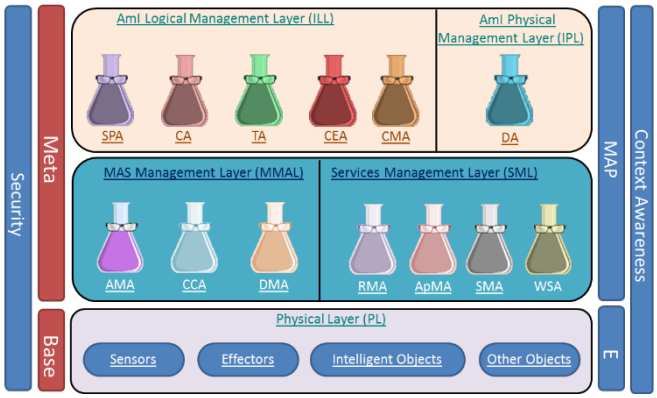


Figura 1. Arquitectura multi-capa de AmICL.
Source: [10].

En AmICL, los servicios académicos en la nube se combinan con objetos inteligentes que integran el ambiente, para adaptarse y responder a los requerimientos de enseñanza aprendizaje de los usuarios. Esto posibilita poner a disposición de los usuarios de forma inteligente, los servicios apropiados que permitan mejorar el desarrollo de las actividades académicas que realizan en un momento específico, adaptados a la forma de aprender del estudiante.

AmICL está compuesto por un MAS de cinco capas (Ver Fig. 1). La capa física (PL) representa los diversos dispositivos físicos del ambiente, definidos como agentes dispositivos en la capa de gestión de AmI Físico (IPL). PL está conectada con el Sistema Operativo, y trabajan en conjunto para realizar las diferentes actividades que se llevarán a cabo en el entorno. La capa de gestión de AMS (MMAL) está compuesta por una comunidad de agentes que soporta la ejecución de aplicaciones multi-agentes. Esta capa se basa en el estándar FIPA [18] y ha sido definida en artículos previos [10-11]; Ésta capa es una abstracción del estándar FIPA, donde el agente AMA representa al directorio facilitador (DF [18]), y el agente CCA sirve como punto de enlace para coordinar la comunicación entre los demás agentes, tanto internos como externos al MAS (ver [18] para más detalles de esos agentes). La capa de gestión de servicios (SML) es donde se registran, descubren, invocan, etc., los servicios requeridos por el ambiente en un momento determinado. La capa AmI lógico (ILL) representa los diferentes componentes de software que son usados en la plataforma educativa. En adición, esta capa tiene dos agentes, el agente perfil de estxcudiante (SPA) y el agente tutor (TA), que caracterizan a estudiantes y profesores, respectivamente.

Los detalles de AmICL pueden encontrarse en [8]. En general, los agentes del middleware son, en MMAL: el Agente Administrador de Agentes (AMA), Agente Controlador de Comunicación (CCA) y el Agente Gestor de Datos (DMA), este último encargado de poner a disposición de los demás agentes, los datos que requieran para llevar a cabo sus funciones; La SML está compuesta por el Agente Gestor de Servicios (SMA, se encarga de gestionar el directorio de servicios en la nube), El Agente de Servicio (WSA, un proxy que sirve para invocar servicios web), el Agente Gestor de Recursos (RMA, gestiona el uso de los recursos requeridos por los demás agentes) y el agente gestor

de aplicaciones (ApMA, encargado de gestionar el uso de las aplicaciones por parte de los demás agentes). Los agentes de CAL son: el Agente Perfil de Estudiante (SPA, caracteriza un estudiante en el AmI), el agente colaborativo (CA, posibilita el aprendizaje colaborativo entre estudiantes), el Agente Tutor (TA, caracteriza al profesor de la clase y sirve como un tutor virtual para el estudiante), el Agente de aplicaciones de Gestión de Contenido (CMA, gestiona el ambiente virtual de aprendizaje agregando dinámicamente los recursos recomendados de acuerdo al estilo de aprendizaje del estudiante), y el Agente de aplicaciones de Espacio Colaborativo (CEA, similar a CMA pero solo para aplicaciones de trabajo colaborativo). Finalmente, la IPL está compuesta de un único agente, el Agente Dispositivo (DA, representa los diversos dispositivos desplegados en el aula inteligente).

2.2. Subsistema de gestión de la nube en AmICL

En particular, el aspecto que permite unir el AmI con el paradigma de computación en la nube, se encuentra en SML. Esta capa contiene algunos agentes, que son una extensión del estándar FIPA (ver [18]), para gestionar tanto los recursos y aplicaciones presentes en el sistema operativo, como los servicios educativos en la nube, lo que permite que otros agentes de AmICL puedan usar servicios académicos en la nube, para mejorar las actividades de los usuarios del ambiente educativo. En particular los agentes RMA y ApMA manejan recursos y aplicaciones del sistema operativo en el MAS [18]; mientras que los agentes que representan un puente entre el AmI y la nube se detallan a continuación:

El Agente Gestor de Servicios (SMA) Controla, registra y administra los servicios web disponibles en el Sistema; de esta forma, cuando un agente del MAS requiere un servicio específico, debe contactar al SMA, para que éste ubique al Agente de Servicio (WSA) que caracteriza, en el MAS, al servicio web requerido. SMA caracteriza lo que se conoce dentro del tema de Servicios Web como UDDI [41] (Universal Description, Discovery and Integration, por sus siglas en inglés).

Por su parte, el Agente Servicio (WSA) es la representación lógica de un servicio web. Es decir, éste agente caracteriza a un servicio web (un agente por cada servicio web), por esa razón éste sabe cómo invocarlo (tiene su interfaz) y cuáles son los requerimientos necesarios para acceder al endpoint del servicio web. El WSA tiene acceso al archivo de descripción del servicio (WSDL [42], por sus siglas en inglés). El diseño de estos agentes se describe con mayor detalle en [10-11].

El mayor problema que tiene este subcomponente de AmICL es que solo permite la integración en forma unidireccional (desde AmICL hacia la nube), eso significa que el AmI no podrá exponer sus servicios a otros sistemas desarrollados bajo el paradigma SOA, limitando las capacidades de integración de AmICL. En esta investigación se analizarán dos propuestas de integración bidireccional SOA-MAS y se realizarán un conjunto de pruebas que permitirán determinar cuál de las dos presenta mayores ventajas en cuanto a escalabilidad, tiempo de respuestas, entre otras.

3. Trabajos relacionados

La integración entre MAS y SOA ha sido objeto de diversos estudios durante los últimos años [20,31-37,43-45]. Esas propuestas se dividen en dos grandes grupos: aquellas que utilizan agentes proxy (Gateway) y aquellas que son completamente orientadas a servicios.

3.1. Estudios basados en Agente Proxy

Las investigaciones realizadas en [20-31] permiten realizar composición y descubrimiento de servicios por medio de agentes. La arquitectura propuesta se basa en el uso de una pasarela (Gateway Proxy) para la integración de servicios web (WSIG, por sus siglas en inglés), que permite el registro y descubrimiento, tanto de agentes como de servicios web. El registro/deregistro es realizado por medio del componente "GatewayRegistry", éste a su vez utiliza el "Service Description Translator" para traducir de WSDL a ACL/OWL y viceversa, de tal forma que el servicio web (WS, por sus siglas en inglés) sea registrado en el DF como un agente, y los agentes sean registrados en el UDDI como servicios web. De esta manera, los agentes pueden hacer descubrimiento de servicios web a través del DF, mientras que los servicios web descubrirán a los agentes usando el UDDI.

Cuando un agente requiere invocar un WS, WSIG pasa el requerimiento al componente "Web Service Invocation", el cual traduce el mensaje a SOAP usando el "Message Translator", y lo envía al endpoint del WS, espera su respuesta, y la retorna al agente solicitante. De forma similar, para que un servicio web comunique a un agente su solicitud la pasa al componente "AgentService Invocation", el cual traduce el mensaje de SOAP a ACL, y lo envía al agente especificado, espera su respuesta en ACL, y una vez recibida, la remite al WS solicitante en forma de mensaje SOAP. Para que todo este proceso pueda llevarse a cabo, la plataforma debe contar con un servidor web y un UDDI, ejecutándose como servicios de JADE [46-47].

Por su parte, en [32] proponen AgentWeb Gateway, el cual es un middleware que permite la integración de MAS con SOA de forma muy similar a la realizada por [20,31], ya que se basa en los mismos principios. Su arquitectura está integrada por 3 componentes. El primero es el "Search query converter", el cual se encarga de procesar las búsquedas de agentes y servicios web, a fin de que puedan descubrirse entre ellos. El modulo "DF to UDDI" permite a los agentes descubrir servicios web registrados en el UDDI, mientras que su contraparte (UDDI to DF) permite a un servicio web ubicar a los agentes. El siguiente componente (Service Description Converter) se compone de dos módulos, el "DFAgentDesc to WSDL" permite a los agentes de software publicar sus servicios en el directorio UDDI de la plataforma web, a fin de que puedan ser consumidos por otros servicios web, mientras que el módulo "WSDL to DFAgentDesc" permite que un servicio web publique sus servicios en el DF del MAS, de tal manera que los agentes puedan consumir dichos servicios de forma transparente. El tercer componente, denominado "Communication protocol converter", permite la invocación entre agentes y servicios web. En particular, el módulo "ACL to SOAP" permite a los agentes del MAS

invocar un servicio web (traduce el mensaje de ACL a SOAP), mientras que el módulo "SOAP to ACL" permite a un servicio web invocar los servicios ofrecidos por los agentes, traduciendo los mensajes de SOAP a ACL.

Otra investigación basada en la integración por medio del uso de agentes Gateway es la que se realizó en [33]. WS2JADE permite a los agentes de JADE descubrir y consumir servicios web, con un alto grado de automatización al desplegar los mismos como agentes de JADE en tiempo de ejecución. A su vez, puede generar agentes a partir del archivo WSDL. De igual forma, WS2JADE permite el mapeo M a N entre agentes y servicios web, lo que posibilita que varios agentes puedan consumir un servicio web de forma concurrente. Las capas que componen la arquitectura de WS2JADE son de dos tipos, la capa de interconexión y la capa de administración. La primera capa contiene entidades (agentes de servicios web, ontologías y protocolos) que permiten interconectar agentes y servicios web. La capa de administración crea y gestiona dichas entidades de forma dinámica. Los agentes de servicios web (WSAG) son creados por la capa de administración de forma dinámica en tiempo de ejecución, para ello, el descubrimiento de servicios web se debe realizar de forma activa (mapea, registra, etc.). Los servicios web son vistos por los agentes de JADE como servicios ofrecidos por algún WSAG. Al ofrecer el mismo servicio web en diferentes WSAG, se logra el acceso concurrente a los servicios web.

