Review of Architectural Patterns and Tactics for Microservices in Academic and Industrial Literature

G. Márquez, F. Osses and H. Astudillo

***Abstract*—Microservices are an emerging trend for development of service-oriented software. This approach proposes to build each application as a collection of small services running on separate process and inter-communicating with lightweight mechanisms. Systematic development of microservices is hampered by the lack of a catalog of emerging recurrent architectural solutions (*architectural patterns*) and design decisions (*architectural tactics*). This article describes a systematic review of academic and industrial literature regarding architectural patterns and architectural tactics for microservices. The review yield 44 architectural patterns in academic sources and 74 in industrial ones, as well as a few architectural tactics originally proposed to address related problems. Most architectural patterns and tactics are associated to one of just five quality attributes: scalability, flexibility, testability, performance, and elasticity. Also, most microservices in academic (but not industrial) literature are related to DevOps and IoT. The findings lead to propose a new taxonomy of microservice architectural patterns.**

Index Terms—**Architectural patterns, architectural tactics, mi- croservices, taxonomy, sistematic literature review, academy, industry**

I. INTRODUCCIÓN

UN estilo arquitectónico basado en microservicios es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su proceso y comunicándose con mecanismos livianos. Estos servicios se basan en capacidades de negocio y se pueden implementar de forma independiente [1] [2]. Sin embargo, a pesar del gran interés en microservicios [3] [4], no es posible observar propuestas que describan problemas recurrentes (*patrones de arquitectura*) o decisiones de diseño (*tácticas de arquitectura*) que están surgiendo en esta disciplina. Los *patrones de arquitectura* describen una estructura de un sistema a alto nivel con su comportamiento asociado [5]. Muchos de los beneficios y responsabilidades de estos patrones se relacionan con atributos de calidad (QA) [6].

G. Márquez, Toeska Research Group, Departamento de Informática Universidad Técnica Federico Santa María Valparaíso, Chile, gas-ton.marquez@sansano.usm.cl

F. Osses, Toeska Research Group, Departamento de Informática, Universidad Técnica Federico Santa María Valparaíso, Chile, felipe.osses@sansano.usm.cl

H. Astudillo, Toeska Research Group, Departamento de Informática, Universidad Técnica Federico Santa María Valparaíso, Chile, hernan@inf.utfsm.cl

Los QAs permiten establecer la base de los patrones de arquitectura a través de la construcción de "bloques arquitectónicos" llamados *tácticas de arquitectura* [6] las cuales son decisiones de diseño que influyen en el logro de una respuesta de un atributo de calidad (QA).

Debido a esta falta de evidencia respecto a patrones de arquitectura y tácticas, el objetivo de nuestro estudio es explorar qué patrones y tácticas se han descrito para microservicios en la academia, así como en la industria. Para lograr este objetivo, hemos realizado una revisión sistemática de la literatura con el afán de estudiar la evidencia tanto en la academia como la industria. Nuestro artículo se compone de la siguiente manera: en la Sección II hablaremos de la motivación y trabajos relacionados. Posteriormente en la Sección III describiremos la planificación de la revisión. En la Sección IV ilustraremos los resultados y análisis obtenidos y, posteriormente, describiremos en la Sección V la evaluación de calidad que aplicaremos a la evidencia encontrada para dar pie a la Sección VI, en donde describiremos la propuesta de nuestra taxonomía. Finalizaremos detallando las amenazas a las valideces (Sección VII) y las conclusiones correspondientes (Sección VIII).

II. MOTIVACIÓN Y TRABAJOS RELACIONADOS

Según [1], los microservicios se basan en capacidades empresariales y se pueden implementar de forma independiente mediante un procedimiento totalmente automatizado [2]. Una de las empresas pioneras en la adopción de microservicios es Netflix. Hace algún tiempo, Netflix comenzó la migración de una arquitectura monolítica hacia microservicios, documentan- do abiertamente todo lo aprendido. Hoy en día, la aplicación Netflix está impulsada por una arquitectura que cuenta con APIs que maneja cerca de dos mil millones de solicitudes cada día y que son administrados por aproximadamente 500 microservicios. Por otro lado, Uber tiene aproximadamente 1300 microservicios para mejorar la confiabilidad y escalabilidad [7] de la aplicación, desarrollando estándares globales que se aplican a todos los microservicios.

La literatura describe que Di Francesco et al. [3] contribuye con un marco de trabajo para clasificar, comparar y evaluar soluciones arquitectónicas, métodos y técnicas para microservicios. Vural et al. [4], Alshuqayran et al. [8] y Pahl et al. [9] tienen como objetivo identificar, clasificar y comparar la investigación existente en microservicios y su aplicación en la nube. Dragoni et al. [10] presenta el estado

actual de la tecnología en este campo, revisando la historia de la arquitectura del software. Finalmente, Osses et al. [11] describe la evidencia preliminar de una revisión sistemática con respecto a los patrones de arquitectura en microservicios limitados a la academia.

Aunque hay evidencia de que exploración de patrones de arquitectura y tácticas limitadas a la academia, no hay suficiente evidencia en la industria. Por lo tanto, esto nos da la motivación para realizar esta revisión sistemática.

III. PLANIFICACIÓN DE LA REVISIÓN

En esta sección, describiremos el método de revisión ejecutado en este estudio propuesto por B. Kitchenham [12] [13]. Centramos el estudio en tres fases: la Fase 1 tiene como objetivo extraer la evidencia académica sobre patrones y tácticas de arquitectura. La Fase 2 complementa la evidencia académica con propuestas obtenidas de otras fuentes de información. Pero, como los microservicios son tendencias industriales, fue necesario agregar una Fase 3 para completar nuestro estudio con la evidencia industrial y comparar los resultados.