Una solución para la integración MAS-SOA usando agentes proxy y Java Business Integration (JBI) es propuesta por [34]. La idea fundamental es crear un servicio web (servicio proxy) dentro del contenedor JBI, que actúe como proxy entre el NMR (Normalized Message Router) y el MAS, el cual pueda interactuar tanto con los servicios desplegados en el contenedor JBI como con los agentes de JADE. Para que el servicio proxy pueda comunicarse con los agentes en el MAS debe crearse un agente proxy en el MAS, que actuará como puente entre la comunidad de agentes y el servicio proxy. En consecuencia, esta propuesta contiene dos entidades fundamentales, el servicio proxy desplegado en el contenedor JBI, y el agente proxy desplegado en la plataforma Multiagentes; todo el tráfico entre servicios web y agentes debe pasar a través de estas dos entidades, que funcionan como los extremos de un puente de comunicación. De esta forma, cuando un servicio web en el JBI requiere usar las capacidades de un agente, enviará un mensaje al servicio proxy por medio del NMR, el servicio proxy transformará el mensaje a ACL, y lo remitirá al agente proxy asociado con él, el agente proxy lo enviará al agente especifico, esperará la respuesta, y la transmitirá nuevamente al servicio proxy en formato SOAP, el cual lo remitirá al servicio que hizo la solicitud inicialmente. Por otra parte, cuando un agente de la comunidad requiere usar un WS, envía un mensaje ACL al agente proxy, el cual lo transforma en un mensaje SOAP, y remite al servicio proxy que tiene asociado, quien debe enviarlo al servicio destino, esperar la respuesta, y retransmitirla nuevamente al agente proxy en ACL, para que éste la remita al agente que realizó la solicitud inicialmente. Es importante resaltar que el servicio proxy y agente proxy deben crearse manualmente.

En [36] se desarrolló una librería denominada JADE Gateway Library, que actúa como puente entre un MAS y los servicios que se encuentran desplegados en un Bus de Servicios Empresarial (ESB por sus siglas en inglés) [48]. La librería está compuesta por dos elementos fundamentales, el primero es el Jade-Gateway, que se encarga de gestionar los mensajes que se reciben desde y hacia el ESB; el otro es el Gateway Agent, que tiene como objetivo principal retransmitir los mensajes entre el MAS y el ESB. Cuando un agente quiere consumir un WS, debe enviar un mensaje al GatewayAgent, que a su vez lo retransmite al ESB; por otro lado, cuando un WS necesita enviar un mensaje a un agente, debe hacerlo a través del EntryPoint ESB Service, este lo pasará al Jade-Gateway, y luego al GatewayAgent, que lo retransmitirá al agente solicitado. Durante el proceso de comunicación el Jade-gateway debe realizar conversiones SOAP-ACL y ACL-SOAP, para que cada parte reciba el mensaje en el lenguaje apropiado. De igual forma, Jade-gateway resuelve el problema de integración de las comunicaciones síncronas de los servicios web con las comunicaciones asíncronas de un MAS, bloqueando el hilo del GatewayAgent hasta obtener la respuesta del MAS.

3.2 Estudios basados en Agentes Orientados a Servicios

Las investigaciones realizadas en [35,37,43] proponen SoMAS (Service oriented MultiAgent System) como una arquitectura completamente orientada a servicios, donde tanto agentes como servicios web utilizan SOAP como lenguaje de comunicación. En la arquitectura SoMAS, los agentes deben publicar sus servicios en el directorio de servicios, donde enviaran para ello el archivo WSDL correspondiente a los servicios que ofrece. Cuando un agente o servicio web requiere usar un servicio particular, debe buscarlo en el directorio de servicios, y una vez encontrado establecer la comunicación con el mismo por medio de mensajes SOAP. Esta arquitectura deja a un lado el estándar FIPA, ya que el sistema no utiliza DF ni el lenguaje de comunicación ACL, aunque el autor indica que se pueden usar ambas interfaces de comunicación, lo que supone hacer un trabajo adicional. SoMAS es la propuesta más relevante de este tipo, en la literatura.

Para finalizar esta sección, se presenta un resumen de los problemas encontrados en los trabajos estudiados. En general, las grandes desventajas de las propuestas son: algunas integran en una sola dirección, esto quiere decir que solo permite que los agentes puedan consumir servicios web (como WS2JADE). Otras, no logran traducir el modelo de comunicación con estado de los agentes al modelo de comunicación sin estado de los servicios web, lo que dificulta la comunicación asíncrona entre ambos paradigmas. En algunos casos, los servicios y agentes proxy deben crearse manualmente, y son muy dependientes de la arquitectura. De igual forma, las propuestas que no usan conversión ACL/SOAP dejan de ser FIPA-Compliant.

4. Arquitecturas propuestas

En la siguiente sección se describe la propuesta de integración SOA-MAS, en la cual se muestran dos enfoques de integración. Ambos enfoques son independientes, es decir cada uno define una forma de integración usando diferentes

paradigmas, componentes y tecnologías. Por esta razón es necesario realizar un análisis comparativo, que permita determinar cuál enfoque brinda mejores rendimientos, respecto a los trabajos relacionados, así como entre ellos.

En general, en AmICL (o cualquier SMA basado en la nube) es fundamental que los agentes puedan consumir servicios de la nube, ya que de ello depende que los usuarios puedan usar los servicios ofrecidos por C-Learning. De igual manera, exponer los servicios de AmICL hacía la nube es indispensable, para que otros sistemas puedan interactuar con el AmI (Ej: para describir los dispositivos en el AmI, conocer el contexto de aprendizaje, entre otras cosas). Así, las características que deben cumplir las propuestas de integración se enumeran a continuación:

1. Permitir las comunicaciones bajo ambos estándares (FIPA para comunicaciones agente-agente, FIPA ACL-SOAP y SOAP-FIPA ACL para comunicaciones agente-WS y WS-agentes).
2. Permitir la transformación de mensajes SOAP a FIPA-ACL y FIPA-ACL a SOAP.
3. Crear automáticamente los Agentes Gateway, que servirán como punto intermedio entre las comunicaciones MAS-SOA y SOA-MAS.
4. Permitir las comunicaciones síncronas y asíncronas, esto implica transformar las comunicaciones síncronas que se reciban de parte de los servicios web, en comunicaciones asíncronas para la comunidad con agentes, y viceversa.
5. Permitir el mapeo M a N entre agentes y servicios web. Esto significa: a) Que M agentes puedan consumir un mismo servicio web en el mismo instante, b) Que N servicios web puedan comunicarse con un mismo agente.
6. Especificar la arquitectura, independiente de su implementación.

4.1. Propuesta 1: basada en Agentes Proxy orientados a servicios

En la Fig. 2 se muestra la primera propuesta de integración, la cual permitirá a AmICL consumir los servicios de la nube por medio de agentes proxy convencionales (WSA), y exponer las capacidades de los agentes en el MAS como servicios, por medio de agentes proxy orientados a servicios (WSOA).

La arquitectura propuesta es una mezcla entre el enfoque de agente gateway convencional y el enfoque de agentes orientados a servicios (SoMAS). En la Fig. 2 se pueden ver los agentes que intervienen en el proceso de integración. El agente SMA se encarga de controlar el directorio de Servicios (UDDI), y todo lo referente al registro/de-registro, modificación y búsqueda de servicios web. Cada vez que un servicio web es registrado en el UDDI, se registra en el SMA un WSA que caracteriza dicho servicio web, usando para ello el módulo "WSDL Translator". Por otro lado, WSOA tiene el rol de servir de fachada a los agentes del SMA, para que sus capacidades puedan ofrecerse como servicios a través de la nube, y SRPA representa a cualquier agente del MAS que en momento determinado puede necesitar invocar un Servicio Web (en cuyo caso se llamará RA, ver Fig. 3-4); o a un agente que expone sus funcionalidades como servicios web (en este caso se denominará SA, ver Fig. 5).

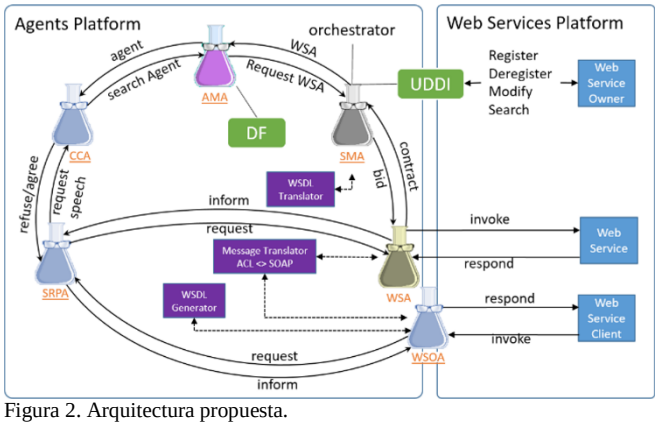


Figura 2. Arquitectura propuesta.
Source: Los autores.

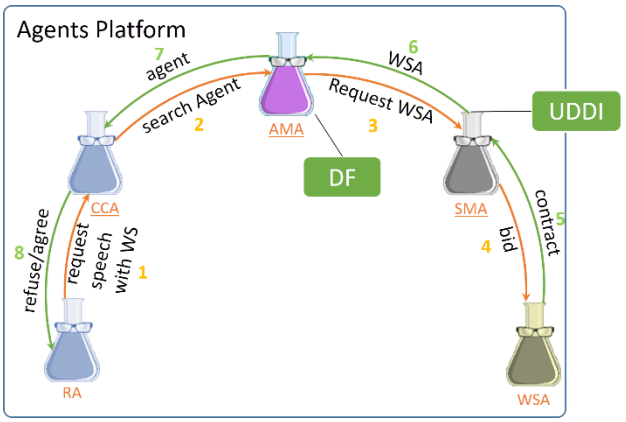


Figura 3. Descubrimiento de servicios web por agentes.
Source: Los autores.

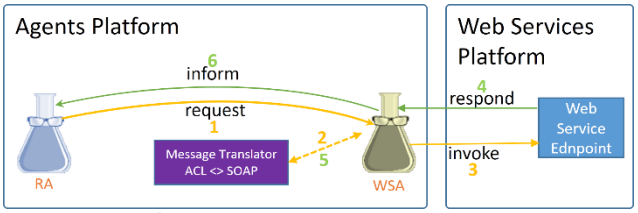


Figura 4. Invocación de servicios web por agentes.
Source: Los autores.

A diferencia de lo propuesto por [20,31-32], el WSA no se registra en el AMA (DF), ya que el encargado de administrar los agentes WSA es el SMA. Por su lado, el Agente AMA es el encargado de controlar el DF, donde se encuentran registrados los agentes de la plataforma Multiagentes; cuando un agente requiere descubrir a otro, AMA busca primero en el DF, si lo encuentra lo retorna, si no lo consigue en el DF, remite el requerimiento al SMA (ya que puede estar requiriendo un WSA). SMA buscará en su directorio, y de encontrarlo, retornará el WSA que sirve como proxy para consumir el servicio web requerido. De esta forma, un agente puede descubrir tanto a otros agentes (registrados en el DF), como a los servicios web disponibles en el ambiente (registrados en el UDDI). La Fig. 3 muestra los pasos seguidos en este proceso, las flechas de color

naranja (mensajes 1-4) muestran el camino seguido hasta encontrar el servicio web requerido, mientras que las flechas de color verde (mensajes 5-8) muestran como el servicio solicitado llega hasta el agente que hizo el requerimiento (RA), con lo cual ya pueden entablar comunicación.

Una vez que RA tiene la información del Agente que caracteriza al servicio solicitado, puede iniciar el proceso de comunicación para consumir el servicio web. En la Fig. 4 se detalla este proceso. Inicialmente RA hará el requerimiento para consumir el servicio web, enviando un mensaje ACL a WSA (1), WSA transformará el mensaje de ACL a SOAP usando el Módulo "Message Translator" (2), y retransmitirá el mensaje al endpoint del servicio web (3). Si es una comunicación asíncrona, WSA debe bloquear su hilo hasta recibir una respuesta del WS. Una vez recibida está respuesta (4), WSA transformará el mensaje de SOAP a ACL (5), usando nuevamente el módulo "Message Translator", y remitirá la información (6), al agente solicitante.