Protocolo de revisión: Para este estudio, establecimos un protocolo [13], que especifica el método utilizado para la revisión sistemática. El método comienza con tres preguntas de investigación; el proceso de búsqueda; los criterios de selección; el protocolo de extracción de datos y revisión de ejecución. Para reducir la posibilidad de sesgo, el estudio fue desarrollado por tres investigadores. La ejecución del protocolo se realizó entre marzo y noviembre del 2017.

Preguntas de investigación: Para establecer las preguntas de investigación, organizamos reuniones con expertos utilizando la técnica de lluvia de ideas centrada en conceptos claves relacionados a la investigación. En esas reuniones, establecimos las siguientes preguntas de investigación que se utilizaron en las Fases 1, 2 y 3: PI1: *¿Qué patrones de arquitectura se han propuesto para microservicios?*, PI2: *¿Qué tácticas de arquitectura se han propuesto para microservicios?* y PI3:

¿Qué atributos de calidad se relacionan con microservicios?

Fase 1 y Fase 2 de protocolo de revisión académico: En la comunidad académica, utilizamos un grupo de términos para encontrar estudios relevantes. Combinamos todos los términos usando los operadores booleanos AND/OR. En la cadena de búsqueda hay bases de dos palabras: *arquitectura de software* y *microservicios*, que deberían incluirse en todas las preguntas. Por lo tanto, se decidió no solo usar palabras *tácticas* y *patrones*, sino que también decidimos expandir la cadena de búsqueda usando las palabras *decisión* y *diseño*.

En la primera fase obtuvimos 744 evidencias en las bases de datos más importantes de revistas y conferencias en la comunidad de ingeniería de software. Las bases de datos consultadas en la primera fase fueron: biblioteca digital de la ACM, IEEE Xplorer, ScienceDirect-Elsevier y Wiley InterScience. En la segunda fase, con el objetivo de ampliar la búsqueda debido a la novedad del microservicio, obtuvimos 323 documentos. Las bases de datos consultadas en la segunda fase fueron: Google Scholar y OATD (Base de datos de tesis abiertas).

Fase 3 de protocolo de revisión industrial: En esta fase se reprodujo los mismos pasos que en la Fases 1 y 2. Se usaron buscadores web tales como Google y Bing. Las fuentes que fueron revisadas son *blogs*, literatura gris, libros y editoriales.

Criterios de inclusión y exclusión para las tres fases: Cada evidencia se analizó utilizando criterios de inclusión y exclusión, para verificar si la información encontrada es adecuada para responder las preguntas de investigación. En general, los criterios de inclusión consideran que la evidencia debe estar relacionada a la ingeniería de software, que la evidencia debe estar publicada en eventos y conferencias del área y otros. Por otro lado, los criterios de exclusión consideraron aquellos trabajos que no se relacionan con la ingeniería de software, si la evidencia está repetida en varias fuentes, si la evidencia no describe bien su contribución, entre otros.

IV. RESULTADOS Y ANÁLISIS

En esta sección se detallarán los resultados obtenidos. Las Tablas I y II resumen los patrones de arquitectura encontrados. A continuación, procederemos a responder las preguntas descrita en la sección anterior.

PI1: ¿Qué patrones de arquitectura se han propuesto para microservicios? La Tabla I resume los 44 patrones de arquitectura de microservicios encontrados en la comunidad académica y la Tabla II ilustra los 74 patrones de arquitectura en la industria. En el aspecto académico, la revisión sistemática de Osses et al. [11] muestra el primer indicio de patrones de arquitectura propuestos por la academia. Hemos decidido resumir nuestros hallazgos utilizando como referencia el esquema propuesto en [27], que es: referencia, ID (AMSP e IMSP, ***A****cademic/****I****ndustrial* ***M****icro* ***S****ervice* ***P****attern*), nombre, contexto, problema y solución. Al momento de analizar los patrones de arquitectura tanto académicos como industriales, nos percatamos que existen tópicos que convergen en estos dos mundos.

Tanto la industria como la academia entienden que este tema describe que una aplicación puede ser inicialmente dirigida a "contendedores". No teníamos dudas de que este concepto iba a estar presente en los patrones de arquitectura. Un contenedor encapsula componentes discretos de la lógica de la aplicación provisionada solo con los recursos mínimos necesarios para hacer su trabajo. A diferencia de las máquinas virtuales (VM), los contenedores no necesitan un sistema operativo integrado; las llamadas que se realizan para los recursos del sistema operativo se hacen a través de una API. Lo anterior, nos permite debatir ¿Por qué los *quiebres de circuito* han surgido en los microservicios y, tanto la academia y como la industria, lo visualizaron? Porque la idea es simple y resuelve problemas recurrentes en microservicios. Finalmente, los patrones de arquitectura coinciden en que los datos persistentes de cada

microservicio deben ser privados y solo se puede acceder a ellos a través de su API.

PI2: ¿Qué tácticas de arquitectura se han propuesto para microservicios? Las tácticas de arquitectura que se encuentran en esta revisión no están directamente relacionadas con los microservicios. Observamos que las tácticas de arquitectura para microservicios como tales no existen en la academia. Sin embargo, hay indicios de tácticas de arquitectura que se complementan entre sí con otras disciplinas, como DevOps (Desarrollo y Operaciones) [28] [29]. Artículos como el de [30] describen dos indicaciones de lo que pueden ser tácticas de arquitectura: (1) *scaling* y (2) *independence*. Por un lado, (1) detalla que un microservicio debe implementarse independientemente con cada microservicio que tenga su arquitectura de despliegue; y no compartir sus recursos en contenedores y almacenes de datos con otros componentes de software empresarial.

Por otro lado, (2) se complementa con (1) y detalla que la escala de un microservicio es independiente de otros microservicios.

TABLA I
PATRONES DE ARQUITECTURA ENCONTRADOS EN LA ACADEMIA

Ref.	ID	Nombre	Contexto	Problema	Solución
[14]	AMSP1	***Arquitectura web moderna***	Proporcionar una interfaz de usuario completa	¿Capacidades de diferentes dispositivos de interfaz?	Combinar componentes de interfaz con back-end
	AMSP2	***Página simple***	Nueva aplicación web o refactorizar	¿Tomar ventaja de los navegadores modernos?	Aplicación de diseño como una sola página
	AMSP3	***Aplicación móvil nativa***	Soporte en una variedad de plataformas	¿Experiencia de usuario más optimizada?	Aplicación móvil nativa para cada plataforma importante
	AMSP4	***Caché cercano***	Operar de manera eficiente en latencia	¿Reducir el número de llamadas?	Usa un caché ubicado en la implementación del cliente
	AMSP5	***Arquitectura de microservicios***	Aplicación con módulos sean independientes	¿Estructurar conjunto de módulos independientes?	Diseñe con módulos independientes del negocio
	AMSP6	***Microservicios de Negocio***	Arquitectura de microservicios o refactorizar	¿Administrar lógica de negocio?	Desarrollar cada función de negocio en microservicio
	AMSP7	***Backend para Frontend***	No están claras las necesidades específicas del cliente	¿Representar una interfaz de servicio específica?	Crear un *backend* para la interfaz con una API
	AMSP8	***Microservicio adaptador***	API no usan el enfoque RESTful	¿Manejar la traducción en API de microservicios?	Convertir la API que el cliente microservicios espera
	AMSP9	***Cache resultante***	Llamadas a bases de datos o servicios remotos es costoso	¿Mejorar el rendimiento?	Realizar atajos en las llamadas repetidas al mismo servicio
	AMSP10	***Página Cache***	*Backend* para interfaces	¿Información mostrada de manera fácil?	Usa un caché de página con backend para frontend
	AMSP11	***Almacenamiento escalable***	Estado persistente para representar la interacción del usuario	¿Representar el estado persistente?	Usar estados escalables
	AMSP12	***Almacenamiento de llave***	Compilación con una arquitectura de microservicios	¿Acceso a datos?	Almacenar los datos en un almacén escalable
	AMSP13	***Almacenamiento de documento***	Contenidos de HTTP como documentos de JSON	¿Acceso a datos HTTP?	Almacenar los documentos JSON en un almacén de datos
	AMSP14	***DevOps microservicios***	Aplicación compleja con una arquitectura de microservicios	¿Equilibrar necesidades entre desarrollo y operación?	Aislar cada microservicio tanto como sea posible
	AMSP15	***Agregador de registros***	Problemas de depuración en componentes	¿Visualizar y buscar todos los diferentes log?	Extraer archivos en una única base de datos
	AMSP16	***Correlación de ID***	Aplicación usa gráficos complejos	¿Depurar un gráfico complejo?	Agregar identificador en la solicitud de servicio
	AMSP17	***Registro de servicio***	Muchos microservicios diferentes en una aplicación	¿Desacoplar?	Agregar identificador único
[15]	AMSP18	***Habilitar integración conti.***	Migrar el sistema a microservicios	¿Disponer de artefactos para la producción?	Configurar la integración continua
	AMSP19	***Recuperar arquitectura actual***	Se necesita conocer la arquitectura del sistema actual	¿Imagen actual del sistema?	Mantener los artefactos lo más simple posible