Por otro lado, como se dijo antes, los agentes WSOA (ver Fig. 2) sirven como fachada para exponer las capacidades de los agentes del MAS como servicios web. Estos agentes son creados de forma automática, cuando el agente al que sirven se registra en el AMA (1 WSOA por cada agente registrado en el AMA). El archivo WSDL de cada WSOA será registrado en el UDDI, usando para ello el módulo "WSDL Generator". Sin embargo, ningún WSA es creado en este proceso (no tiene sentido que los WSOA tengan un WSA que los caracterice como servicios web en el MAS), de esta manera, los WS externos pueden descubrir a los WSOA (y por ende, al agente al que sirven de fachada) realizando una búsqueda en el UDDI. Los servicios ofrecidos por cada WSOA deben ser desplegados en un servidor web, o por medio de un framework como Apache CXF [49], lo cual debe ser decidido al momento de la implementación. La Fig. 5 detalla tanto el proceso de descubrimiento de los WSOA por parte de un servicio web (1), como el proceso seguido para consumir los servicios expuestos por los agentes del MAS.

Una vez que el "Web Service Client" ha encontrado al agente WSOA que expone los servicios del agente SA por medio del UDDI, puede invocar dichos servicios, enviando un mensaje SOAP al WSOA (2), el WSOA tomará el requerimiento y lo transformará en un mensaje ACL (3), usando para ello el modulo "Message Translator", para luego remitirlo al agente (SA) (4). Debido a la naturaleza asíncrona de las comunicaciones entre agentes, WSOA debe esperar hasta recibir respuesta del SA (5), y una vez que tenga la información transformarla en un mensaje SOAP (6) por medio del "Message Translator", y reenviarla al "Web Service Client" (7) que realizó la solicitud inicialmente.

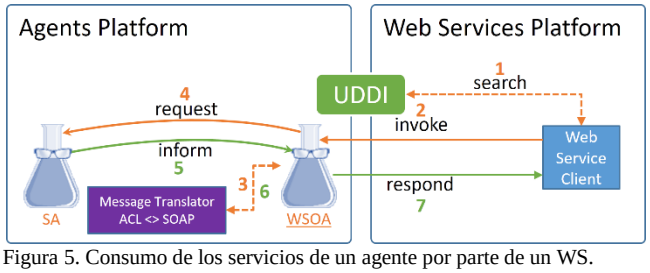


Figura 5. Consumo de los servicios de un agente por parte de un WS.
Source: Los autores.

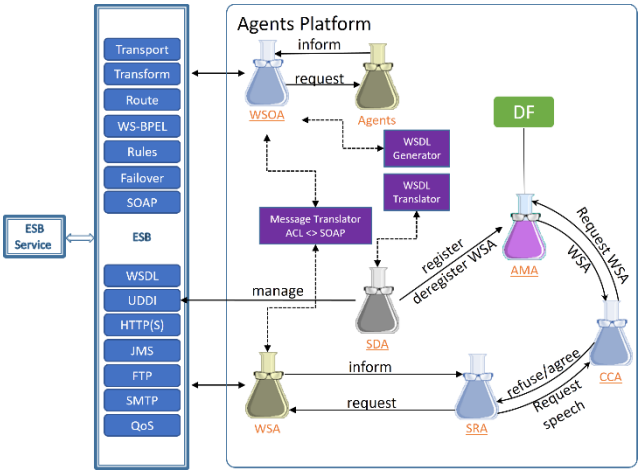


Figura 6. Arquitectura propuesta.
Source: Los autores.

4.2. *Propuesta 2: basada en un ESB usando Agentes Proxy orientados a servicios*

La arquitectura de ésta segunda propuesta se muestra en la Fig. 6. Ésta propuesta no solo permite aprovechar las características del enfoque de agentes Gateway y el enfoque SoMAS, sino que además incorpora elementos importantes del paradigma orientado a servicios, puesto que la integración se da por medio de un Bus de Servicios Empresarial (ESB, por sus siglas en inglés).

El ESB brinda al componente de la nube de AmICL características importantes, tales como: transporte, transformación y ruteo de mensajes, de esta manera, se pueden utilizar múltiples protocolos de transporte como HTTP(S)/REST [50], FTP [51], SMTP [52], etc. y estandarizarlos en un solo protocolo como SOAP, lo cual mejora en gran medida la integración con otros sistemas.

En la Fig. 6 se observan los elementos de la arquitectura que permiten lograr la integración MAS-SOA. Los agentes WSOA y SMA tienen las mismas funciones de la propuesta 1, sin embargo, en esta propuesta existe un nuevo agente denominado SDA (Service Discovery Agent), encargado de administrar el UDDI del ESB, y ejecutar los cambios correspondientes en el MAS. Así mismo, el SRA representa agentes del AMS que requieren consumir servicios web registrados en el ESB, mientras que "Agents" se refiere a todos aquellos agentes del MAS que exponen sus funcionalidades como servicios web usando un WSOA como fachada.

En las Fig. 7-11 se detallan los procesos básicos que se realizan en esta arquitectura: Registro de un servicio web, Descubrimiento de servicios web por agentes, Invocación de servicios web por agentes, Registro de agentes del MAS y WSOA, y Consumo de los servicios de un agente por parte de un WS. Vemos que, en esta arquitectura, hay dos procesos básicos extras. Por otro lado, el SRA representa los agentes RA que consumen los servicios web (ver Fig. 8-9), mientras que "Agents" son los agentes SA que exponen sus funcionalidades como servicios web (ver Fig. 10-11).

En la Fig. 7 se detalla como un Proveedor de servicios publica sus servicios en el ESB (1), el ESB lo registra en el

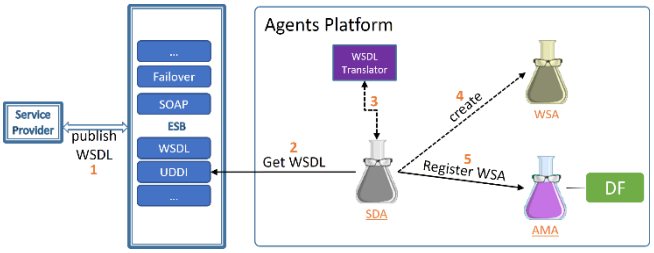


Figura 7. Registro de un servicio web.
Source: Los autores.

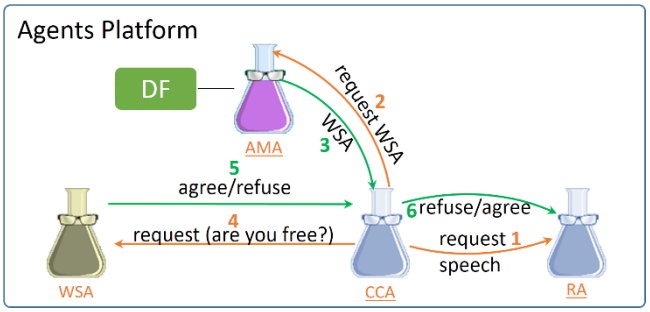


Figura 8. Descubrimiento de servicios web por agentes.
Source: Los autores.

UDDI, el SDA, al detectar el nuevo registro debe obtener el archivo WSDL correspondiente al servicio web registrado (2), utilizar el módulo "WSDL Translator" para obtener la descripción de un agente que caracterice a dicho servicio (3), para luego crear el agente WSA (4), y registrarlo (5) en el directorio facilitador administrado por el AMA. De forma similar, el SDA respondería cuando se modifica o de-registra un servicio web, efectuando los cambios necesarios en la plataforma.

Debido a que los WSA son registrados en el AMA, los agentes del MAS pueden descubrir sus servicios, como si de otro agente se tratase (buscando en el AMA). En la Fig. 8 se detalla el proceso.

En primera instancia, el agente RA debe preguntar (1) al CCA si es posible establecer comunicación con un WSA (CCA siempre está presente al inicio de la comunicación entre dos agentes, y es quien controla si los agentes pueden entablar una conversación o no). CCA debe buscar si el agente solicitado se encuentra registrado en el AMA (2). De ser así, el AMA responderá con la referencia del agente solicitado (3); CCA preguntará al WSA si está disponible para ejecutar una tarea (4), a lo cual el WSA puede responder afirmativo o negativo (5), dependiendo de esta respuesta, el CCA permitirá o no el inicio de la conversación entre el RA y el WSA (6).

Una vez que el RA tenga permiso para hablar con el WSA, puede invocar sus servicios (ver Fig. 9). Para ello, envía un requerimiento al WSA (1) por medio de un mensaje ACL. WSA usa el módulo "Message Translator" para transformar el mensaje de ACL a SOAP (2), y lo remite al ESB (3), el ESB se encarga de entregar dicho mensaje al endpoint del WS requerido (4 y 5), el WS procesa el mensaje y envía la respuesta al ESB (6) en un mensaje SOAP. El ESB se encarga de entregar este mensaje (en comunicaciones síncronas, WSA bloquea su hilo

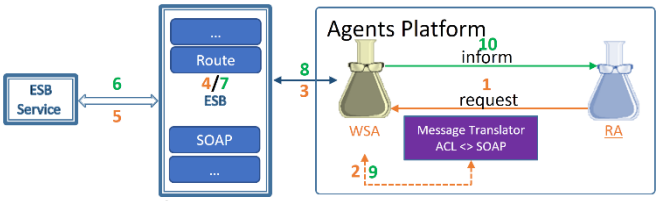


Figura 9. Invocación de servicios web por agentes.
Source: Los autores.

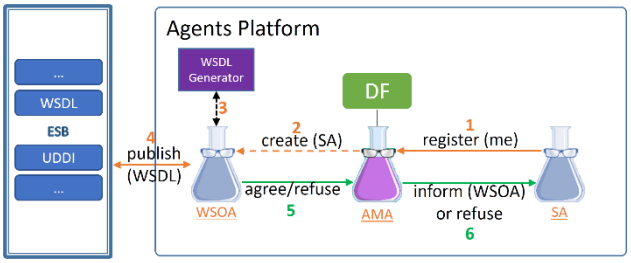


Figura 10. Registro de agentes del MAS y WSOA.
Source: Los autores.

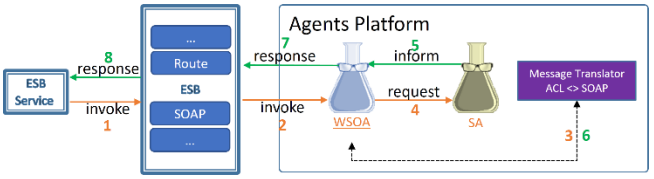


Figura 11. Consumo de los servicios de un agente por parte de un WS.
Source: Los autores.

hasta obtener respuesta del ESB) al WSA solicitante (7 y 8), quien usa nuevamente el "Message Translator" para transformar (9) el mensaje SOAP a ACL, y remitirlo al agente que realizó el requerimiento inicialmente (10).