	AMSP20	***Descomponer monolítico***	Sistema de software monolítico	¿Descomponer el sistema en elementos más pequeños?	Enfoque de diseño controlado por el dominio (DDD)
	AMSP21	***Descomponer datos***	Sistema monolítico sin un dominio complejo	¿Qué tan grandes deben ser los fragmentos?	Descomponer el sistema según la propiedad de los datos
	AMSP22	***Cambiar dependencia***	Componente del sistema que actúa como una dependencia	¿Cambiar la dependencia del nivel de código?	Mantener los servicios código tan separado como sea posible
	AMSP23	***Intro. servicio***	Instancias pueden cambiarse dinámicamente	¿Ubicar dinámicamente?	Almacenar direcciones de instancias de servicio
	AMSP24	***Intro. servicio cliente***	Número de instancias puede cambiarse dinámicamente	¿Se ha implementado una nueva instancia?	Conocer la dirección del servicio
	AMSP25	***Intro. balanceador interno***	Cada servicio puede ser un cliente del resto	¿Equilibrar la carga en un servicio?	Tener un equilibrador de carga interno
	AMSP26	***Intro. balanceador externo***	Cada servicio puede ser un cliente del resto	¿Equilibrar la carga de un servicio entre sus instancias?	Recupera la lista de instancias disponibles
	AMSP27	***Intro. quiebre de circuito***	Las solicitudes de los usuarios necesitan comunicación	¿Servicio no disponible?	El componente disponible no hace nada
	AMSP28	***Intro. config. servidor***	Instancias disponibles están a través de un servicio	¿Modificar la configuración de las instancias en ejecución?	Dos repositorios separados
	AMSP29	***Intro. servidor de borde***	Introducir nuevos servicios fácilmente	¿Ocultar la complejidad del servicio?	Otra capa de dirección en el sistema
	AMSP30	***Servicios en contenedores***	La canalización de integración continua está en su lugar	¿Producir los mismos resultados para el mismo código?	Servicio en una máquina virtual en aislamiento
	AMSP31	***Desarrollar clúster***	Una imagen de contenedor lista para producción	¿Implementar las instancias de un servicio en un clúster?	Administrar un clúster de nodos
	AMSP32	***Monitorear sistema***	El sistema se ejecutó en un clúster de contenedores	¿Monitorear la infraestructura subyacente?	Instalación de supervisión independiente
[16]	AMSP33	***Auto contenedor de servicios***	Autocontención es uno de los aspectos centrales	¿Impulsar software hacia mantenimiento y escalabilidad?	Backend como parte del servicio
	AMSP34	***Quiebre de circuito***	Cada servicio proporciona una interfaz para monitoreo	¿Monitorea y previene la cascada de fallas?	Comprobar el estado de la salud
	AMSP35	***Balanceador***	Cada servicio proporciona una interfaz para monitoreo	¿Supervisión y prevención de cascada de fallas?	Carga de trabajo distribuida en servicios iguales
	AMSP36	***Contenedor***	Encierro del microservicio, incluidas las bibliotecas y datos	¿Simplificar el despliegue de aplicaciones?	Carga de trabajo en equilibrador de carga
	AMSP37	***Manejar versiones de servicios***	Al probar una aplicación, el artefacto no se debe alterar	¿Altera los artefactos específicos?	Incluir nuevas versiones de cada servicio
	AMSP38	***Desarrollo de tipo blue green***	Reemplazar aplicaciones por nuevas versiones	¿No alterar artefactos específicos?	Las versiones tienen que ejecutarse en paralelo
	AMSP39	***Liberación canaria***	Desafíos con la automatización de la implementación	¿No alterar artefactos específicos?	Enrutar los nuevos servicios iterativamente
[17]	AMSP40	***Servicio de base de datos***	Comportamientos a nivel de la base de datos	¿Mejorar en términos de velocidad y escalabilidad?	Adición de nuevos comportamientos a nivel de base de datos
[18]	AMSP41	***Provisión IoT***	Dispositivos distribuidos geográficamente	¿Dispositivos como entorno de línea de base confiable?	Virtualización basada en contenedores
	AMSP42	***Desarrollo IoT***	Mantener dispositivos de IoT remotos	¿Implementar código en muchos dispositivos IoT?	Utilizar sistemas de control de versiones
	AMSP43	***Orquestación IoT***	Habilitar una gran cantidad de dispositivos conectados	¿Ordenar dispositivos de IoT?	Mecanismos de descubrimiento de servicios
	AMSP44	***Diámetro IoT***	Los mecanismos variables en modelos de negocio	¿Monitorear servicios IoT?	Protocolo liviano para telemetría de aplicaciones IoT

TABLA II PATRONES DE ARQUITECTURA ENCONTRADOS EN LA INDUSTRIA

Ref.	ID	Nombre	Contexto	Problema	Solución
[19]	IMSP1	***Descomponer capacidad del negocio***	*No especificado*	¿Descomponer en servicios?	Definir capacidades del negocio
	IMSP2	***Descomponer subdominio***	*No especificado*		Definir servicios por diseño dirigido por dominio
	IMSP3	***Múltiples instancias de servicio***	Sistema como conjunto de servicios	¿Desplegar servicios?	Implementar instancias de servicio en un solo host
	IMSP4	***Instancia de servicio por host***			Implementar cada servicio en su *host*
	IMSP5	***Instancia de servicio por máquina virtual***			Implementar cada servicio en su máquina virtual
	IMSP6	***Instancia de servicio por contenedor***			Implementar cada servicio en su contenedor
	IMSP7	***Despliegue sin servidor***			Implementar servicios en plataforma sin servidor
	IMSP8	***Plataforma de servicios***			Implementar servicios de forma semiautomática
	IMSP9	***Armazón de microservicios***	EL desarrollo implica mucho tiempo	¿Manejar problema *cross-cuting*?	Marco de trabajo para manejar problemas *cross-cuting*
	IMSP10	***Configuración externa***	Uso de mucha infraestructura		Externalizar configuración
	IMSP11	***Invocación remota***	Servicios que manejan peticiones	¿Comunicación?	Aplicar protocolo RPI
	IMSP12	***Mensajería***			Comunicación asíncrona
	IMSP13	***Protocolo de dominio específico***			Usar protocolo específico
	IMSP14	***Compuerta API***	Múltiples versión de producto	¿Comunicación externa?	Proveer interfaz unificada
	IMSP15	***Backend para front-end***			Compuerta API separada para clientes
	IMSP16	***Descubrimiento por cliente***	Virtualizar ambientes	¿Localizar instancia?	Registrar servicio de consultas
	IMSP17	***Descubrimiento por servidor***			Consultar en rutas
	IMSP18	***Registro de servicio***	Localización de instancias del servicio	¿Localizar una instancia?	Base de datos de instancias
	IMSP19	***Auto registro***	Registro de servicios	¿Registrar instancias?	Registrar instancia de registro
	IMSP20	***Registro de tercero***			Externalizar servicios
	IMSP21	***Quiebre de circuito***	Colaboración manejando servicios.	¿Prevenir fallas en cascada?	Invocar servicios remotos
	IMSP22	***Base de datos por servicio***	Persistencia en servicios	¿Base de datos en la aplicación?	Cada servicio con su base de datos
	IMSP23	***Bases de datos compartidas***			Servicio comparte base de datos
	IMSP24	***Saga***	Servicio con su base de datos	¿Mantener consistencia?	Secuencia de transacciones
	IMSP25	***Composición API***	Juntar datos de múltiples servicios	¿Implementar consultas?	Implementar consultas en servicios
	IMSP26	***CQRS***	Difícil consulta de datos		Implementar consultas en vistas
	IMSP27	***Eventos en codificación***	Publicación automática de eventos	¿Publicar a pesar de cambios?	Persistencia en eventos
	IMSP28	***Registro de transacciones***			Capturar cambios de transacciones
	IMSP29	***Lanzadores de bases de datos***			Usar lanzadores para capturar datos
	IMSP30	***Eventos de aplicaciones***			Insertar eventos en base de datos
	IMSP31	***Acceso de tokens***	Verificar autorización	¿Comunicar identidad?	Usar *token* seguro
	IMSP32	***Prueba de componentes de servicios***	Numerosos servicios	¿Pruebas fácil?	Pruebas en isolación
	IMSP33	***Prueba de integración de servicios***			Pruebas escritas por desarrolladores
	IMSP34	***Agregar registro***	Múltiples instancias	¿Entender comportamiento?	Agregar registro en cada servicio
	IMSP35	***Métricas de la aplicación***	¿Mínimo sobre carga?		Instrumentos para estadísticas
	IMSP36	***Auditoría de registros***	Acciones realizadas		Guardar acciones en base de datos
	IMSP37	***Trazabilidad distribuida***	Solicitar múltiples servicios		Servicios con ID único
	IMSP38	***Excepción de rastreo***	Errores manejando peticiones		Reportar excepciones
	IMSP39	***Comprobar API***	Monitoreo del sistema		Retornar estado del servicio
	IMSP40	***Registro de desarrollo***	Desarrollo frente a cambios		*No especificado*
	IMSP41	***Fragmentación de componentes***	Interfaces múltiples servicios	¿Mostrar datos?	Construir fragmentos HTML
	IMSP42	***Composición de interfaz usuario***			Construir fragmentos de interfaz
[20]	IMSP43	***Servicio de proxy Cloudify***	Modelo de dominio	¿Entradas y salidas?	Nodo simple de usuario
[21]	IMSP44	***SOA fino***	Conectividad sistemas externos	¿Realizar cambios?	Descomponer servicios
	IMSP45	***API en capas***	Múltiples capas	¿Estructuras difíciles?	Crear capas de microservicios
	IMSP46	***Orientado a mensajes***	Evitar estados rígidos	¿Replicación de datos?	Mensajería en cola
	IMSP47	***Dirigido por eventos***	Reconstruir vistas	¿Replicar eventos?	Usar abstracción común
	IMSP48	***Isolar estados***	Consistencia interna	¿Integridad de datos difícil?	Microservicio como fuente
	IMSP49	***Replicando estados***	Consistencia es requerida	Integridad de datos	Mantener única fuente
[22]	IMSP50	***Agregación de microservicios***	Mecanismos REST livianos	¿Funcionalidad?	Invocar múltiples servicios
	IMSP51	***Proxy de microservicios***	Invocar necesidad de negocio		Obtener datos de negocio en web
	IMSP52	***Microservicios en cadena***	Respuesta consolidada		Respuestas en diferentes servicios
	IMSP53	***Rama de microservicios***	Respuestas simultáneas		Cadena de invocación
	IMSP54	***Datos compartidos en microservicios***	Compartir datos		Almacenar en base de datos
	IMSP55	***Mensaje asín. en microservicios***	Mensajes REST		Mensaje compartido en cola
[23]	IMSP56	***Back-end para Front-end***	Coordinación diferentes API	¿Mejorar desarrollo?	APIs para móvil y web
	IMSP57	***Identificador de recursos uniformes***	Identificadores en microservicios	¿Diferentes objetos?	Usar datos en identificadores
	IMSP58	***Malla de servicios***	Comunicación entre servicios	¿Servicios distribuidos?	Proxy independiente
[24]	IMSP59	***Compartir base de datos***	Compartir datos	¿Sincronización?	Mantener sincronización en base de datos
	IMSP60	***Integración REST***	Integración de microservicios		Punto final del tipo REST
	IMSP61	***Mensajería***	Desacoplar mensajería		Mensaje de quiebre
[25]	IMSP62	***Estrangulamiento***	Omitir mal comportamiento	¿Estabilidad?	Proveer conexiones
	IMSP63	***Tiempo límite***	Completar petición		Tiempo límite para lectura/escritura
	IMSP64	***Hilos dedicados***	Hilos de pools		Hilos para cada servicio
	IMSP65	***Quebrar circuito***	Impacto de no acceso		Verificar disponibilidad
[26]	IMSP66	***Embajador***	Conectividad común	*No especificado*	Servicios auxiliares
	IMSP67	***Capa anti corrupción***	Implementar fachada		Traducir peticiones
	IMSP68	***Backends para Frontends***	Múltiple interfaces		Separar *back-end*
	IMSP69	***Mampara***	*No especificado*		Isolar elementos
	IMSP70	***Agregar compuerta***	Múltiples llamadas		Agregar peticiones individuales
	IMSP71	***Compuerta fuera de carga***	Servicios compartidos		Usar compuerta proxy
	IMSP72	***Compuerta ruteada***	Múltiples servicios		Enrutar peticiones
	IMSP73	***Archivo adicional***	Solicitud de funcionalidad		Desarrollar en forma separada
	IMSP74	***Estrangulador***	Servicios obsoletos		Incrementar migración