Por otro lado, cuando un agente del MAS es registrado en el AMA, se debe crear automáticamente el WSOA (excepto si el agente que se registra es un WSA) que servirá de fachada para exponer sus capacidades como servicios web en el ESB (el mensaje de registro enviado a AMA, debe contener información que indique cuando crear o no el WSOA). La Fig. 10 refleja cómo se realiza esta tarea. En primer lugar, el agente del MAS (SA) solicita registrarse en el AMA (1); AMA lo registra en su DF y al conocer que el agente requiere un WSOA, lo creará (2). Una vez que el WSOA es creado, utiliza el módulo "WSDL Generator" para obtener un archivo WSDL (3) correspondiente a las capacidades del SA, que serán expuestas como servicios web y publicadas (4) en el ESB (WSOA guarda la referencia del SA al cual presta servicios, ya que lo necesitará para consumir sus servicios). WSOA responderá al AMA si fue posible o no publicar los servicios en el ESB, y dependiendo de la respuesta, AMA informará al SA quien es su WSOA o rechaza el registro del agente.

Debido a que los agentes WSOA tienen registrados sus servicios en el UDDI del ESB, cualquier servicio web que esté acoplado al ESB puede descubrir esos servicios por medio del mismo ESB. De esta manera, servicios web del ESB pueden invocar las capacidades de los agentes del MAS (ver Fig. 11).

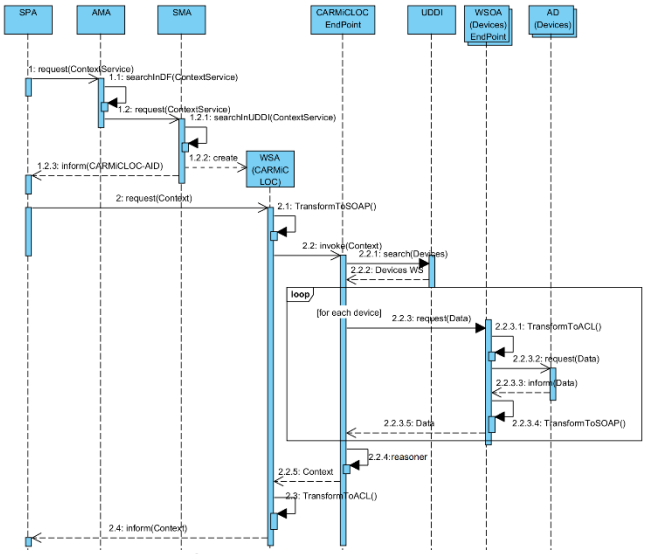


Figura 12. Conversación entre SPA y CARMiCLOC de la Propuesta 1. Source: Los autores.

El proceso se inicia cuando el ESB recibe un mensaje (1) para invocar un servicio de agente específico, el ESB se encarga de entregar el mensaje al endpoint del WS perteneciente al WSOA que expone ese servicio (2), WSOA transforma el mensaje de SOAP a ACL (3) usando el "Message Translator", y remite el resultado al agente SA al cual sirve de fachada (4). En comunicaciones síncronas, WSOA quedará en espera (bloqueo del hilo de ejecución), hasta que SA emita una respuesta. SA procesa los datos e informa el resultado al WSOA (5), el cual debe transformar nuevamente el mensaje de ACL a SOAP (6) usando el "Message Translator", y remitir el mensaje ya en SOAP al ESB (7), para que éste se encargue de entregarlo al servicio web que realizó el requerimiento inicialmente.

5. Experimentación

5.1. Instanciación de los dos enfoques en un caso de estudio

Para validar las propuestas de integración MAS-SOA presentadas en la sub-sección anterior, se presenta un caso de estudio, en el cual uno de los agentes de AmICL (SPA, Agente Perfil de Estudiante por sus siglas en inglés) descubre e invoca un servicio web de consciencia contextual (denóminado CARMiCLOC [53]), el cual a su vez, debe descubrir y comunicarse con los dispositivos presentes en el AmI (los cuales están caracterizados como agentes dispositivos en AmICL), para poder caracterizar el contexto (eso le permite a CARMiCLOC, descubrir y modelar el contexto). De esta manera, en este caso de estudio se requiere una comunicación bidireccional entre MAS y SOA (Agente consumen servicios web por medio de WSA, y los servicios web usan las funcionalidades de los agentes por medio de los WSOA). La información del contexto recibida por SPA puede usarse para tomar decisiones que permitan mejorar las actividades del estudiante en el AmI, de acuerdo a la situación del momento.

Se asume que CARMiCLOC ya se encuentra registrado en el SMA, así como los dispositivos del ambiente (camáras, pizarras electrónicas, sensores, etc.) caracterizados como agentes dispositivos (DA), y expuestos como servicios web a través de los WSOA, los cuales publican su archivo de descripción de servicios en el UDDI de la plataforma de servicios web. La conversación que se da entre agentes del AMS y los servicios web en la nube para la Propuesta 1, se muestra en la Fig. 12.

Inicialmente, el agente SPA hace una solicitud al agente AMA (DF), para ubicar un agente que provea un servicio de consciencia de contexto (1); AMA realiza la búsqueda en su directorio (1.1), y no lo encuentra debido a que no existen agentes con esas capacidades en el AMS, y delega la búsqueda al SMA (UDDI), para que verifique si existen servicios que provean consciencia contextual (1.2). SMA busca en su directorio (1.2.1) y consigue a CARMiCLOC, y crea un WSA que caracteriza a este servicio web (1.2.2). Este WSA se encarga de servir como proxy entre el SPA y el mismo CARMiCLOC; el Id del Agente (AID) de este WSA se envía al SPA que realizó la solicitud inicialmente (1.2.3). Nótese que todas estas conversaciones se dan en FIPA-ACL, ya que todas las entidades involucradas son agentes. Una vez que SPA tiene el AID del WSA que brinda el servicio de consciencia contextual, realiza un requerimiento de contexto (2) al WSA; por su parte, el WSA recibe el mensaje, y como está en FIPA-ACL realiza la transformación a SOAP (2.1), y lo remite al EndPoint de Servicio Web de CARMiCLOC (2.2). Como CARMiCLOC necesita tomar datos en AmICL para descubrir y modelar el contexto, debe comunicarse con los sensores y dispositivos presentes en el ambiente que le permitan realizar tal acción. Para ello, debe invocar los servicios de los Agentes que caracterizan dichos dispositivos en AmICL, que son los agentes DA. CARMiCLOC puede descubrir los DA como servicios, realizando una búsqueda de Servicios de Dispositivos en el UDDI de AmICL (2.2.1), donde encontrará los Endpoint de servicio web de los WSOA que sirven de fachada a los DA (2.2.2). De esta forma, CARMiCLOC se comunica con cada WSOA para obtener información del ambiente (2.2.3); cada WSOA transforma el mensaje de SOAP a ACL (2.2.3.1), y lo remite al DA (2.2.3.2), que es quien realmente obtiene la data del dispositivo físico, y una vez que obtiene los datos, los retorna a su WSOA en un mensaje ACL (2.2.3.3), el cual después lo transforma a un mensaje SOAP (2.2.3.4) y lo retorna al EndPoint de CARMiCLOC (2.2.4). CARMiCLOC le entrega el contexto al WSA proxy de CARMiCLOC (2.2.5), quien se encuentra esperando por el resultado. Como WSA recibe los datos en SOAP, transforma el mensaje a ACL (2.3), y remite la información al SPA (2.4), el cual lo usará para razonar y tomar decisiones, de acuerdo a la situación que indique el contexto en ese momento.

Por otra parte, la Fig. 13 muestra la conversación entre los agentes y los servicios web, para el mismo caso de estudio, usando ahora la Propuesta 2.

En este caso, el agente SPA envía un mensaje al AMA (1) para localizar un agente que provea servicios de consciencia contextual. En esta propuesta, en AMA se encuentran registrados tanto los agentes del MAS como los agentes de servicios web (WSA). Es así que AMA busca en su directorio

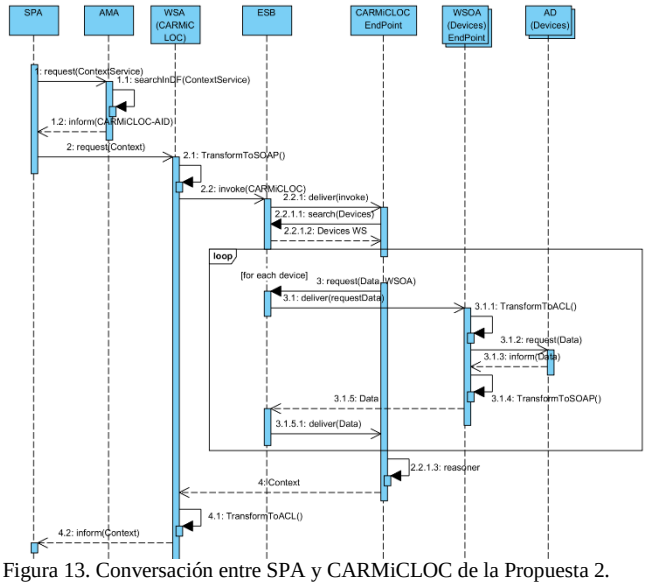


Figura 13. Conversación entre SPA y CARMiCLOC de la Propuesta 2. Source: Los autores.

de servicios (1.1), y consigue al agente WSA que caracteriza al servicio web CARMiCLOC como un agente, el cual es retornado a SPA (1.2). Una vez que SPA conoce cual agente puede ayudarlo a reconocer el contexto, realiza un requerimiento de contexto a dicho agente (2), WSA, recibe la solicitud en FIPA-ACL y la transforma a SOAP (2.1), para remitirla al ESB (2.2), el cual se encargará de entregar el mensaje al EndPoint de CARMiCLOC (2.2.1), quien actúa de la misma forma que en la conversación de la Fig. 12 (el comportamiento de CARMiCLOC es el mismo para ambas propuestas). Ahora bien, todo mensaje enviado por CARMiCLOC será puesto en el ESB, y este se encargará de transportarlo a su destinatario. Es así como CARMiCLOC inicia el descubrimiento de los servicios web de dispositivos presentes en el ambiente (2.2.1.1), y es el ESB quien usa su UDDI para localizar los servicios y entregar a CARMiCLOC los Endpoint de los WSOA que sirven de fachada a todos los dispositivos en AmICL (2.2.1.2). Con esta información, CARMiCLOC inicia la solicitud de datos, para recabar la información del ambiente que le permita descubrir el contexto, poniendo un mensaje en el ESB (3) para solicitar la información a cada WSOA. El ESB entregará el mensaje al WSOA correspondiente (3.1), quien a su vez, lo transformará a un mensaje ACL (3.1.1), y lo remitirá al AD que caracteriza (3.1.2). El AD tomará los datos requeridos del dispositivo físico, y los retornará a su WSOA en un mensaje ACL (3.1.2). Por su parte, el WSOA transforma el mensaje a SOAP (3.1.4) y lo pone en el ESB (3.1.5), para que sea entregado a CARMiCLOC (3.1.5.1). Una vez que éste obtiene toda la información necesaria, inicia el proceso de modelado (2.2.1.3), para finalmente distribuir el contexto al WSA correspondiente (4), el cual está a la espera de la respuesta. El WSA toma el mensaje recibido en SOAP y lo transforma a ACL (4.1), enviando el resultado al SPA (4.2) que inició el proceso.