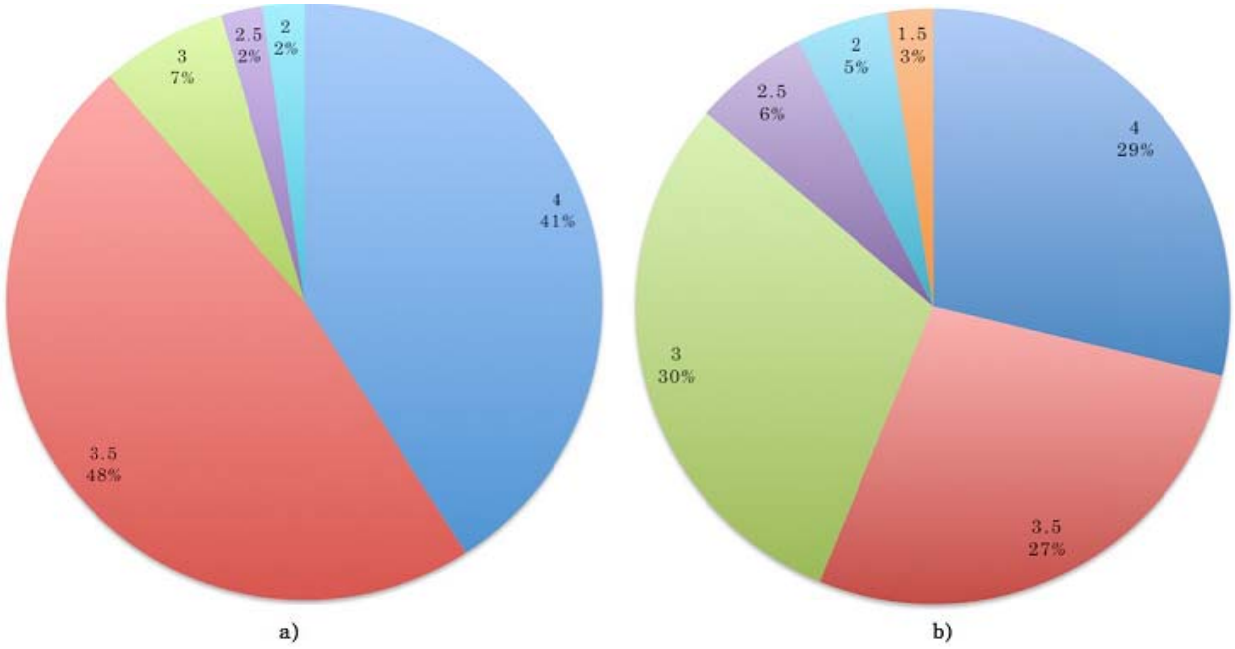

Figura 1. Distribución de los puntajes de calidad para a) patrones de arquitectura académicos y b) patrones de arquitectura industriales

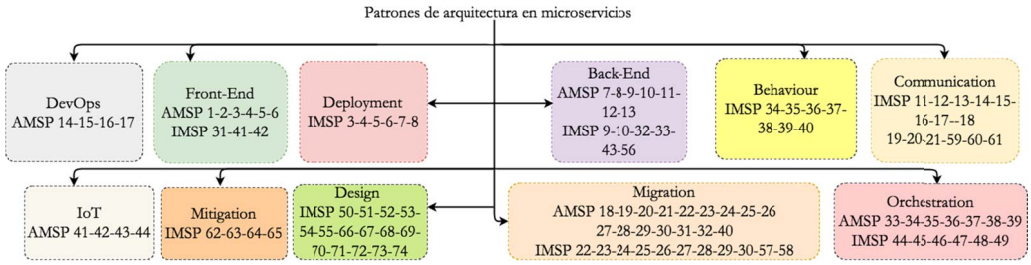

Figura 2. Taxonomía propuesta de patrones de arquitectura en microservicios

PI3: ¿Qué atributos de calidad se relacionan con micro-servicios? Debido a que por el lado de las tácticas de arquitectura no encontramos suficiente evidencia, enfocamos nuestros esfuerzos en los patrones. Para rescatar los atributos de calidad (QA) asociados con los 118 patrones de arquitectura encontrados en nuestros estudios, utilizamos técnicas de recuperación de información, como el algoritmo *Rapid Automatic Keyword Extraction* (RAKE) [31]. Después de aplicar algunos filtros manuales de las palabras clave obtenidas por el algoritmo, los QA con el puntaje más alto fueron: escalabilidad, flexibilidad, *testability*, elasticidad y rendimiento.