5.2. Comparación preliminar entre los dos enfoques

Las propuestas presentadas en las subsecciones anteriores, mezclan los enfoques de integración discutidos en la sección 3. (Agente Proxy y SoMAS). La diferencia principal entre ambas, radica en que la segunda propuesta utiliza un ESB, que se hará cargo de enrutar los mensajes desde el endpoint solicitante hacia el enpoint que presta el servicio, sin vincular el emisor del mensaje con el recibidor (acoplamiento débil); y de hacer transformaciones de mensajes, posibilitando la integración con otras tecnologías web, como es el caso de los servicios web REST, cuyo protocolo de comunicación no es SOAP. De igual manera, el ESB tiene componentes para brindar seguridad a las comunicaciones, así como para asegurar la calidad del servicio (QoS), entre otras cosas. Así, la segunda propuesta podría brindar más beneficios en la integración de MAS con SOA.

Sin embargo, en la primera propuesta el número de mensajes intercambiados y elementos que intervienen en las comunicaciones generalmente es menor; Por ejemplo, en las conversaciones de la Fig. 12 (primera propuesta) y Fig. 13 (segunda propuesta), para la primera propuesta se intercambian 13 mensajes, mientras que en la segunda se requieren 15, lo que podría indicar que las comunicaciones en la primera propuesta tendrán mejor rendimiento. Con respecto a los requerimientos de cómputo, en ambas propuestas los agentes WSOA, y WSA tienen los mismos requerimientos computacionales; sin embargo, los agentes AMA y SMA de la primera propuesta requerirán un mayor costo computacional al momento de realizar las búsquedas, pero equivalente al necesitado por el AMA de la segunda propuesta, ya que las tareas del SMA de la primera propuesta pasan a formar parte del AMA en la segunda propuesta. Adicionalmente, el agente SDA de la segunda propuesta requerirá revisar periódicamente el UDDI del ESB, para determinar cambios y hacerlos efectivos en el MAS. De forma similar, el ESB requerirá un costo computacional adicional al momento de usar elementos de ruteo, transformación, entre otros;

Por otra parte, la primera propuesta utiliza el agente SMA, cuyas funciones son administrar el UDDI y los agentes WSA (creación, búsqueda, composición, orquestación de WSA, entre otros); en cambio, en la segunda propuesta muchas de esas tareas son realizadas por el propio ESB (composición, orquestación, etc.), y en parte por el SDA (creación, modificación, entre otros), pero la ubicación de WSA queda delegada al AMA, tal que los agentes del MAS pueden ubicar a los WSA como si de otro agente de la plataforma se tratase.

En ambas propuestas, el agente CCA es el encargado de definir cuando un agente del MAS puede establecer comunicación con un WSA, dependiendo de la ocupación que éste tenga. Además, es él fundamental para garantizar el mapeo M a N entre servicios y agentes. Los objetivos y tareas de los agentes WSOA y WSA en ambas propuestas son los mismos. El WSOA sirve como puente en la dirección SOA-MAS, mientras que el SMA es el puente de comunicación en la dirección MAS-SOA.

En general, analizando criterios como pase de mensajes y agentes participantes en las propuestas, para los casos de descubrimiento y consumo de servicios web y uso de las capacidades de los agentes desde un servicio web (son los casos más importantes), tenemos que:

Para que un agente pueda descubrir un servicio web, en la primera propuesta se utilizan 8 pasos, mientras que en la segunda se requieren 6 (ver las Figs. 3, 8, 12, 13). Además, en la primera propuesta intervienen 5 agentes, mientras que en la segunda intervienen 4. Finalmente, en la primera propuesta los WSA se deben buscar tanto en el DF como en el UDDI, mientras que en la segunda solo se busca en el DF (ver Figs. 3, 8).

Con respecto a la invocación del servicio web en sí, en la primera propuesta se utilizan 6 pasos mientras que en la segunda se utilizan 10 (ver las Figs. 4, 9, 12, 13). De forma similar, en la primera propuesta intervienen 2 agentes, los módulos de transformación de mensaje y el servicio web cliente, mientras que en la segunda intervienen, además, el ESB y sus componentes de ruteo y transformación de mensajes.

Para que un WS haga uso de una tarea de un agente (consuma), en la primera propuesta se utilizan 8 pasos e intervienen 2 agentes y el servicio web cliente; mientras que en la segunda propuesta se utilizan 8 pasos, intervienen 2 agentes, el WS cliente, y los componentes del ESB de ruteo y transformación de mensajes (ver Figs. 5, 11-13).

5.3. Simulaciones

Con el fin de realizar una comparación cuantitativa de ambas propuestas en términos de rendimiento, se realizaron un conjunto de simulaciones. El modelo de simulación comprende los nodos necesarios para medir el comportamiento del sistema real. Estos nodos puede ser agentes (SMA, WSA, WSOA, Otros Agentes del Sistema, etc.) o elementos de hardware y software como: internet, red local, traductor de mensajes, endpoint de servicios web, entre otros. Las variables consideradas en las simulaciones son las siguientes: a) Número de agentes de AmICL que ofrecen sus funcionalidades como servicios b) Número de servicios web registrados en el UDDI (Servicios en la nube usables por los agentes) c) Número máximo de agentes WSA soportados por la plataforma, el cual depende de los recursos del SMA d) Tiempo medio de generación de nuevos requerimientos de servicios web, por parte de los agentes (Agente→WS) e) Tiempo medio de generación de nuevos requerimientos de servicios a los agentes, por parte de los servicios web (WS→Agente).

También existen otras variables en la simulación que se mantuvieron constantes, ya que afectan a ambas propuestas por igual e influirán de la misma manera en ambos casos. Estás variables son: a) Tiempo medio de atención de requerimientos por parte de los servicios web (tiempo que tarda un servicio web en atender una solicitud) b) Desviación estándar del tiempo de atención de requerimientos por parte de los servicios web c) Tiempo medio de atención de requerimientos por parte de los agentes (tiempo que tarda un agente en atender una solicitud) d) Desviación estándar del tiempo de atención de requerimientos por parte de los agentes e) Tiempo promedio de atención de solicitudes del agente SMA f) Tiempo promedio de traducción de mensaje (SOAP-ACL, ACL-SOAP) g) Latencia de Internet h) Latencia de la red local.

Se diseñaron tres escenarios de simulación que se detallan más adelante. Para todos los escenarios se estableció un tiempo de simulación de 1800 segundos y un máximo de agentes WSA de 2000, es decir, la plataforma SMA no puede crear más de 2000 agentes WSA. Eso significa que, si hay muchas invocaciones concurrentes de servicios web, y se supera el valor máximo de WSA soportados, las solicitudes siguientes serán rechazadas, hasta que el número de WSA baje nuevamente.

Las métricas consideradas en estos escenarios, son las siguientes: a) **Tiempo total de atención promedio de los servicios en la nube**: es el tiempo promedio que dura una solicitud de servicio web en ser procesada, esto incluye el pase de mensaje entre el agente solicitante y el SMA, la búsqueda en el UDDI, el pase de mensajes entre el agente solicitante y el WSA, la invocación del servicio, el envío de los datos a través de internet, la traducción de los mensajes (ACL/SOAP, SOAP/ACL), y el retorno de los resultados por parte del servicio. b) **Tiempo total de atención promedio de las tareas de agentes como servicios**: Es el tiempo promedio que tarda una solicitud a una tarea de agente en ser procesada, esto incluye el tiempo de búsqueda del servicio de agente en el UDDI, el pase de mensajes a través de internet desde los servicios de la nube hasta los agentes WSOA, el pase y traducción de mensajes (SOAP/ACL, ACL-SOAP) entre el WSOA y el agente destino, y el cálculo y el retorno de los resultados hasta el servicio web solicitante. c) **Número de WSA creados**: Indica el número de agentes WSA que se deben crear para atender todas las solicitudes que llegan al sistema, lo cual depende directamente de los recursos del propio SMA. Esta variable nos ayuda no solo a medir la cantidad de recursos que se necesitan para poner en marcha el sistema, sino, además, cómo se comporta la concurrencia en la invocación de servicios. d) **Otras variables**: Para realizar análisis más exhaustivos de las simulaciones, también se tomarán en cuenta variables, como longitud de la cola de espera en los nodos, el tiempo de atención de cada uno de los nodos de la simulación, así como la cantidad de solicitudes atendidas en cada nodo.

Escenario 1: En este escenario se probará la comunicación en el sentido MAS-SOA, donde son los agentes del sistema los que invocan servicios de la nube por medio del uso de los mecanismos de integración presentados en las dos propuestas. Se probará cómo se comporta el sistema con pocos servicios registrados (10), regular cantidad de servicios registrados (80), y muchos servicios registrados (1000). También se variará el número de solicitudes por segundo, empezando con 2 solicitudes por segundo, luego 4 solicitudes por segundo, 8 solicitudes por segundo, y terminando con casos extremos de 50 y 100 solicitudes por segundo. Al utilizar muchas solicitudes por segundo, también se prueba el sistema ante llamadas concurrentes de servicios, de tal forma que podamos observar cómo crece la cantidad de agentes WSA para poder atender todas las solicitudes.

En la Fig. 14, se muestra la comparación de ambas propuestas, tomando en cuenta el tiempo de atención total promedio de cada solicitud de servicio hecha por parte de los agentes.

Se puede observar claramente, que los tiempos de atención siempre son mayores en la segunda propuesta y a medida que aumenta el número de solicitudes de servicios

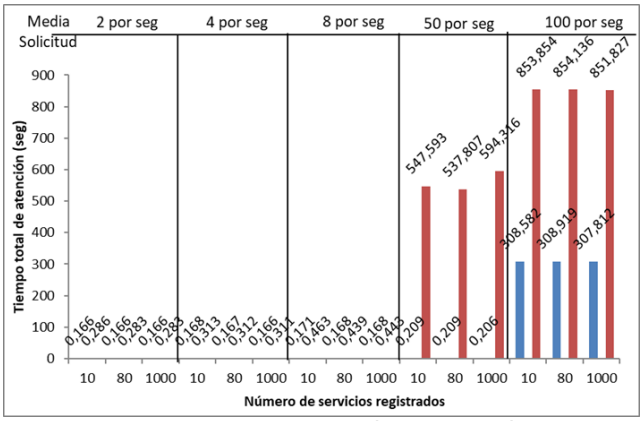


Figura 14. Tiempo total promedio de atención respecto al número de servicios en el UDDI (azul propuesta 1, rojo propuesta 2).
Source: Los autores.

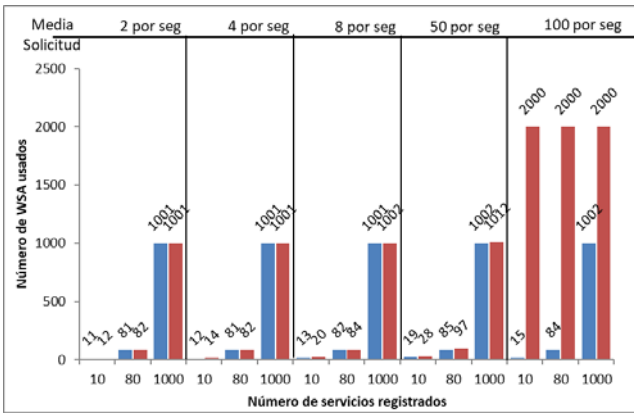


Figura 15. WSAs requeridos respecto del número de servicios en el UDDI (azul propuesta 1, rojo propuesta 2).
Source: Los autores.

por parte del SMA, aumenta también el tiempo de atención total de las solicitudes, siendo mayor la discrepancia en el caso de la Propuesta 2. También se logra apreciar que ambas propuestas son independientes del número de servicios registrados en el UDDI con respecto al tiempo de atención, ya que el tiempo se mantiene más o menos constante para 10, 80 y 1000 servicios para una misma media de solicitud. Por ejemplo, para una media de 2 solicitudes por segundos, el tiempo total de atención promedio en la primera propuesta siempre es de 0,166; mientras que para la segunda propuesta es de 0,283; independientemente del número de servicios registrados en el UDDI. Eso significa que podemos agregar o quitar servicios del UDDI, sin afectar el rendimiento de la plataforma.