V. Evaluación de Calidad

En esta sección detallaremos la evaluación de calidad que se aplicó en los resultados para ilustrar aquellos patrones de arquitectura que, según nuestros criterios, satisfacen los siguientes cuatro criterios de calidad (QC) (adaptados de [32]): QC1: ¿El patrón de arquitectura describe el *contexto* en el que se encuentra?; QC2: ¿El patrón de arquitectura describe el *problema* que trata de resolver?; QC3: ¿El patrón de arquitectura describe la *solución* respecto al contexto y problema?; QC4: ¿El patrón de arquitectura está relacionado a uno o más atributos de calidad? Para la evaluación, usamos respuesta del tipo *sí*, *parcialmente* y *no*. Por ejemplo, para QC1, el criterio fue: Y (*sí*), el patrón de arquitectura describe claramente el contexto en el que se encuentra; P (*parcialmente*), el patrón de arquitectura describe parcialmente el contexto en el que se ubica; y N (*no*), el patrón de arquitectura describe vagamente el contexto.

El procedimiento de puntuación fue Y = 1, P = 0,5, *N* = 0. La Fig. 1-(a) muestra que de un total de 18 de los 44 patrones arquitectónicos cumplen nuestros criterios de calidad, lo que equivale al 48%. Sin embargo, aquellos patrones de arquitectura con bajo puntaje (entre 2 y 2.5) que son equivalentes al 4 % de los AMSP, llaman nuestra atención.

Estos patrones arquitectónicos (**AMSP35** y **AMSP37**) no muestran claramente sus contribuciones según los criterios que hemos establecido. En el lado industrial, los resultados muestran que el 29 % de los 74 patrones de arquitectura cumplen nuestros criterios de calidad (Ver Fig. 1–(b)). Encontramos que este es un porcentaje muy bajo, pero es posible tener una explicación. En nuestra revisión industrial nos dimos cuenta de que muchos autores intentan ilustrar de

manera "acelerada" las contribuciones con respecto a los microservicios (por ejemplo, **IMSP58** y **IMSP69**). Aunque estos patrones de arquitectura pasaron nuestros protocolos de revisión de filtros descritos en la Sección III, la calidad de la información mostrada por estos patrones arquitectónicos es insuficiente.

VI. TAXONOMÍA DE PATRONES DE ARQUITECTURA EN MICROSERVICIOS

En esta sección describiremos una taxonomía que hemos definido a partir de la evidencia encontrada en nuestro estudio (Ver Fig. 2). Para crear nuestra taxonomía, seguimos la estrategia propuesta por Velardi et al. [33] que es un procedimiento automatizado para lograr un factor de aceleración significativo (en nuestro caso, categorías) en el desarrollo de recursos (patrones de arquitectura) con validación y refinamiento humanos. Una vez que identificamos las categorías correspondientes, asociamos cada patrón de arquitectura a la categoría, con el objetivo de crear nuestra taxonomía final. Destacamos que las descripciones de los patrones arquitectónicos revelan que los microservicios y la virtualización han cambiado la industria, brindando agilidad e innovación en este dominio. Por otro lado, DevOps comparte enfoques comunes para el desarrollo de software, estructuras organizativas y culturas de desarrollo similares. Finalmente, hemos observado que tanto los microservicios como los patrones de arquitectura de la taxonomía son transversales a variadas áreas de la ingeniería de software.

VII. AMENAZAS A LA VALIDEZ

Usando como referencia [34], para minimizar las amenazas a la validez interna del equipo de investigación que realizó la revisión, se discutió cada artículo encontrado basado en un esquema, que tiene la siguiente estructura: ID, resumen, resumen de la introducción, descripción de la propuesta, breve descripción de las secciones del artículo, y observaciones/comentarios. Por el lado de la validez externa, hemos descartado aquellos trabajos que no han sido capaces de validar concretamente los patrones y tácticas de arquitectura. Finalmente, para reducir el sesgo de selección, tres investigadores realizaron la selección de artículos durante nuestra revisión. Dos investigadores aplicaron los criterios de inclusión y exclusión, compararon los resultados y resolvieron conflictos siguiendo las definiciones de [34]. Una tercera persona estaba a cargo de administrar la selección y participar cada vez que una resolución es difícil de hacer por los otros dos.

CONCLUSIONES

Este estudio muestra un resumen de los patrones y tácticas de arquitectura propuestas para microservicios en la academia y la industria. Para realizar la revisión, se desarrolló una planificación de revisión arrojando como resultados 13 evidencias tanto académicas como industriales. Tanto para los patrones de arquitectura en el campo académico e industrial, la revisión reveló 118 patrones. El punto anterior nos permitió comprender qué categorías se exploraron, proporcionando evidencia interesante para presentar nuestra taxonomía basada en ellas. Por el lado de las tácticas de arquitectura en la academia y la industria, no encontramos tácticas relacionadas con los microservicios, lo que nos permite saber y entender que se trata de un área inexplorada. Acerca de los atributos de calidad, encontramos que los patrones y tácticas de arquitectura están asociados con cinco atributos de calidad: flexibilidad, *testability*, elasticidad, rendimiento y escalabilidad, siendo los dos últimos los más dominantes.

AGRADECIMIENTOS

Este estudio ha sido apoyado por (1) Comisión Nacional de Investigación Científica (CONICYT) CONICYT-PCHA/Doctorado Nacional/2016-21161005 y (2) FONDECYT regular 1140408. También agradecemos la ayuda de la Armada de Chile.

REFERENCIAS

[1] J. Lewis and M. Fowler, "microservices-a definition of this new architectural term", *https://martinfowler.com/articles/microservices.html*.

[2] S. Newman, "Building microservices: designing fine-grained systems.", *O'Reilly Media, Inc.*, 2015.

[3] P. Di Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption", *IEEE International Conference on Software Architecture (ICSA)*, pp. 21– 30, 2017.

[4] H. Vural, M. Koyuncu, and S. Guney, "A systematic literature review on microservices", *International Conference on Computational Science and Its Applications*, pp. 203–217, 2017.

[5] J. F. Maranzano, S. A. Rozsypal, G. H. Zimmerman, G. W. Warnken, P. E. Wirth, and D. M. Weiss, "Architecture reviews: Practice and experience", *IEEE Software*, vol. 22, no. 2, pp. 34–43, 2005.

[6] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation", *Journal of Systems and Software*, vol. 83, no. 10, pp. 1735–1758, 2010.

[7] G. Lawton, "How microservices patterns made uber's architecture perform better", *http://www.theserverside.com/feature/How-microservices-patterns-helped-Uber-systems-perform-better*, 2017.

[8] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture", *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51, 2016.

[9] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study", *Proceedings of the 6th International Conference on Cloud Computing and Services Science CLOSER*, vol. 1, pp. 137–146, 2016.

[10] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, M. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow", *CoRR*, 2016.

[11] F. Osses, G. Márquez, and H. Astudillo, "Architectural tactics and patterns in microservices: A systematic literature review", *36 th International Conference of the Chilean Computer Science Society (SCCC)*, 2017.

[12] B. Kitchenham, "Procedures for performing systematic reviews", tech. rep., Keele University, 2004.

[13] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering", *Engineering*, vol. 2, p. 1051, 2007.

[14] K. Brown and B. Woolf, "Implementation patterns of microservices architectures", *Conference on Pattern Languages of Programs (PLoP)*, 2016.

[15] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices migration patterns", *Technical Report No. 1, TR- SUT-CE-ASE-2015-01, Automated Software Engineering Group, Sharif University of Technology*, 2015.

[16] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things", *IEEE 21st International*

Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–6, 2016.
[17] A. Messina, R. Rizzo, P. Storniolo, and A. Urso, "A simplified database pattern for the microservice architecture", *The Eighth International Con- ference on Advances in Databases, Knowledge, and Data Applications (DBKDA)*, 2016.
[18] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani,A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar, "Iot design patterns: Computational constructs to design, build and engineer edge applications", *IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 277–282, 2016.
[19] C. Richardson, "A pattern language for microservices", 2017.
[20] N. Shalom, "Building large scale services with microservices", *Cloudify*, 2017.
[21] MuleSoft, "The top six microservices patterns. how to choose the right architecture for your organization", 2017.
[22] A. Gupta, "Microservice design patterns", 2015.
[23] P. Calçado, "Patterns of microservices architecture", *http://philcalcado.com/microservices-patterns.html*, 2017.
[24] L. Majerowicz, "Integration patterns for microservices architectures: The good, the bad and the ugly", 2017.
[25] R. Dhall, "Performance patterns in microservices based integrations", 2016.
[26] M. Wasson, "Design patterns for microservices", 2017.
[27] C. Alexander, S. Ishikawa, M. Silverstein, J. R. Ramió, M. Jacobson, and I. Fiksdahl-King, "A pattern language", *University of California, Berkeley*, 1977.
[28] H. M. Chen, R. Kazman, S. Haziyev, V. Kropov, and D. Chtchourov, "Ar- chitectural support for devops in a neo-metropolis bdaas platform", *IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, pp. 25–30, 2015.
[29] L. Bass, I. Weber, and L. Zhu, "Devops: A software architect's perspective", *Addison-Wesley Professional*, 2015.
[30] Z. Xiao, I. Wijegunaratne, and X. Qiang, "Reflections on soa and microservices", *4th International Conference on Enterprise Systems (ES)*, pp. 60–67, 2016.
[31] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents", *Text Mining: Applications and Theory*, pp. 1–20, 2010.
[32] "Effectiveness matters. the database of abstracts of reviews of effects (DARE)", *United Kingdom: The University of York. https://www.york.ac.uk/media/crd/em62.pdf*, 2002.
[33] P. Velardi, A. Cucchiarelli, and M. Petit, "A taxonomy learning method and its application to characterize a scientific web community", *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 2, 2007.
[34] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and Wesslén, "Experimentation in software engineering", *Springer Science Business Media*, 2012.

Gastón Márquez es Ingeniero Civil Informático y Magíster en Ciencias de la Computación de la Universidad del Bío-Bío. Ha trabajado en empre- sas del rubro naviero y financiero. Actualmente, es docente y candidato a doctor en Ingeniería Informática en la Universidad Técnica Federico Santa María, Chile.

Felipe Osses es oficial especialista en Telecomunicaciones de la Armada de Chile, quien se encuentra estudiando el Magister en Ciencias de la Informática en la Universidad Técnica Federico Santa María, Chile. Durante su carrera naval ha desempeñado diferentes puestos en buques de la Armada, destacando su paso como Comandante de una Lancha de Patrullaje Costero.

Hernán Astudillo es doctor en Ciencias de la Computación e Información (Georgia Tech) y profesor de informática en la Universidad Técnica Federico Santa María, Chile. Ha tenido experiencia como arquitecto en empresas de consultoría tanto en Chile como en Estados Unidos. Es miembro de IFIP TC2 (Ingeniería de Software), fue presidente fundador de ArquiTIC (asociación chilena de arquitectos de tecnologías de la información) y ha representado a Chile en CLEI (Conferencia Latinoamericana de Informática)