De la Fig. 14 también se observa que los tiempos de atención promedio se mantienen por debajo de 1 segundo en ambas propuestas, para medias de solicitud de 2, 4 y 8 solicitudes por segundo, lo cual es bueno. Sin embargo, para los casos extremos de 50 y 100 solicitudes por segundos, se observa que para la segunda propuesta el tiempo total de atención promedio, crece considerablemente, tomando valores de más de 500 y 800 segundos, respectivamente; mientras que la primera propuesta solo se ve afectada cuando la media de solicitudes es de 100 solicitudes por segundo.

Analizando un poco más los resultados de la simulación, se aprecia, en el caso de la primera propuesta, que el nodo que representa la comunicación con la nube (internet) se encuentra sobresaturado debido a la gran cantidad de datos enviados por el canal de comunicación, mientras que, para la segunda propuesta, están sobresaturados tanto el nodo de comunicación con la nube como el nodo que representa al ESB, lo cual deriva en un incremento del tiempo total de atención promedio.

Por otra parte, en la gráfica de la Fig. 15, se analiza la dependencia de la plataforma, con respecto al número de agentes WSA necesarios para atender las solicitudes de servicios (como se indicó anteriormente, se estableció un límite superior de 2000 WSAs).

En este caso, se puede apreciar que ambas propuestas no dependen en gran medida de los agentes WSA, pero si requieren tener suficientes recursos para contar con al menos un WSA por cada servicio registrado en el UDDI, puesto que la cantidad de WSA adicionales que se requieren para atender solicitudes en paralelo en este escenario es poco. Note que si hay 10 servicios registrados se requieren 12 o 13 WSA para cada propuesta, para 80, se requieren entre 81 y 86, y para 1000 se requieren menos de 1015 agentes WSA, manteniéndose estos valores prácticamente independientes del número de solicitudes de servicio, lo que también indica que el sistema puede lidiar con la concurrencia en la invocación de servicios, sin ningún problema. El único caso donde se presenta una variación considerable del número de agentes WSA, es para la segunda propuesta cuando la media de solicitudes es de 100 solicitudes por segundo. En este caso, el sistema alcanza el valor máximo establecido de 2000 WSAs.

Al analizar en detalle el proceso de simulación, se nota que al estar congestionados el nodo ESB y el nodo de comunicación con la nube, se empiezan a generar muchas solicitudes paralelas que se van acumulando, hasta alcanzar el límite de 2000 WSA, debido a que ningún WSA termina su requerimiento (el tiempo total de atención promedio es superior a 800 segundos), y se están generando 100 solicitudes cada segundo. A partir de allí el sistema empezará a denegar las solicitudes de servicio, puesto que no se cuentan con los recursos para atenderlas.

Escenario 2: En este segundo escenario se busca probar la comunicación SOA-MAS, donde los servicios en la nube invocan las tareas de los agentes del sistema como servicios. Se probará cómo se comporta el sistema con pocos agentes que ofrecen sus capacidades como servicios (10), una cantidad regular (80), y con muchos agentes (1000). De igual manera, se variará el número de solicitudes por segundo, tomando valores de 2, 4, 8, 50 y 100 solicitudes por segundo.

La Fig. 16, muestra una gráfica en la cual se puede deducir rápidamente que la propuesta 2 tiene problemas con el tiempo promedio total de servicio de los agentes, cuando el número de solicitudes por segundo es superior a 8, llegando a superar los 400 segundos, independientemente del número de agentes que ofrecen sus tareas como servicios, lo cual es inconcebible para sistemas que requieren una respuesta inmediata.

De forma similar, la propuesta 1 presenta problemas cuando la media de solicitudes es de 100 solicitudes por segundo,

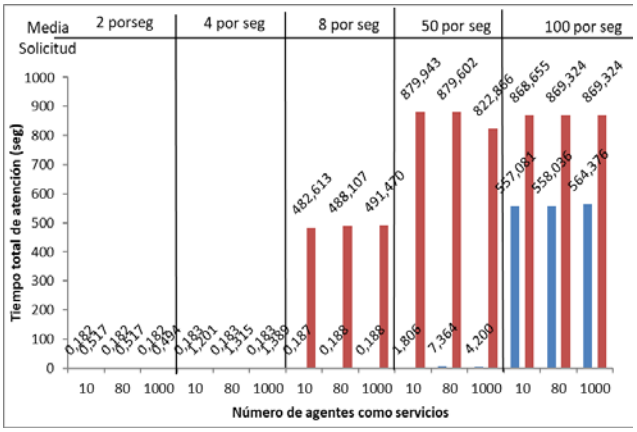


Figura 16. Tiempo de atención promedio respecto del número de agentes que ofrecen tareas como servicios (azul propuesta 1, rojo propuesta 2).
Source: Los autores.

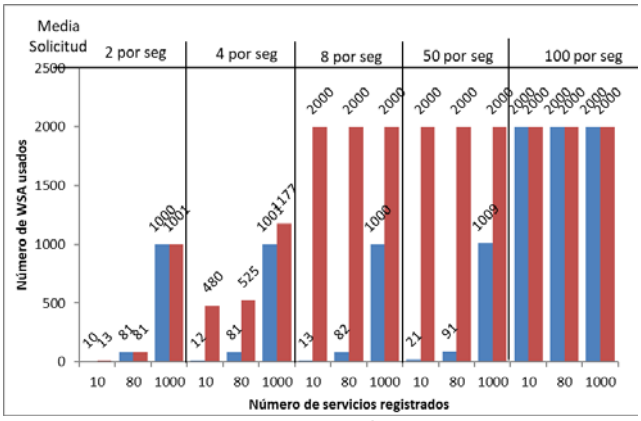


Figura 18. WSAs requeridos respecto del número de servicios en el UDDI (azul propuesta 1, rojo propuesta 2).
Source: Los autores.

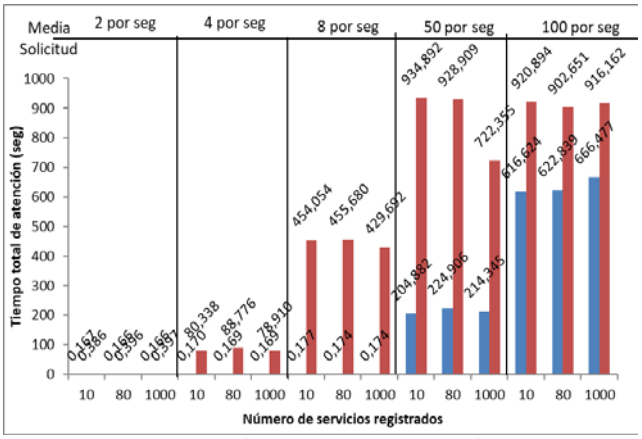


Figura 17. Tiempo de atención promedio respecto del número de servicios en el UDDI (azul propuesta 1, rojo propuesta 2).
Source: Los autores.

alcanzando tiempos superiores a los 500 segundos, aunque nunca tan malo como la propuesta 1. En general, cuando el número de solicitudes por segundo es inferior a 50, la propuesta 1 se comporta de manera adecuada, mientras que la propuesta 2 lo hace cuando el número de solicitudes por segundo es menor a 8, ya que se observa que el tiempo total de atención promedio es inferior a 2 segundos.

Al analizar más a fondo los resultados de la simulación, se observa nuevamente que el problema se presenta en el nodo de comunicación con la nube para la propuesta 1, y para la propuesta 2 se observa congestionamiento en los nodos ESB y de comunicación con la nube, lo cual hace que los tiempos de atención totales aumenten considerablemente cuando la red se satura.

Escenario 3: En este escenario se prueba la comunicación en ambos sentidos (bidireccional), con el fin de verificar si se afectan entre sí y se generan conflictos. Los valores que se toman para las variables son los mismos de los escenarios anteriores, solo que esta vez actúan en conjunto, indicando que habrá el doble de solicitudes, ya que se generan tanto solicitudes de servicios web, como solicitudes de tareas de agente como servicio.

Una vez realizadas las simulaciones se crearon tres gráficas, que se presentan en las Fig. 17-19.

En la Fig. 17 se muestra el comportamiento del sistema respecto del número de servicios registrados en el UDDI y el tiempo total de atención promedio. Se aprecia en esta figura, que cuando la media de solicitud de servicios es baja (2 solicitudes por segundos), ambas propuestas tienen un comportamiento aceptable, incluso similares, en cambio, cuando la media de solicitud es regular (8 solicitudes por segundo) o alta (50 y 100 solicitudes por segundo), la propuesta 2 comienza a presentar resultados muy poco aceptables, pues el tiempo de atención se incrementa considerablemente, mientras que la propuesta 1 comienza a presentar problemas para medias de solicitudes altas (50 y 100 solicitudes por segundo).

Analizando más a fondo los resultados, se consigue que nuevamente para la propuesta 2, el nodo ESB y el nodo de comunicación con la nube están saturados, mientras que para la propuesta 1 solo el nodo de internet se congestiona, lo cual produce que los tiempos de atención aumenten, debido a la cantidad de datos que deben pasar por la red. En este escenario esto aún más ocurre, debido a que están activos ambos sentidos de la comunicación, lo que significa que están pasando el doble de datos por la red que en los escenarios 1 (comunicación solo en sentido MAS-SOA) y 2 (comunicación solo en sentido SOA-MAS). Debido a lo anterior, los problemas en este escenario empiezan con medias de solicitud más bajas, con respecto a los dos escenarios anteriores. De igual manera, en este escenario, la propuesta 2 también presenta un mayor tiempo total promedio de atención con respecto a la propuesta 1, ya que los datos deben pasar primero por el ESB para que sean reenviados a los destinos correspondientes.

Análogamente, en la Fig. 18, donde se presenta el número de WSAs necesarios para cumplir con los requerimientos de consumo de servicios web por parte de los agentes.

De la Fig. 18 se tiene que para la propuesta 1, el número de WSAs requeridos es muy cercano al número de servicios registrados en el UDDI en todos los casos, excepto para una media de solicitud de 100 solicitudes por segundo, donde el sistema alcanza el límite máximo de 2000 WSA. Sin embargo, para la propuesta 2 el número máximo de WSA es

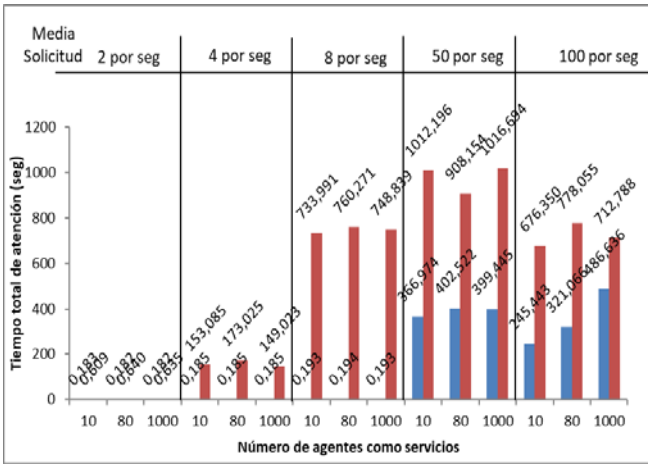


Figura 19. Tiempo de atención promedio respecto del número de agentes que ofrecen tareas como servicios (azul propuesta 1, rojo propuesta 2). Source: Los autores.

alcanzado para cuando la media es de 8 solicitudes por segundo. Esto se debe al congestionamiento que ocurre en los nodos ESB y comunicación con la nube, lo cual provoca que el tiempo total de atención promedio se eleve, ya que se generan muchas solicitudes concurrentes, debido a que los WSA no terminan de atender una solicitud cuando ya se genera otra gran cantidad, lo cual provoca que se vayan acumulando las solicitudes y el número de WSA se dispare hasta alcanzar su valor máximo.

Por otro lado, respecto a la comunicación SOA-MAS, se observa (ver Fig. 19) que en el caso de la primera propuesta los tiempos se mantienen bajos cuando la media de solicitud es menor a 8 solicitudes por segundo, con valores muy similares a los del segundo escenario. En cambio, en el caso de la propuesta 2, los tiempos de respuesta se mantienen bajos solo cuando la media es de 2 solicitudes por segundo, pero cuando la media cambia a 4 solicitudes por segundo o más, los tiempos de respuesta pasan a ser inaceptables.

Revisando con mayor detalle la simulación, se observa, que el tiempo total de atención también se ve afectado debido al retardo generado en los nodos ESB y comunicación con la nube, viéndose mayormente afectada la segunda propuesta, debido a la inclusión del ESB, donde todos los datos de la comunicación deben pasar a través del bus de servicios, para que sean ruteados a su destino final.

Según estos resultados, se puede deducir que la propuesta 1 tiene siempre mejor rendimiento que la propuesta 2, puesto que siempre los tiempos de atención promedios y el número de WSA máximo requerido siempre son menores que para el caso de la propuesta 2. A pesar que la propuesta 1 se ve afectada cuando el número de solicitudes es muy alto (100 solicitudes cada segundo), el problema se debe mayormente al canal de comunicación que se satura debido a la alta cantidad de datos que deben pasar a través de él, y no a la propuesta en sí misma. En cambio, para la propuesta 2, siempre los tiempos son superiores a los de la propuesta 1, debido a que el ESB genera un retardo para procesar y enrutar los datos, lo que afecta al tiempo de atención total.

En general, se observa que el rendimiento de la propuesta 1 es mejor el de la propuesta 2.

Table 1.
Comparación de las propuestas con trabajos previos.

# Car.	[20,31,32]	[33]	[34]	[36]	[35,37,43]	Prop. 1 y 2
1	✓	×	✓	✓	✓	✓
2	✓	✓	✓	✓	×	✓
3	×	✓	×	×	n/a	✓
4	×	×	✓	×	✓	✓
5	×	✓	×	×	✓	✓
6	✓	×	×	×	✓	✓

Source: Los autores.

5.4. Comparación con trabajos previos

Debido a que los autores de las propuestas anteriores, no presentaron resultados cuantitativos, la comparación de las propuestas se hará en forma cualitativa, en base a lista de 6 características presentadas al inicio de la sección 4. El resultado se resume en la Tabla 1, donde se observa que los trabajos previos a esta investigación no cumplen con todas las características deseadas para la integración MAS-SOA. Además, algunos son muy poco automatizados, y otros no saben cómo lidiar con la naturaleza sin estado/con estado de MAS y SOA.

La investigación que cumple con la mayoría de requerimientos es [35, 37, 43], pero la misma no es compatible con el estándar FIPA-ACL, limitando la integración con otros MAS que sí lo sean.

Por su parte, las propuestas 1 y 2 realizadas en esta investigación poseen todos los requerimientos planteados, lo cual se mostró en las anteriores subsecciones de esta sección. Sin embargo, en estás subsecciones no queda muy claro cómo resolver el problema para mapeo M a N. El mismo se resuelve de la siguiente forma:

1. Si un agente cualquiera (RA1) del MAS está consumiendo un servicio web caracterizado como WSA1, y algún otro agente (RA2) necesita consumir el mismo servicio web; en el momento que el agente de servicio sea solicitado al CCA, éste se dará cuenta que el WSA1 está ocupado atendiendo otro requerimiento, por lo cual debe solicitarle al SMA (en la primera propuesta) o al AMA (en la segunda) que cree un nuevo agente WSA2 que caracterice el mismo servicio; de esta forma, WSA2 podrá atender los requerimientos del RA2, posibilitando así la concurrencia al servicio.
2. Para controlar que no haya un crecimiento exponencial de agentes de servicio (WSA), el AGS o AMA (dependiendo de la propuesta), debe eliminar el registro de los WSA en el MAS, cuando se determine que tienen un tiempo específico sin atender solicitudes.
3. De manera similar se procedería con los agentes WSOA, cuando dos servicios intenten comunicarse con un mismo agente en un mismo momento.

6. Consideraciones finales

Se puede ver que ambas propuestas permiten la integración bidireccional MAS-SOA; igualmente, sus arquitecturas son FIPA-compliant. De igual forma, también son orientadas a servicios, ya que por medio de los WSOA se estará usando el paradigma SOA. Los agentes Gateway (WSA y WSOA) se crean

de manera automática, cuando servicios y agentes son registrados en los directorios respectivos. La transformación entre comunicaciones síncronas y asíncronas se hace en los agentes WSA y WSOA. El componente "Message Translator" se hará cargo de transformar los elementos semánticos que acompañan los mensajes, para que sean entendibles tanto por servicios web como por agentes.

En el mismo sentido, el mapeo M a N se realiza al crear varios agentes WSA y WSOA para un mismo servicio web o agente, eso se realizaría cuando el agente CCA determine que el WSA (o el WSOA) que caracteriza a un servicio (o agente) no se encuentra disponible. Finalmente, la arquitectura propuesta fue especificada de forma independiente a la implementación ya que no está ligada a ninguna librería particular, y a la hora de implementarla será cuando se decida que librerías deben utilizarse.

Por su parte, cada propuesta permite a los agentes de AmICL consumir servicios en la nube, así como exponer sus funcionalidades como servicios web (integración bidireccional de AmICL con la nube). Por otro lado, debido a que los agentes necesarios para integrar MAS y SOA son los agentes AMA, CCA, WSA, SMA y WSOA para la primera propuesta, y AMA, CCA, WSA, SDA y WSOA para la segunda propuesta, y debido a que AMA y CCA son abstracciones del estándar FIPA (en JADE por ejemplo, AMA es el DF y CCA es el CCA proveído por el framework), estás propuestas pueden ser extendidas a cualquier plataforma de despliegue de sistemas multi-agente que requiera servirse de la integración SMA y SOA. A dichas plataformas, solo se les deberá incorporar (implementar) los respectivos agentes establecidos en cada propuesta no considerados por FIPA (por ejemplo, en la propuesta 1 a WSA, SMA y WSOA).

De acuerdo a los experimentos realizados, se puede concluir que la primera propuesta brinda un mejor rendimiento, en cuanto a tiempos de respuesta y consumo de recursos del sistema, la cual podría utilizarse en AmIs donde el número de solicitudes no sea tan elevado y la latencia de internet sea muy baja.

El trabajo futuro está orientado a mejorar los tiempos de respuesta ante números de solicitudes elevados, para lidiar con problemas de tiempo real y latencia de internet altas, mediante la incorporación de un componente basado en Fog Computing, que servirá para poder usar este componente en sistemas a gran escala, como es el caso de las ciudades inteligentes.

Referencias

[1] Mhiri, F. and Ratté, S., AARTIC: development of an intelligent environment for human learning. SIGCSE Bulletin, 41(3), pp. 359-359, 2009. DOI: 10.1145/1562877.1563001

[2] Mikulecký, P., Smart environments for smart learning, Proceedings of the 9th International Scientific Conference on Distance Learning in Applied Informatics DIVAI, Sturovo, Slovakia, 2012.

[3] Shi, Y., Qin, W., Suo, Y. and Xiao, X., Smart classroom: bringing pervasive computing into distance learning [Online]. pp. 881-910, 2010. Available at:https://goo.gl/7VK6Bc

[4] Stenvall-Virtanen, S. and Nordell, K., Smart environments: technology, protocols and applications. Tech. Rep.: University of Turku, Turku, Finland, 2014.

[5] Duraes, D., Castro, D., Bajo, J. and Novais, P., Modelling an intelligent interaction system for increasing the level of attention, Proceedings of the International Symposium on Ambient Intelligence [Online]. pp. 210-217, 2017. Available at: https://goo.gl/gGVzkq

[6] Bai, Y., Shen, S., Chen, L. and Zhuo, Y., Cloud learning: A new learning style, Proceedings of the International Conference on Multimedia Technology, 2011. DOI: 10.1109/ICMT.2011.6002268

[7] Castillo-Rodríguez, C. and Rios-Moyano, S., Promoción del c-learning a través del uso de herramientas, software y entornos virtuales. Historia y Comunicación Social [En línea]. 18(1), pp. 305-317, 2013. Disponible en: https://goo.gl/3TWifL

[8] IBM, Introducción a SOA y servicios web [En línea]. 2007. [fecha de referencia: Febrero 27 de 2018]. Disponible en: https://goo.gl/aQSnjZl.

[9] Serrano, N., Hernantes, J. and Gallardo, G., Service-oriented architecture and legacy systems. IEEE software, 31(5), pp. 15-19, 2014. DOI: 10.1109/MS.2014.125

[10] Sanchez, M., Aguilar, J., Cordero, J. and Valdiviezo, P., A smart learning environment based on cloud learning. International Journal of Advanced Information Science and Technology (IJAIST), 39(39), pp. 39-52, 2015.

[11] Sánchez, M., Aguilar, J., Cordero, J. and Valdiviezo, P., Basic features of a reflective middleware for intelligent learning environment in the cloud (IECL), Proceedings of the Asia-Pacific Conference on Computer Aided System Engineering, 2015. DOI: 10.1109/APCASE.2015.8

[12] Valdiviezo, P., Cordero, J., Aguilar, J. and Sánchez, M., Conceptual design of a smart classroom base on multiagent system, Proceedings of the International Conference on Artifitial Intelligence (ICAI'15), 2015.

[13] Aguilar, J., Sánchez, M., Cordero, J., Valdiviezo-Díaz, P., Barba-Guamán, L. and Chamba-Eras, L., Learning analytics tasks as a services in smart classrooms. International Journal of Universal Access in the Information Society [Online]. pp. 1-17, 2017. Available at: https://goo.gl/YV4xWN.

[14] Aguilar, J., Sánchez, M., Valdiviezo, P. and Cordero, J., Mecanismos de coordinación en un salón inteligente, Proceedings of the 6to Congreso Iberoamericano de Estudiante de Ingeniería Eléctrica, 2015.

[15] Sánchez, M., Aguilar, J., Cordero, J., Valdiviezo, P., Barba-Guamán, L. and Chamba-Eras, L., Cloud computing in smart educational environments: application in learning analytics as service. New advances in information systems and technologies, 444(1), Springer International Publishing, pp. 993-1002, 2016. DOI: 10.1007/978-3-319-31232-3_94

[16] FIPA. IEEE [Online]. 2013. [date of reference: Febrary 27th of 2018]. Available at: http://www.fipa.org.

[17] Deen, S.M., Agent-based manufacturing: advances in the holonic approach. Springer, 2013.

[18] Aguilar, J., Ríos, A., Hidrobo, F. and Cerrada, M., Sistemas multiagentes y sus aplicaciones en automatización industrial, 2nd ed., Mérida: Talleres Gráficos, Universidad de Los Andes, 2013.

[19] Lopes, F. and Coelho, H., Negotiation and argumentation in multi-agent systems: Fundamentals, theories, systems and applications. Bentham Science Publishers, 2014.

[20] Greenwood, D., Buhler, P. and Reitbauer, A., Web service discovery and composition using the web service integration gateway, Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005. DOI: 10.1145/1329125.1329458

[21] Sánchez, A., Villarrubia, G., Zato, C. and Chamoso, P., A gateway protocol based on FIPA-ACL for the new agent platform PANGEA. Trends in Practical Applications of Agents and Multiagent Systems [Online]. Springer, pp. 41-51, 2013. Available at: https://goo.gl/yHNNKJ

[22] Marin, C., Monch, L., Leitao, P., Vrba, P., Kazanskaia, D., Chepegin, V., Liu, L. and Mehandjiev, N., A conceptual architecture based on intelligent services for manufacturing support systems, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 4749-4754, 2013. DOI: 10.1109/SMC.2013.808

[23] Singapogu, S.S., Gupton, K. and Schade, U., The role of ontology in C2SIM, Proceedings of the 21st ICCRTS, London, 2016.

[24] Geetha, R. and Shunmuganathan, K., Intelligent query processing from biotechnological database using co-operating agents based on FIPA standards and Hadoop, in a secure cloud environment, Proceedings of the 4th International Conference on Advanced Computing and Communication Systems [Online]. pp. 1-4, 2017. Available at: https://goo.gl/HwSVBz

[25] Oracle, Simple object access protocol overview [Online], Oracle Corporation, 2001 [date of reference: February 27th of 2018]. Available at: https://goo.gl/MR1ZJv.

[26] De-la-Cruz, A., Una aproximación MDA para la conversión entre servicios web SOAP y RESTful, Universidad complutense de Madrid,

Madrid, 2013.
[27] López, M., Carrillo, R. and Martínez, P., Subsistema Informático para la Interoperabilidad de la Plataforma SIUDERLAN desarrollada en la empresa ETECSA. Universidad y Sociedad, 8(4), pp. 100-105, 2016.
[28] Tang, J., Liu, S.L., Gu, Z. and Gaudiot, J.L., Acceleration of xml parsing through prefetching. IEEE Transactions on computers, 62(8), pp. 1616-1628, 2013. DOI: 10.1109/TC.2012.88
[29] W3C, Extensible Markup Language (XML) [Online], W3C, 2016 [date of reference: February 27th of 2018]. Available at: https://goo.gl/9FLm2E
[30] Fielding, R. and Reschke, J., Hypertext transfer protocol (HTTP/1.1): Message syntax and routing [Online], 2014 [date of reference: February 27th of 2018]. Available at: https://goo.gl/qdae7f
[31] Greenwood, D. and Calisti, M., Engineering web service-agent integration, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2004. DOI: 10.1109/ICSMC.2004.1399962
[32] Shafiq, M.O., Ding, Y. and Fensel, D., Bridging multi agent systems and web services: towards interoperability between software agents and semantic web services, Proceedings of the IEEE International Conference on Enterprise Distributed Object Computing Conference [Online]. 2006. Available at: https://goo.gl/JeSFGm
[33] Nguyen, X.T. and Kowalczyk, R., WS2JADE: Integrating web service with jade agents, Proceedings of Service-Oriented Computing: Agents, Semantics, and Engineering. Springer, Berlin, Heidelberg, pp. 147-159, 2007. DOI: 10.1007/978-3-540-72619-7_11
[34] Mordacci, P., Poggi, A., Tiso, C.G. and Turci, P., Using agent technology as a support for an enterprise service bus, Proceedings of the Workshop on Objects and Agents (WOA 2008), 2008.
[35] Leitão, P., Towards self-organized service-oriented multi-agent systems. In: Service orientation in holonic and multi agent manufacturing and robotics. Springer, pp. 41-56, 2013. DOI: 10.1007/978-3-642-35852-4_3
[36] Fuksa, M., Methods and tools for intelligent ESB, Czech Technical University in Prague, Prague, 2014.
[37] Pinto-Pereira, A., Towards robustness and self-organization of ESB-based solutions using service life-cycle management [Online], Polytechnic Institute of Bragança, 2014. Available at: https://goo.gl/RLDfDi
[38] Vizcarrondo, J., Aguilar, J., Exposito, E. and Subias, A., ARMISCOM: Self-healing service composition. Service Oriented Computing and Applications, 11(3), pp. 345-365, 2017.
[39] Vizcarrondo, J., Aguilar, J., Exposito, E. and Subias, A., ARMISCOM: Autonomic Reflective MIddleware for management Service COMposition, Proceedings of the Global Information Infrastructure and Networking Symposium. pp. 1-8, 2012. DOI: 10.1109/GIIS.2012.6466760
[40] Lalanda, P., McCann, J. and Diaconescu, A., Autonomic Computing, Springer, 2013.
[41] Su, Z., Song, C., Dai, L., Ge, F., Yang, R. and Biennier, F., A security criteria regulation middleware using security policy for Web Services on multi-Cloud tenancies, Proceedings of the 3rd International Conference on Computational Intelligence & Communication Technology (CICT), 2017. DOI: 10.1109/CIACT.2017.7977320
[42] Paik, H., Lemos, A., Barukh, M.C., Benatallah, B. and Natarajan, A., Web Services-SOAP and WSDL. In: Web Service Implementation and Composition Techniques. Springer, pp. 25-66, 2017, DOI: 10.1007/978-3-319-55542-3_2
[43] Colombo, A., Karnouskos, S., Mendes, J. and Leitão, P., Industrial agents in the era of service oriented architectures and cloud based industrial infrastructures. In: Industrial Agents: Emerging Applications of Software Agents in Industry, Elsevier, 2015. DOI: 10.1016/B978-0-12-800341-1.00004-8
[44] Greenwood, D., Lyell, M., Mallya, A. and Suguri, H., The IEEE FIPA approach to integrating software agents and web services, Proceedings of the 6th international joint conference on autonomous agents and multi-agent systems, 2007. DOI: 10.1145/1329125.1329458
[45] Kumar, A., Tayal, A., Kumar, S. and Bindhumadhava, B.S., Multi-agent autonomic architecture based agent-web services, Proceedings of the 16th International Conference on Advanced Computing and Communications (ADCOM), 2008. DOI: 10.1109/ADCOM.2008.4760469
[46] Buitrago, S. and Sánchez, M., VMAS-Modeller: Una aplicación visual para el modelado de sistemas multi-agentes guiado por la metodología MASINA. ReVeCom, 4(1), pp. 47-58, 2017.
[47] JADE, JAVA Agent DEvelopment Framework [Online]. 2017 [date of reference: February 27th of 2018]. Available at: http://jade.tilab.com/
[48] Kress, J., Maier, B., Normann, H., Schmeidel, D., Schmutz, G., Trops, B., Utschig-Utschig, C. and Winterberg, T., Enterprise Service Bus [Online]. Oracle Corporation, 2013 [date of reference: February 27th of 2018]. Available at: https://goo.gl/sQBzBk
[49] CXF, Apache CXF [Online]. Apache, 2007 [date of reference: February 27th of 2018]. Available at: https://cxf.apache.org/
[50] Richardson, L. and Ruby, S., RESTful web services, Beijing: O'Reilly, 2008.
[51] Postel, J. and Reynolds, J., File transfer protocol [Online]. 1985 [date of reference: February 27th of 2018]. Available at: https://goo.gl/KaiEaz
[52] Hoffman, P., SMTP service extension for secure SMTP over transport layer security [Online]. 2002 [date of reference: February 27th of 2018]. Available at: https://goo.gl/yyDGWu
[53] Jerez, M., Aguilar, J., Exposito E. and Tierry, V., CARMICLOC: Context Awareness Middleware in Cloud Computing, Proceedings of the XLI Conferencia Latinoamericana en Informática, pp. 532-541, 2015. DOI: 10.1109/CLEI.2015.7360013

M. Sánchez, recibió el título de Ingeniero de Sistemas en 2002 por la Universidad de los Andes, Venezuela. El grado de MSc. en Ciencias de la Computación por la Universidad de los Andes, Venezuela, en 2012. Actualmente, está desarrollando una tesis doctoral para obtener el título de Doctor en Ciencias Aplicadas (Ambientes Inteligentes) en la Universidad de los Andes, Venezuela. Es profesor asociado del Departamento de Ingeniería en Informática en la Universidad Nacional Experimental del Táchira, Venezuela. Sus intereses en investigación incluyen Inteligencia Artificial, Ambientes de Aprendizaje, Computación Gráfica y lenguajes de programación.
ORCID: 0000-0002-3310-3394.

J. Aguilar, recibió el título de Ingeniero de Sistemas de la Universidad de los Andes, Venezuela en 1987, el grado de MSc. en Ciencias de la Computación en 1991 de la Universite Paul Sabatier, France, y el grado de PhD en Ciencias de la Computación en 1995 de la Universite Rene Descartes, France. Realizó estudios post-doctorales en el Departamento de Ciencias de la Computación en la Universidad de Houston (1999-2000). Es profesor titular del Departamento de Ciencias de la Computación en la Universidad de los Andes, Mérida, Venezuela, y fue investigador Prometeo en la Universidad Técnica Particular de Loja, y en la Escuela Politécnica Nacional, Ecuador. Es miembro de la Academia de Ciencias de Mérida y de la IEEE CIS Technical Committee on Neural Networks. Ha publicado más de 500 artículos y 10 libros, en el campo de computación paralela y distribuida, inteligencia computacional, ciencias y gestión tecnológica, etc. Sus intereses de investigación incluyen Inteligencia Artificial, Minería semántica, Big Data, Computación Emergente y Ambientes Inteligentes.
ORCID: 0000-0003-4194-6882.

E. Exposito, es profesor titular en la Université de Pau et des Pays de l'Adour en la UFR ST Côte Basque en Anglet, Francia. En 1994, obtuvo el grado de Ing. en Ciencias de la Computación de la Universidad Centro-Occidental Lisandro Alvarado, Venezuela. En 2003, obtiene su grado de Dr. en Informatique et Télécommunications, del Institut National Polytechnique de Toulouse, Francia. En 2004, trabajó como investigador en el centro de investigación de NICTA (National ICT Australia Limited) en Sydney, Australia. Entre 2006 y 2016 trabajó como profesor asociado en INSA Toulouse e investigador en LAAS/CNRS en Toulouse, Francia. En 2010, obtuvo la Habilitation à diriger des recherches del Institut Nacional Polytechnique de Toulouse. Ha sido presidente y miembro de muchos Comités. Ha participado y coordinado diversas acciones en varios proyectos de investigación europeos y franceses, incluyendo el proyecto europeo IMAGINE relacionado con las Redes de Fabricación Dinámica. Entre 2014 y 2016, trabajó en actividades de coordinación de innovación pedagógica y fue responsable de asuntos internacionales entre el INSA de Toulouse y América Latina.
ORCID: 0000-0002-3310-3394